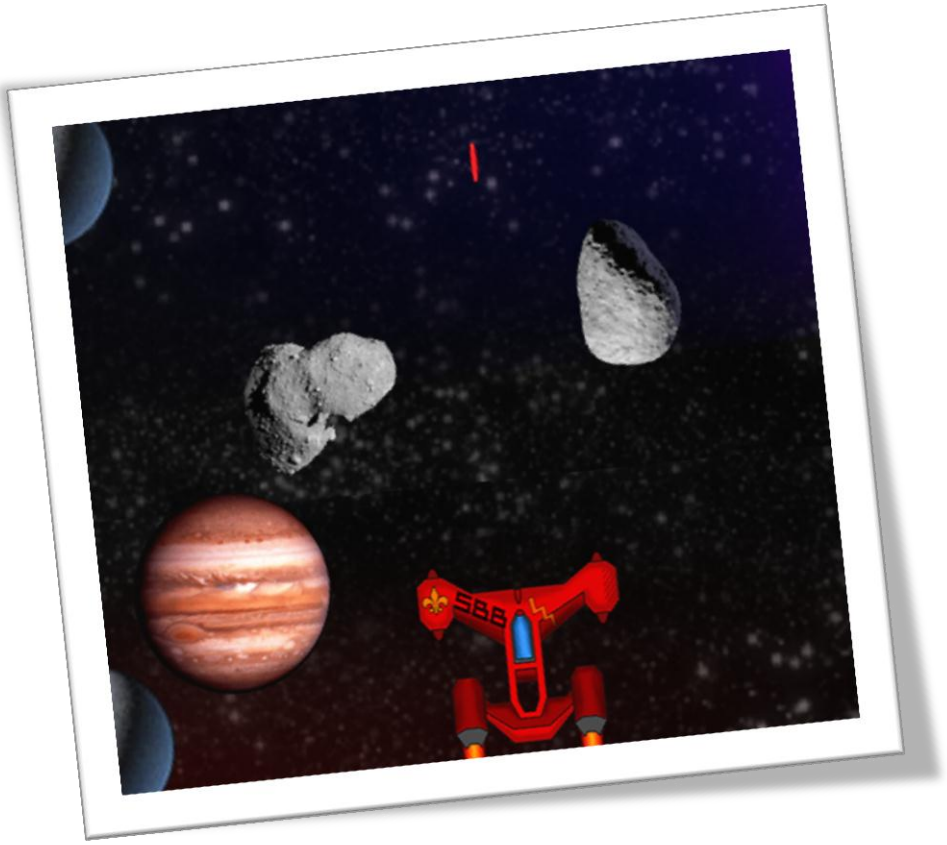
MULTIMODAL SPACE SHOOTER

RAPPORT



Interfaces multimodales
Classe i3

Stéphane Biolley
Jérémy Singy
Xavier Butty

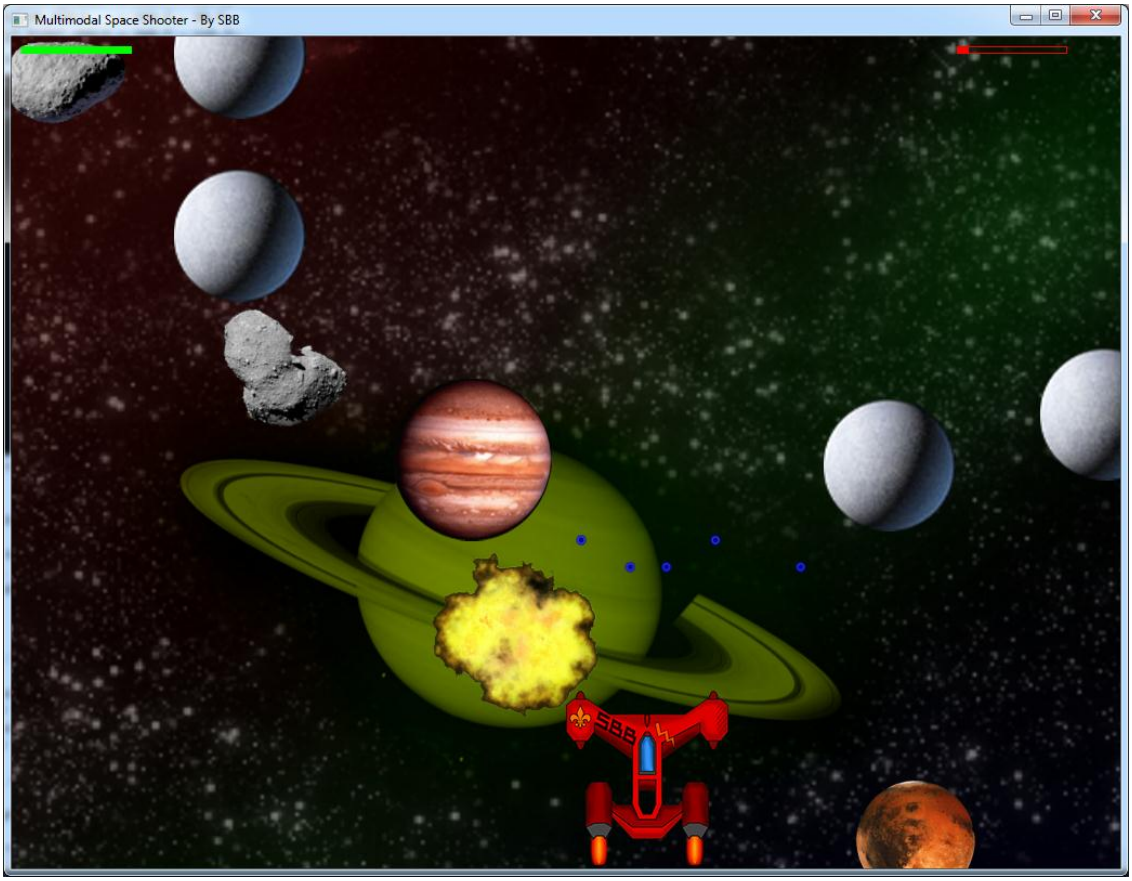
1 INTRODUCTION

Le mini-projet que nous proposons est un jeu de type « scrolling shooter » en 2D. Le joueur peut diriger le vaisseau, c'est-à-dire le déplacer de gauche à droite, et lancer différents types de projectiles pour détruire des objets dans le monde et ainsi gagner des points.

2 DESCRIPTION DU JEU

Dans le monde, le vaisseau se déplace automatiquement à vitesse constante. Le joueur peut alors diriger celui-ci dans le monde en se déplaçant lui-même sur la gauche ou la droite du capteur (position globale du joueur). Il peut ainsi se déplacer pour esquiver d'éventuels objets sur son passage.

Le joueur peut aussi lancer différents types de missiles avec une commande vocale. Cette commande est multimodale. En effet, selon la position de ses bras, différents missiles peuvent être lancés. Si les deux bras sont le long du corps, un simple missile est envoyé en avant. Si le bras gauche est tendu, une rafale de plusieurs missiles est envoyée vers la gauche du vaisseau, et de manière équivalente pour le bras droit. L'interface est donc multimodale avec la voix et le geste.



3 MODALITÉS D'INTÉRACTIONS

Ce chapitre a pour but de décrire les principales interactions du jeu ainsi que de les modalités utilisées pour chacune de celles-ci. Les périphériques de chacune des modalités seront aussi présentés.

3.1 Les différentes interactions possibles

3.1.1 Tir simple

Représente un simple tir droit du vaisseau.

3.1.2 Tir spécial

Salve de 6 missiles se déplaçant plus lentement que le tir simple.

3.1.3 Déplacer le vaisseau

Déplacement horizontal du vaisseau.

3.2 Modalités utilisées

3.2.1 Voix

La voix est utilisée via l'une de ses caractéristiques qui est le volume. L'utilisateur doit faire un son suffisamment puissant afin de déclencher une action. Pour déclencher deux actions de suite, une baisse du volume suffisante, doit être faite avant que celui-ci ne remonte. Il est donc impossible de faire une suite d'actions via un son continu avec un volume inchangé, l'événement se produit lors d'un passage d'un seuil défini dans l'application.

