BERNARD Quentin
FELTZ Ludovic

Escape IR

Documentation Développeur

Cette documentation a pour but de présenter l'ensemble du projet, en développant notamment la partie conception. Ainsi, nous commencerons par présenter le besoin initial, ainsi que le cahier des charges associé avant de rentrer dans le vif du sujet, qui représente le scénario de cette évasion. En effet, envoyé par l'infâme commandant Sirud en mission suicide sur Jupiter, nous avons été capturés pendant 5 semaines sur cette planète, ou nous n'avons guère pu dormir. C'est l'horrible et sinistre compte. Xaroff, qui dispose ni plus ni moins qu'une immense float de Mune, qui nous a détenu pendant tout ce temps. Pour échapper à ce piège, nous avons ruser, coder, et architecturer un Shoot'em Up spatiale représentant notre fuite pour revenir chez nous. C'est ainsi après 5 semaines de travail intense, avec en moyenne 6 heures hebdomadaires, que nous avons réussis à réaliser ce pari insensé : Finir le projet, et revenir à temps sur la planète Mars pour le concert de Johnny Halliday, mais cela, c'est une autre histoire...

Plan :

I°) Besoin et cahier des charges

Ce projet de Java avancé réalisé en binôme a pour but de mettre les compétences acquises au cours de ce premier semestre à l'épreuve. Il nous amènera à structurer un gros projet et à se répartir les taches équitablement au sein du binôme. A mettre au point des techniques de reconnaissance de forme simple, à apprendre à utiliser la libraire JBox2D etc…

II°) Scénario

Dans un Univers lointain, ou lignes de code rime avec nuit blanche, deux jeunes étudiants, dans l'espoir de décrocher une maigre reconnaissance, s'en vont avec leur bataillon pour une mission sur Jupiter, mais dès leurs arrivé tout bascula.

Arrivé sur place plein de confiance, l'attaque des premiers partiels décima une grande partie de leur équipe et de leurs équipements. Il ne pouvait plus compter que sur eux même ! Et c'est alors qu'ils élaborent un projet leur permettant de quitter cette planète, même s'il allait leurs falloir beaucoup de sacrifices dans l'élaboration de celui-ci et presque sombrer dans la folie.

C'est ainsi qu'est née le projet Escape_IR !

III°) Conditions de développement

Dès le début du projet, nous avons mis en place un ensemble de techniques nous permettant d'être plus efficace. Aussi bien au niveau communication (la plupart du développement se faisant de nuit il est difficile de pouvoir

Outils utilisé

Communication avec TeamSpeak : un logiciel de conversation distante a été installée sur chacune de nos machines personnelles, pour nous permettre de rester en communication durant toute la conception du projet. Cela a permis de mieux se répartir le travail, et la ou un cerveau bloquait, deux ont sus résoudre les nombreuses difficultés rencontrés sur ce projet.

Partage d'écran avec TeamViewer : nous avons voulus avoir une vision partagée des écrans, pour pouvoir réaliser des phases d'extrême programming, mais cela à distance par le biais de TeamViewer, un logiciel permettant cette accès distant.