Nous avons choisi d'utiliser la voix de cette manière simple plutôt que de la reconnaissance vocale, car après les premiers tests nous nous sommes rendu compte que le temps de « processing » nécessaire par les bibliothèques de reconnaissance vocal était beaucoup trop long (de l'ordre de la seconde) pour notre application ludique.

3.2.1.1 Utilisation

3.2.1.1.1 Tir

Les tirs sont déclenchés via un son, comme par exemple un « *TaTa* » ou encore « *PiouPiou* ». Cette modalité est associée à la seconde modalité : le geste, pour définir le type de tir.

3.2.1.1.2 Navigation dans les menus

Pour sélectionner une option dans les menus du jeu, on peut utiliser la voix de la même manière que le tir (c'est le changement de volume qui est déterminant).

3.2.1.2 Périphérique

Il s'agit ici d'un simple micro. Pour plus d'efficacité, il est recommandé d'utiliser un micro-casque pour éviter tout bruit parasite.

3.2.2 Gestes

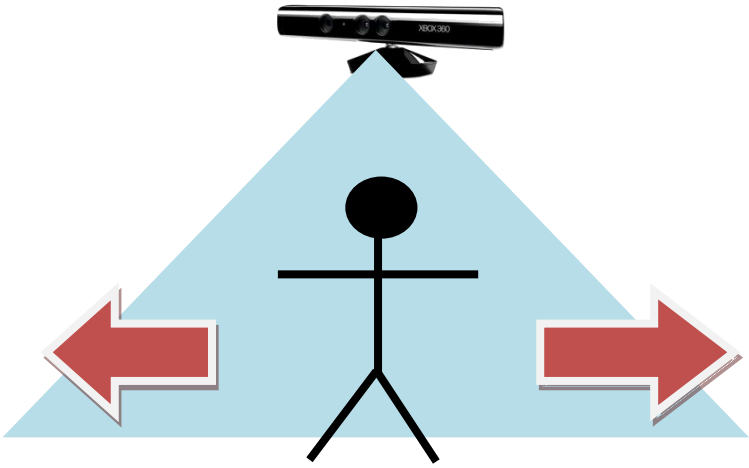
La deuxième modalité utilisée est le geste. Celle-ci peut être divisée en deux parties:

- La position du corps (plus précisément le centre de notre corps)
- La position des bras (positions des mains par rapport au centre du corps)

3.2.2.1 Utilisation

3.2.2.1.1 Déplacement

Le déplacement horizontal du vaisseau se fait en nous déplaçant simplement de gauche à droite. Bien sûr, ce mouvement doit être fait en phase du périphérique d'entrée lié destiné à la reconnaissance des mouvements et dans sa zone de visibilité.

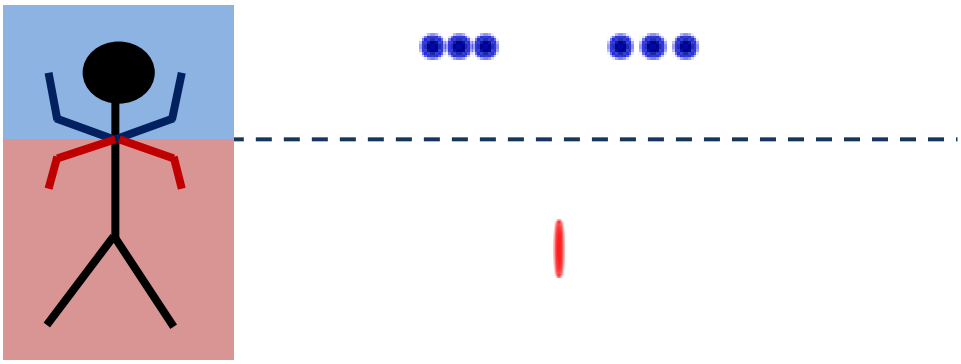


3.2.2.1.2 Définition du type de tir

Le jeu offre deux types de tir différents. Ceux-ci dépendent de la position de nos bras. Si nos bras sont dirigés vers le haut et que l'on utilise notre voix pour tirer, cela déclenchera un tir spécial. Dans le cas contraire, le vaisseau fera un tir simple.