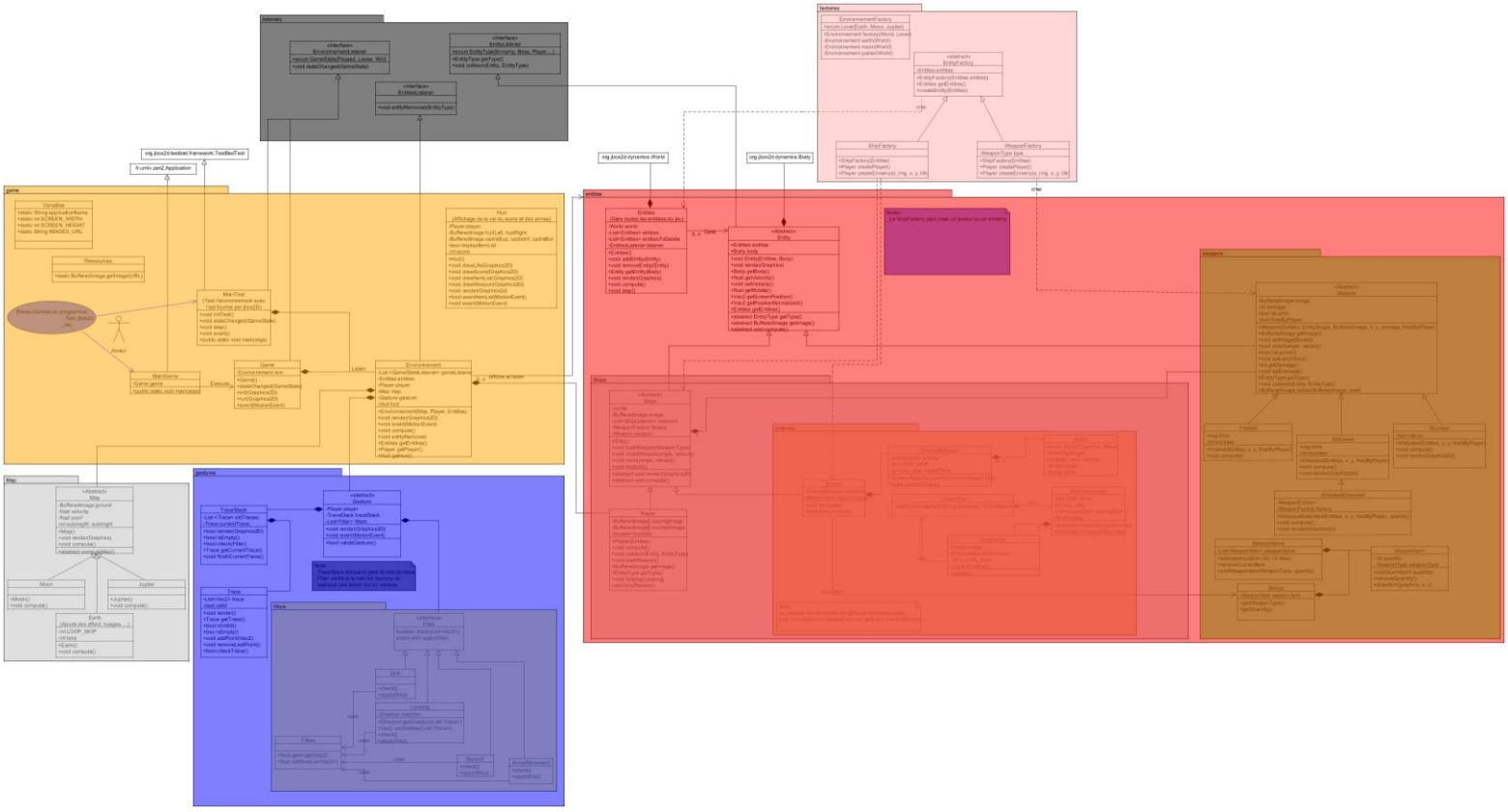
Partage du code source avec GitHub : Nous avons voulus utiliser un logiciel de gestion de versions différent de celui utilisé l'année dernière, donc nous avons mis notre projet sur GitHub. Qui a de plus la particularité d'être libre (comme nos sources)

Création et modification graphique avec Gimp : Pour la création visuelle nous avons utilisé le logiciel libre Gimp.

Ces nouvelles technologies nous auront permis de nous construire un véritable environnement pour travailler avec un minimum de confort, dans cette prison de 25 pouces.