Cette position haute des bras pour le tir spécial évite qu'il soit utilisé trop souvent. En effet l'utilisateur se fatiguera vite s'il abuse de cette arme.

La figure ci-dessous monte les deux zones différentes qui définissent le type de tir selon la position des deux mains.



3.2.2.1.3 Navigation dans le Menu

Une fois le mode multimodal choisi, l'utilisateur peut naviguer dans les menus en déplaçant sa main droite. Le curseur réagit alors à cette modalité selon la position de la main.

3.2.2.2 Périphérique

Le capteur de mouvement Kinect de Microsoft est utilisé afin de détecter la position du joueur est celle de ses bras.

3.2.3 Modalités d'entrée simples

Outre les modalités vues précédemment, le jeu offre aussi la possibilité d'utiliser une interface basique commune : le clavier et la souris, pour les personnes ne disposant pas des périphériques tels que le microphone ou la Kinect.

Ce mode a aussi été utile pour la phase de développement car nous ne disposions que d'un seul capteur Kinect pour le groupe. Nous avons choisi de le garder pour être le plus générique possible

3.2.3.1.1 Tir

Un tir peut être lancé par une pression sur la barre d'espace qui a le même effet que l'utilisation de la voix.

3.2.3.1.2 Tir spécial

Le tir spécial se fait au clavier en combinant la touche «↑» et une des modalités pour le tir, que ce soit la barre d'espace ou en utilisant la voix.

3.2.3.1.3 Déplacement

Avec le clavier, le vaisseau peut être déplacé via les touches «← » et «→».

3.2.3.2 Périphérique

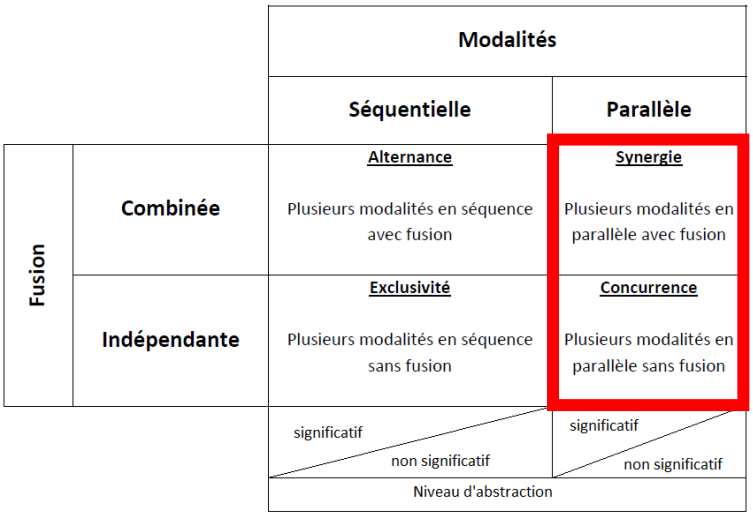
Le périphérique est naturellement le clavier et la souris pour naviguer dans le menu.

3.3 Interactions selon les différents modèles

3.3.1 CASE

Si l'on observe les types de communications multimodales côté machine selon le modèle CASE, on va travailler en parallèle au niveau des modalités. Quant à la fusion, elle sera combiné pour les tirs spéciaux (gestes/touches et voix/touches) et indépendante pour le déplacement.

Par rapport au tableau ci-dessous, on voit donc que nos interactions de tirs sont synergiques et que le déplacement est concurrent.



3.3.2 CARE

Ci-dessous chacune des interactions est définie du point de vue de son utilisation au niveau du modèle CARE.

3.3.2.1 Tir

Propriété d'utilisation : Equivalence

Pour le tir, l'utilisateur peut utiliser soit le clavier avec la barre d'espace soit la voix.