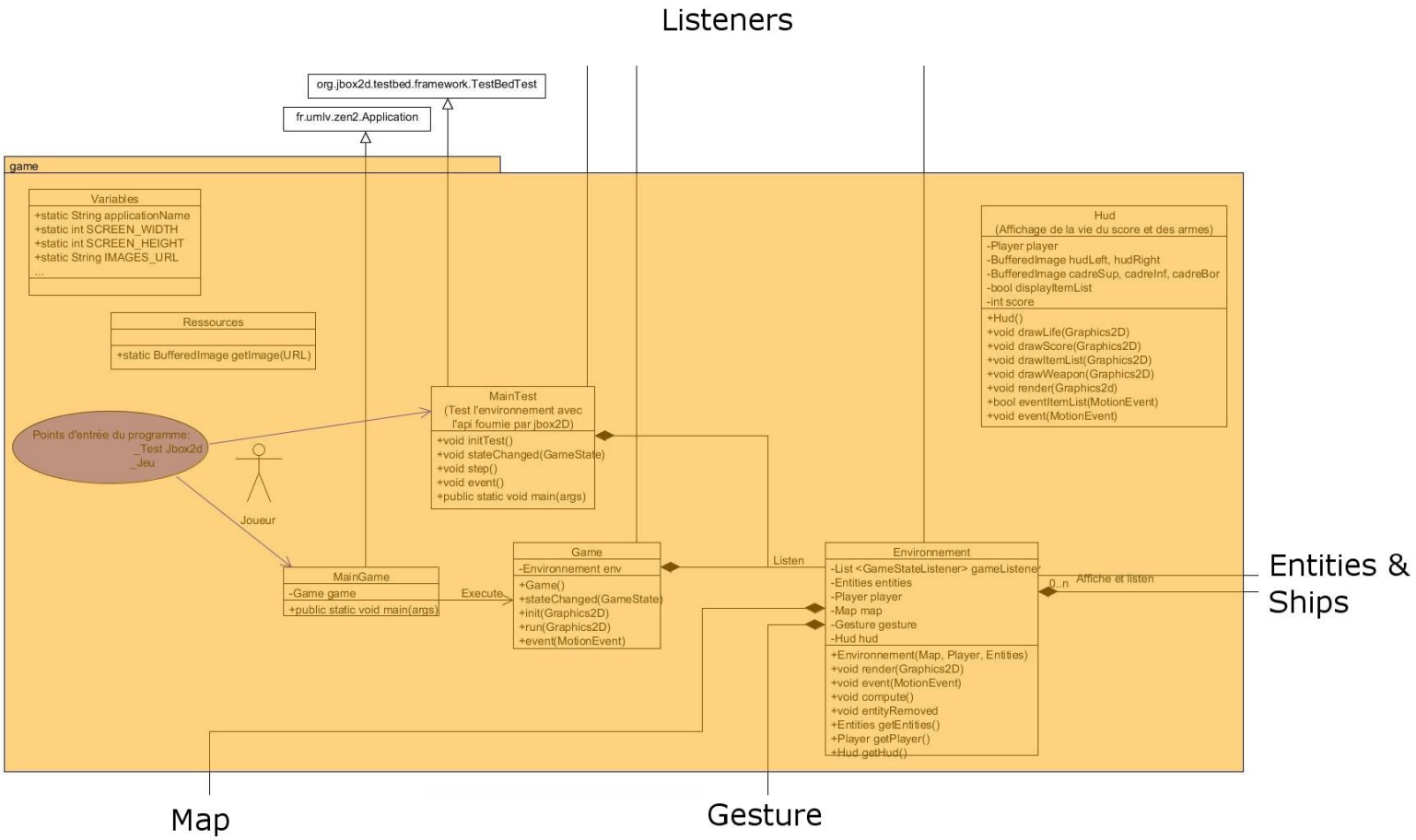
IV°)Architecture

Le projet dans son ensemble



(Détaillé dans la suite)

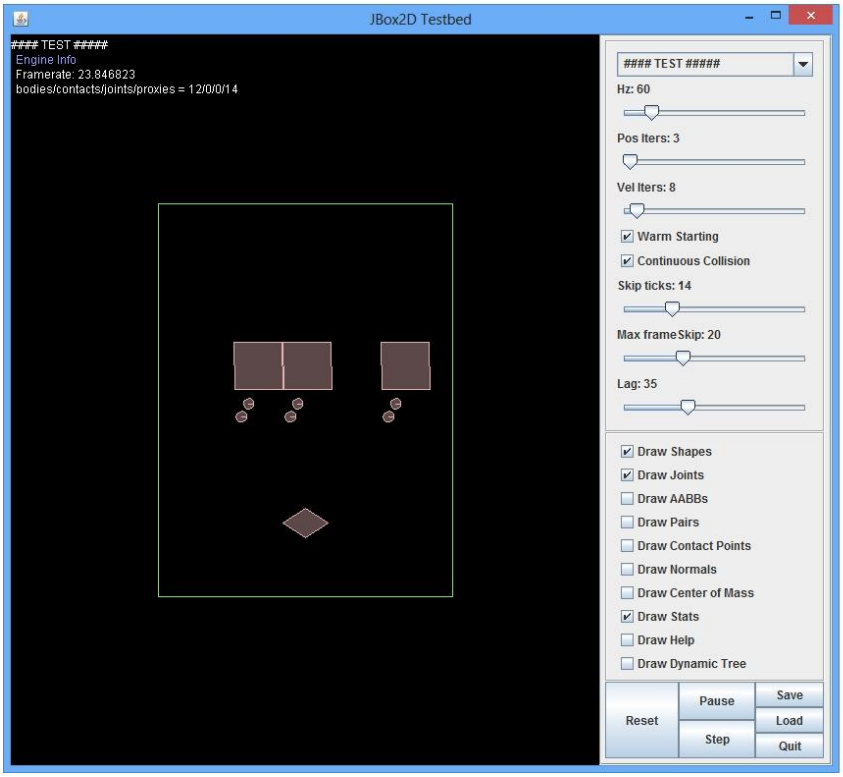
Package principal

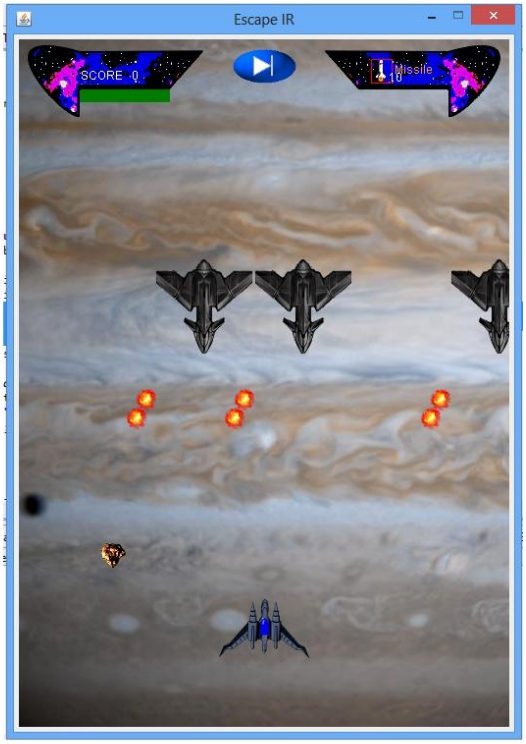


Le package principal « **game** » est le package qui gère toutes les autres classes.

Les points d'entrées.

Il comporte deux points d'entrée : **MainGame** qui permet de lancer le jeu avec un rendu graphique et le **MainTest** qui permet de lancer le jeu dans l'environnement de **JBox2d**. Cette configuration nous a permis de débugger plus facilement les erreurs qui auraient pu être difficile à détecter avec l'interface de jeux. On peut grâce à l'interface de **JBox2d** afficher ce qui se trouve en dehors de la zone d'affichage ainsi que les collisions.





Ces deux points d'entrée utilisent la même classe Game ce qui permet d'exécuter indifféremment le jeu dans **JBox2d** ou dans l'interface graphique par défaut. La seule différence étant que l'exécution par l'interface de **JBox** de lance pas la méthode **Game**.**render**() qui se charge de l'affichage des différents éléments.

Ces points d'entrée se chargent donc d'appeler à intervalle régulier les méthodes **render**() et **compute**() de la classe **Environnement**. Ces entrée gère ainsi les évènements tel que la mort du joueur ou la fin du niveau et permettent le changement de niveau et l'affichage de l'histoire.

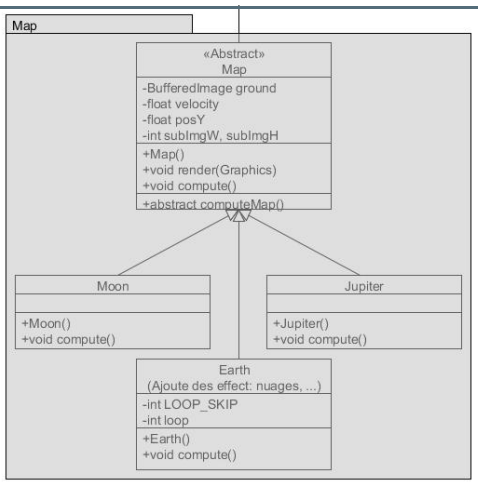
L'environnement

L'environnement est la classe maitresse du projet. C'est elle qui gère la **Map**, le **Joueur**, les **Entités**, les **Ennemies**, ainsi que la détection des **Gesture**. Elle écoute directement la classe gérant toutes les entités (**Entities**) afin de détecter lorsque le joueur ou le boss sont tué.

On pourrait penser que gérer toutes ces classe rend l'environnement compliqué à appréhender mais en fait cette classe ne se charge que d'écouter les évènements renvoyé par les **Entities** et appelle seulement les méthodes **render**(), **event**() et **compute**() de ses sous classe.