3.3.2.2 Tir spéciaux

Propriété d'utilisation : Complémentarité et Equivalence

Les modalités utilisées pour ces interactions sont complémentaire et équivalente. En effet, deux modalités doivent être utilisées ensemble pour faire un tir spécial mais on a le choix entre plusieurs modalités (gestes/touches + voix/touches).

3.3.2.3 Déplacement

Propriété d'utilisation : Equivalence

Le déplacement du vaisseau peut se faire via deux modalités : le geste et le touches du clavier.

4 TECHNOLOGIES UTILISÉES

4.1 Langage de programmation C++

Notre choix s'est porté sur le C++ du fait de la compatibilité avec OpenNI et SFML ainsi que de nos intérêts personnels.

4.2 IDE Microsoft Visual Studio 2010

Travaillant tous sur un système Windows, indispensable pour les drivers que nous utilisons, nous avons utilisé Visual Studio. En effet, les binaires (fichiers .lib et .dll) de OpenNI sont compilés sur Windows pour être utilisés avec Visual Studio De plus, l'outil de développement de Microsoft est fourni gratuitement aux étudiants de l'école et possède une extension pour l'utilisation facile avec le gestionnaire de version Git.

4.3 Git et serveur Github.com

4.3.1 Description

Nous avons utilisé Git suite aux expériences positive que nous avons eu avec cette outil de gestion de version dans le cadre d'autres projets. Le site de « *social coding* » github a été utilisé comme dépôt pour nos sources.

4.3.2 Lien du projet sur Github

https://github.com/jeremysingy/MultimodalSpaceShooter

4.4 Bibliothèque SFML

4.4.1 Description

SFML (Simple and Fast Multimedia Library) est une API multimédia libre écrite en C++ offrant un accès bas niveau simplifié pour les graphismes 2D, l'audio, les événements. Elle est orientée objet, portable et flexible par sa fragmentation en petits modules. Son intégration est facile et elle est dispose de plusieurs « bindings » pour des langages tels que Python, C ou encore .NET.

4.4.2 Motivations d'utilisation

Notre choix s'est porté sur cette bibliothèque du fait de sa simplicité et richesse. Le langage choisi pour le développement de notre jeu étant le C++, cela s'accordait parfaitement avec la décision de prendre SFML qui a déjà été beaucoup utilisée pour des jeux du même type que le nôtre.

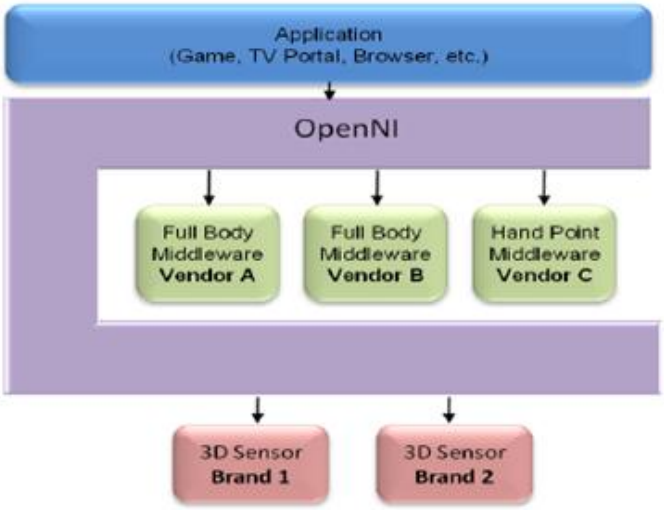
4.4.3 Lien

http://www.sfml-dev.org/index.php

4.5 OpenNI

4.5.1 Description

OpenNI signifie « Open Natural Interaction ». C'est un Framework open source multi-langage et multiplateforme se chargeant de crée le pont, la communication entre différents type de senseurs (caméra 2D/3D, microphones, …) et les « middlewares » chargés de traiter les données de ceux-ci. En résumé, c'est une architecture modulaire fournissant les interfaces entre les senseurs et modules de traitements (middleware).