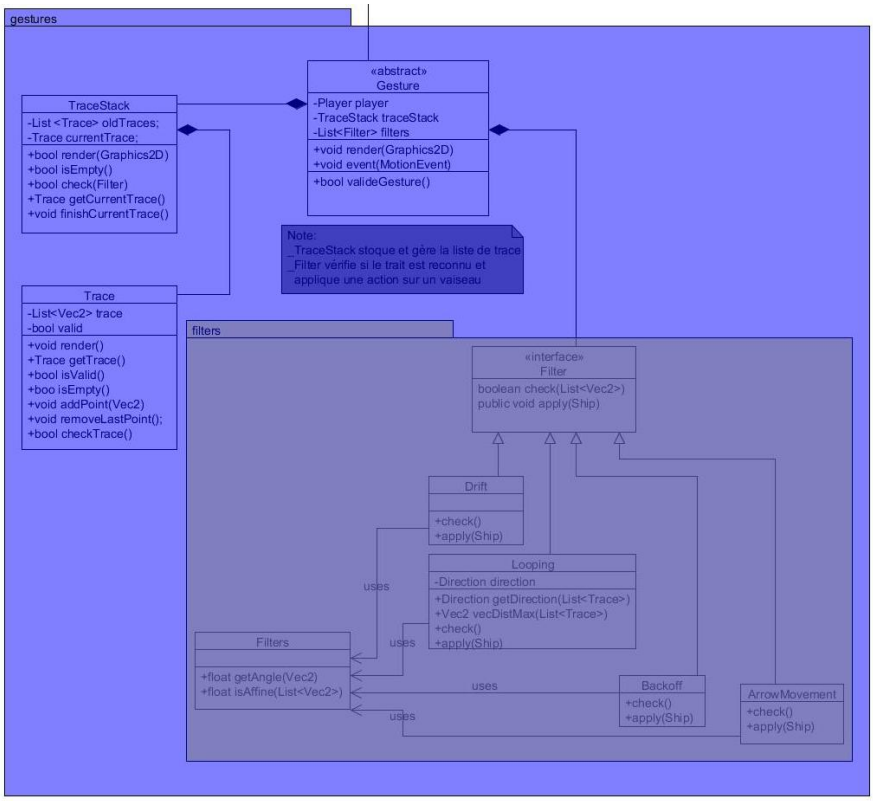
L'environnement est construit par l'**EnvironnementFactory**.

La Map



Affiche seulement un background défilant et positionne aléatoirement des effets dessus (nuages, météorites, …)

Les Gesture

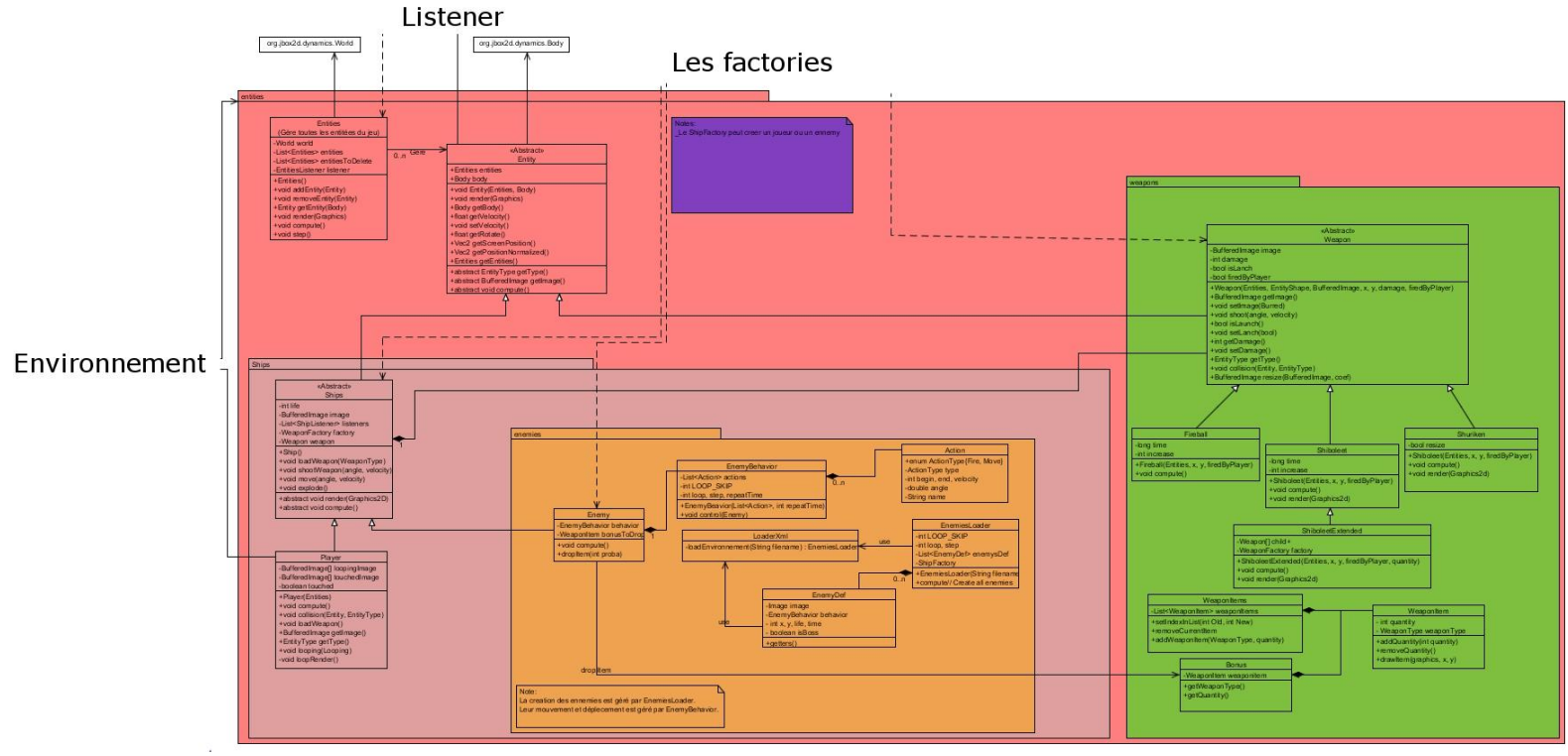


La classe **Gesture** permet de détecter les différents motifs réalisé à la souris. Si elle valide une action elle contrôle directement le joueur (mouvements, looping, tirer, …)

Pour vérifier qu'un mouvement est correct elle utilise une liste de **Filter**

Cette classe gère aussi l'affichage de la « **trace** »

Les Entities



- La classe **Entities** est la classe qui gère le world de JBox2d elle contient une liste d'**Entity.** Cette classe se charge d'afficher et de calculer les déplacements (appel des méthodes **render** et **compute**) de toutes les **Entity,** et de détecter les collisions entre **Entity.**

- Une **Entity** est une classe abstraite qui contient un **Body** de JBox. Cette classe permet d'afficher seulement une entité à l'écran, elle contient donc l'image à afficher (le **body** est créé à la taille de l'image). Elle permet d'afficher et de contrôler une **Entity** de façon indépendante (**setVelocity**(), **setRotation**(), **setDamping**(), …)

  Les objets ayant besoin d'être affiché et de détecter les collisions héritent donc de la classe **Entity : Weapon**, **Ship**, **Player**, **Enemies, Boss**.

  Les **Entity** possèdent un ou plusieurs **Listener** qui écoute en permanence les collisions.

Les Ships et classe hérités

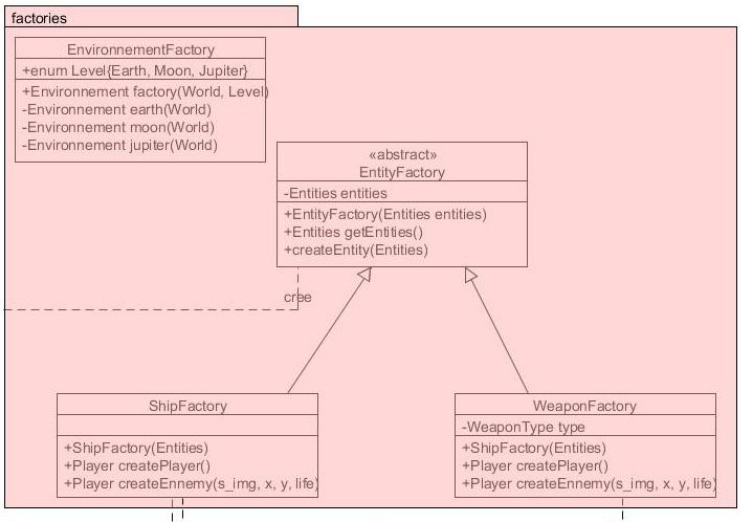
La classe abstraite Ship fournie les méthodes permettant de tirer, de bouger, et de détruire un vaisseau