4.5.2 Motivations d'utilisation

Notre utilisation de ce Framework vient du fait que la société PrimeSense met à disposition des drivers pour la Kinect ainsi que ses modules pour OpenNI en open-source. Cela simplifie l'utilisation de différentes fonctions de la Kinect et permet l'utilisation de NITE.

4.5.3 Lien

http://www.openni.org/

4.6 NITE

4.6.1 Description

En dehors des drivers de la Kinect et de ses modules pour OpenNI, PrimeSense met aussi à disposition un lot de « middlewares » permettant justement d'être utilisés avec OpenNI. NITE offre des algorithmes de traitements offrant par la suite un accès à des informations intéressantes telles que la position des différents parties du corps ou encore le nombre de personnes présentes devant la caméra.

4.6.2 Motivations d'utilisation

Ce choix s'est fait rapidement du fait de la puissance de NITE et des contrôles qu'il offre par la suite. Cela permet de simplifier nettement la puissance de la Kinect pour tout ce qui est gestuel.

4.6.3 Lien

http://www.primesense.com/?p=515

4.7 TinyXML

4.7.1 Description

TinyXML est une petite bibliothèque pour parser des fichiers XML en C++. Elle est particulièrement facile d'intégration puisque le code source (réparti dans quatre fichiers) peut être directement inclus dans l'application. TinyXML fournit une interface DOM pour la lecture, la création et la modification des fichiers XML.

4.7.2 Motivations d'utilisation

Ce parseur nous est utile pour lire des fichiers XML représentant les niveaux. Au vue de la tâche à réaliser, la taille et la simplicité de ce parseur convenait parfaitement à nos besoins.

4.7.3 Lien

http://www.grinninglizard.com/tinyxml/

5 CONCEPTION

5.1 Architecture de l'application

Notre application a été développée de façon à être relativement modulaire. Elle est séparée en plusieurs sections de code définies (réparties dans des dossiers différents) qui sont détaillées ci-dessous. Le diagramme de classe est fourni en annexe dans le fichier *class_diagram.png*.

5.1.1 Partie **/core**

Cette partie contient les classes de base nécessaire au jeu, comme la classe Game qui contient la boucle principale du jeu ou encore le moteur audio.

5.1.2 Partie **/entities**

Contient toutes les entités du jeu. Une entité représente un objet lors de la phase de jeu, comme le vaisseau, les planètes, les missiles ou encore les explosions. Chaque entité peut se mettre à jour au moyen de la méthode `update()` et se dessiner au moyen de `draw()`. Une classe gère toutes ces entités et s'occupe de les mettre à jour et de les dessiner à chaque tour de boucle, la classe `EntityManager`.

5.1.3 Partie **/gui**

Contient des objets de bases d'interfaces utilisateur dont nous avons eu besoin (SFML n'est pas une bibliothèque de GUI). Nous avons créé des boutons aussi que des menus ou encore des barres de progression, directement intégrables dans le jeu. Tous ces éléments dérivent de la classe `Widget`.

5.1.4 Partie **/input**

Contient toute la gestion des événements utilisateurs, y compris les événements multimodaux. La classe `GestureManager` encapsule la bibliothèque OpenNI alors que la classe `VolumeRecorder` permet de connaître le volume actuel du micro (au moyen de SFML). Ces modalités sont regroupés et si besoin fusionnées dans la classe `MultimodalManager` qui est accessible aux autres objets.

5.1.5 Partie **/loaders**

Contient les gestionnaires pour le chargement des ressources comme les images, les sons ou encore les niveaux. Les images et les sons sont si besoin chargés, et ensuite stockés dans un type « map » avec comme clé le nom du fichier. Ainsi, tous les objets ayant besoin d'une image peuvent facilement avoir accès à ce gestionnaire qui évite d'avoir plusieurs images du même type chargées en mémoire, et qui gère le chargement de manière transparent. Concernant le chargement des niveaux, Les entités sont mises dans une file de priorité triée selon leur temps d'arrivé, et à chaque étape du jeu cette file peut être directement requêtée pour vérifier si une nouvelle entité doit être crée.