- **Enemy** hérite de **Ship** et permet de lâcher des objets du jeu (munitions, etc…) et de détecter les collisions avec un **Weapon** du joueur ou le joueur lui-même.
    - Un **Enemy** contient un **EnemyBehavior** qui gère son comportement (**Enemy** appelle simplement la méthode **compute** de son **Behavior.** Un **Behavior** contient une liste d'**Action** à effectuer)
    - Un **Enemy** est chargé par un **EnemyLoader** (appelé par l'**Environnement**).
    - Un **EnemyLoader** est chargé depuis un document **XML** grâce à la classe **LoaderXML.**

- **Player** hérite de **Ship** auquel il ajoute les images et les animations du looping, de l'invincibilité et des dégâts (le vaisseau devient rouge). Détecte aussi les collisions avec les ennemis

- **Weapon** est une classe abstraite qui permet de gérer une arme : tir, collision, dommages engendrés, … Les armes par défaut héritent de cette classe.

Les factories

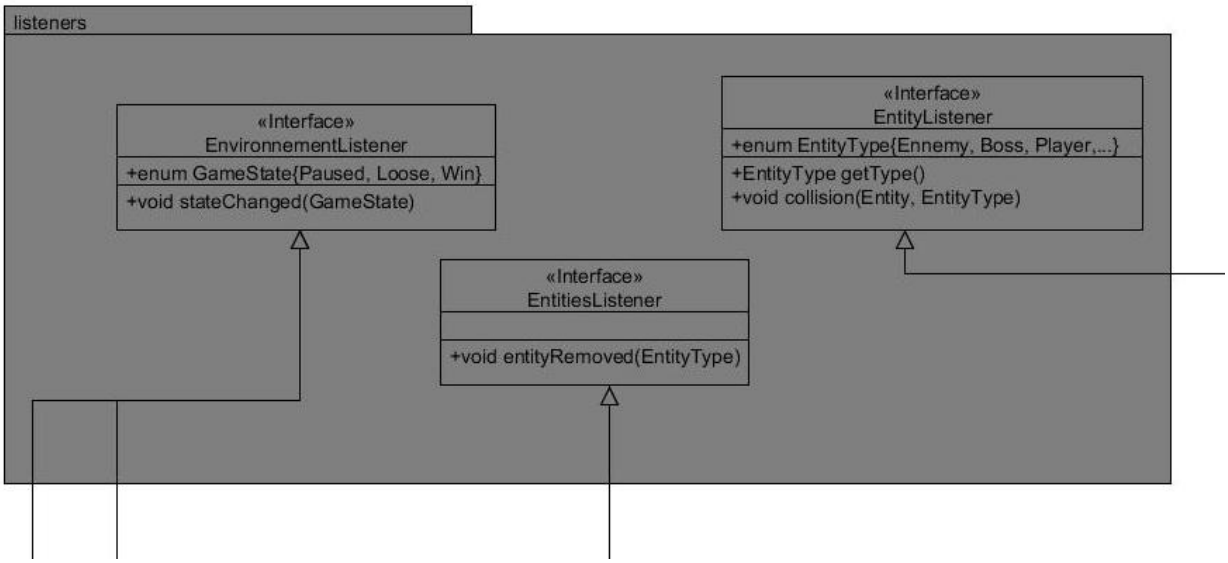


Les factories permettent de créer les objets plus facilement.

- **EnvironnementFactory** crée un **Environnement** correspondant aux différents niveaux du jeu : **Earth**, **Moon**, **Jupiter**
  Elle crée les différents objets qui vont permettre d'initialiser un **Environnement :** Un **Player,** une **Map,** un **EnnemyLoader** qui va charger les ennemies au fur et à mesure de l'avancement du niveau ainsi que la classe **Entities** qui va gérer toutes les **Entity** sur le terrain.

- **EntityFactory :** classe abstraite dont doivent hériter les factory permettant la création de sous classe d'**Entity :**
  - **ShipFactory**
  - **WeaponFactory**

Les Listeners



Les listeners permettent d'écouter les différents évènements du jeu.