5.1.6 Partie **/scenes**

Contient toutes les scènes du jeu. Les scènes représentent un état ou un écran du jeu. Par exemple l'écran d'introduction, la phase « in-game », le menu pause, l'écran « Game Over », etc. Toutes ces scènes sont gérées à travers le `SceneManager` qui permet de définir la scène actuelle et de mettre à jour et dessiner celle-ci de manière transparente.

5.1.7 Partie **/utils**

Contient quelques codes utilitaires comme la conversion d'angles de degrés en radians, ou encore des fonctionnalités ajoutées à SFML, comme les sprites animés (classe `AnimatedSprite`) qui gèrent facilement un sprite ayant une animation de plusieurs images (utilisé pour l'explosion). En programmation de jeu, un « sprite » signifie la partie graphique d'une entité (position courante, image utilisée, etc.). Cette partie de l'application contient aussi le code de la bibliothèque TinyXML discutée dans le chapitre précédent.

5.2 Design Patterns utilisés

5.2.1 Pattern singleton

La classe de base de notre jeu (Game) utilise le pattern le pattern Singleton. En effet, on ne peut avoir qu'une instance de Game à la fois. Chose logique du fait qu'un seul jeu se déroule à la fois.

Concernant l'accès aux différents moteurs généraux comme `AudioEngine`, `EventManager`, `EntityManager`, etc. nous avons utilisé un pattern utilisé couramment dans le développement de jeux vidéo qui consiste à ce qu'ils soient accessibles via des fonctions libres. Ceci permet d'éviter les dépendances complexes pour des objets qui auront presque tous accès à ces moteurs (par exemple toutes les entités peuvent jouer un son, et aussi les phases de menu pour réagir au clic).

Ceci permet d'éviter les problèmes connus des singletons qui cassent toute la réentrance et qui fournissent un ordre de construction indéterminé qui conduit souvent à des bugs complexes. Ici, tous nos managers ont leur durée de vie gérée par la classe `Game` qui les construit dans un ordre déterminé.

5.2.2 Pattern Observer

Pour les événements générés dans notre application, nous utilisons le pattern Observer. Toutes les classes qui doivent écouter les événements de base (comme clavier et souris) dérivent de la classe `EventListener` et redéfinissent la méthode `onEvent()`. Les classes qui doivent écouter les événements multimodaux dérivent de la classe `MultimodalListener` et redéfinissent les méthodes `onTrackingStateChanged()` pour être notifiés selon les états du tracking de l'utilisateur, ou `onMultimodalEvent()` qui réagit aux changements de volume.

Les entrées temps-réel peuvent directement être requêtées sur la classe `MultimodalManager` qui fournit des méthodes pour récupérer la position actuelle de l'utilisateur, la position de ses mains, ou le volume capté dans le micro.

5.2.3 Pattern State

Une version de ce pattern est utilisée pour la gestion des scènes. En effet, chaque scène représente un état du jeu, et le `SceneManager` gère la machine d'état. Les états peuvent être changés au moyen de la méthode `changeCurrentScene()`.

5.2.4 Fichier XML de description de monde

Il est possible de créer ses propres mondes. Ceux-ci sont décrits dans un fichier XML nommé sample.xml. Ce fichier se situe dans le dossier « worlds ».

```
<?xml version="1.0" ?>
<world>
    <background image="background.png" speed="80" />
    <enemies>
        <asteroid image="asteroid1.png" xCoordinate="100" speed="100" angle="-30" rotationSpeed="10" time="1" />
        <asteroid image="asteroid2.png" xCoordinate="750" speed="90" angle="50" rotationSpeed="-20" time="1.5" />
        <planet image="planet1.png" xCoordinate="950" speed="65" time="2.5" />
        <planet image="planet2.png" xCoordinate="350" speed="80" time="3" />
        <planet image="planet3.png" xCoordinate="780" speed="150" time="3.5" />
        <planet image="planet1.png" xCoordinate="750" speed="95" time="4" />
        <asteroid image="asteroid2.png" xCoordinate="750" speed="80" angle="95" rotationSpeed="15" time="4.5" />
        <planet type="planet" image="planet1.png" xCoordinate="150" speed="20" time="5" angle="270" />
        <planet image="planet4.png" xCoordinate="550" speed="155" time="6" />
        <asteroid image="asteroid1.png" xCoordinate="50" speed="50" rotationSpeed="6" time="6.5" />
        <planet image="planet1.png" xCoordinate="150" speed="135" time="7" />
        <planet image="planet5.png" xCoordinate="650" speed="130" time="10" />
        <planet image="planet3.png" xCoordinate="850" speed="110" time="11" />
        <planet image="planet4.png" xCoordinate="450" speed="250" time="13" />
        <asteroid image="asteroid1.png" xCoordinate="50" speed="90" angle="-50" rotationSpeed="-30" time="13.5" />
        <planet image="planet5.png" xCoordinate="350" speed="200" time="13.5" />
        <planet image="planet2.png" xCoordinate="250" speed="85" time="14" />
        <planet image="planet1.png" xCoordinate="150" speed="110" time="15" />
        <planet image="planet3.png" xCoordinate="20" speed="95" angle="270" time="15.5" />
        <asteroid image="asteroid2.png" xCoordinate="900" speed="90" angle="40" rotationSpeed="-30" time="13.5" />
         </enemies>
</world>
```

5.2.4.1 background

- Image
  - Spécification de l'image utilisée en tant que fond au jeu.
- Speed
  - Spécification de la vitesse de défilement du fond.

5.2.4.2 asteroid

- Image
  - Spécification de l'image correspondant à l'astéroïde.
- xCoordinate
  - Coordonnée en X d'apparition de l'astéroïde.
- Speed
  - Vitesse de déplacement de l'astéroïde. [pixels par secondes]
- Angle
  - Angle de défilement de l'astéroïde. [degrés]
- rotationSpeed
  - Vitesse de défilement de l'astéroïde. [degrés par secondes]
- Time
  - Temps auquel apparait l'astéroïde. [secondes]

5.2.4.3 planet

- Image
  - Spécification de l'image correspondant à la planète
- xCoordinate
  - Coordonnée en X d'apparition de la planète
- Speed
  - Vitesse de déplacement de l'astéroïde [pixels par seconde]
- Time
  - Temps auquel apparait la planète [secondes]

6 PROCÉDURE INSTALLATION

La séquence d'installation doit être suivie dans cet ordre précis.

6.1 Installation de OpenNI

- Lancer **OpenNI-Bin-Win32-v1.0.0.25.exe**
- Choisir le répertoire de destination de l'installation
- Cliquer sur « Install »
- Cocher la case « Toujours faire confiance aux logiciels provenant de Prime Sense Ltd. »
- Cliquer sur « Installer »
- Cliquer sur « Close »

6.2 Installation des drivers

- Décompresser le fichier « **avin2-SensorKinect-0124bd2.zip** »
- Dans le dossier décompressé, aller dans Bin et exécuter « SensorKinect-Win32-5.0.0.0.exe » pour démarrer l'installation
- Cliquer sur « Next »
- Cliquer sur « Install »
- Si un message apparaît concernant la sécurité de Windows, cliquer sur « Installer ce pilote quand même »
- Cliquer sur « Close »
- Aller dans le dossier →Plateform→Win32→Driver
- Lancer l'exécutable correspondant au système d'exploitation
- Cliquer sur « Suivant »
- Cliquer sur « Terminer »

6.3 Installation de NITE

- Exécuter NITE-Win32-1.3.0.17.exe
- Cliquer sur « Exécuter »
- Lire les conditions d'utilisation et cliquez sur « I Agree »
- Sélectionner le répertoire d'installation de NITE et cliquez sur « Next > »
- Insérer la clé générique suivante (il s'agit d'une clé officielle et gratuite) :
    - 0KOIk2JeIBYClPWVnMoRKn5cdY4
- Cliquer sur « Close »

6.4 Lancement de l'application

A présent que tous les drivers nécessaires ont été installés, il suffit de lancer l'exécutable de l'application.