- **EnvironnementListener :** permet de détecter un changement dans l'environnement : mise en pause, jeu gagné ou jeu perdu. (Utilisé par la classe **Game** pour charger les niveaux suivant ainsi que les différentes cinématiques)

- **EntitiesListener :** détecte simplement quand une **Entity** va être détruite (utilisé dans l'**Environnement** afin de détecter si le **Boss** ou le **Joueur** ont été détruit ce qui appellera donc les **EnvironnementsListeners**).

- **EntityListener :** détecte les collisions entre **Entity.** Utilisé dans toutes les sous classes d'**Entity,** permet d'enlever des vies à une entité et de la détruire si elle n'a plus de vie (en affichant l'animation de l'explosion). La destruction d'une **Entity** doit appeler les **EntityListeners**

V°)Conception

Les gestures

Les gestures permettent d'identifier un mouvement tracé à l'écran par l'utilisateur. On en distingue deux types :

- Si la gesture est tracé depuis le vaisseau, le missille se charge et part dans la direction ou la trace a été finie.
- Si la gesture est tracé ailleurs, on autorise les déplacements suivants :
  - Le drift : Décomposé en quatre parties, celui-ci reconnait les mouvements classiques d'un drift, de la gauche vers en haut et en bas, ainsi que de la droite vers en haut et en bas.
  -Le backoff : On reconnaît tout trait partant vers l'arrière
  -Les "flèches du clavier" : On reconnaît le mouvement vers la gauche, la droite et vers le haut.

Algorithme de reconnaissance

Pour détecter correctement les gestures, nous avons mis en place un algorithme pour détecter si un ensemble de point forme une droite, ainsi qu'un algorithme de reconnaissance de cercles.

Algorithme pour reconnaître les droites :

On calcule la différence d'angles entre chaque point, et on vérifie que la variation maximum de l'angles maximum et minimum ne dépasse pas une limite précise.

Algorithme pour reconnaître un cercle :

Dans un premier temps, on calcul le point le plus éloigné du premier point de notre liste, pour obtenir le diamètre du cercle, ainsi que son rayon supposé. On calcule ensuite la distance entre chaque point et le centre du rayon, et on vérifie que celle-ci ne subit pas trop de variations.

Jbox2d

Il a fallu comprendre le fonctionnement de **JBox2D** qui est un portage de la libraire **Box2D** écrite en **C++** (la documentation sur cette librairie en **Java** étant peu nombreux il a fallu utiliser la documentation de la version **C++** qui présente quelques différences).

Il a fallu comprendre comment **Box2D** utilisait sont **World** pour créer et gérer les **Body**.

On a ensuite abstrait son fonctionnement en créant une classe **Entity** contenant un **Body** ainsi qu'une classe **Entities** contenant un **World.** Ces deux classes fonctionnent de façon assez similaire (nos **Entities** gèrent les **Entity** de la même façon que le **World** de **JBox** gère ses **Body**)

La classe **Entities** sera ensuite utilisé dans les classes parentes afin de gérer les collisions, l'ajout et la destruction des **Entity.**

La classe **Entity** servira de modèle à des sous classe composant notre environnement.

Problèmes rencontrées

Structuration du projet

Comme l'architecture du projet changé au fur et à mesure de l'avancement, en fonction des connaissances apprises, il a fallut a plusieurs reprises réalisées de la refactorisation de notre code, ce qui nous a fait perdre du temps, sur un timing déjà sérré. En effet, les comportements de nos objets évoluant, nous nous mettions en permanence en question sur le rôle de chacun de nos objets.

Apprentissage de Jbox

Apprendre une librairie, tout en développant un projet aussi grand qu'un jeu, nous a demandé beaucoup de temps et d'effort. Dans un premier temps, nous avons lus de la documentation sur le sujet, avant prendre en main la librairie.

Les horaires…

Réussir à allier les cours (partiels, Tps à rendre, heures de cours) avec le développement de ce projet. Les heures de sommeil étant réduites au minimum durant cette période. Il a parfois été difficile de tout concilier (TP noté sur les WildCards loupés par exemple)

VI°)Possibilités d'améliorations

Les différentes couches du programme ont été développées dans un souci de maintenabilité, il est donc assez facile d'ajouter une fonctionnalité à une couche sans devoir réécrire l'implémentation de couches supérieures.

Ce projet nous aura demandé beaucoup de temps et d'investissement, mais nous sommes satisfaits du résultat, même si les possibilités d'améliorations ont été limitées par le temps. Nous avons cherchés à produire du code maintenable, utilisant les divers aspects vus en cour, tels que les classloader, le dessign pattern factory, ...