

# Documentation

## Description du projet

Dynamo est un jeu de plateforme liant agilité et vitesse pour parvenir à bout d'un unique niveau, long et rempli de pièges.

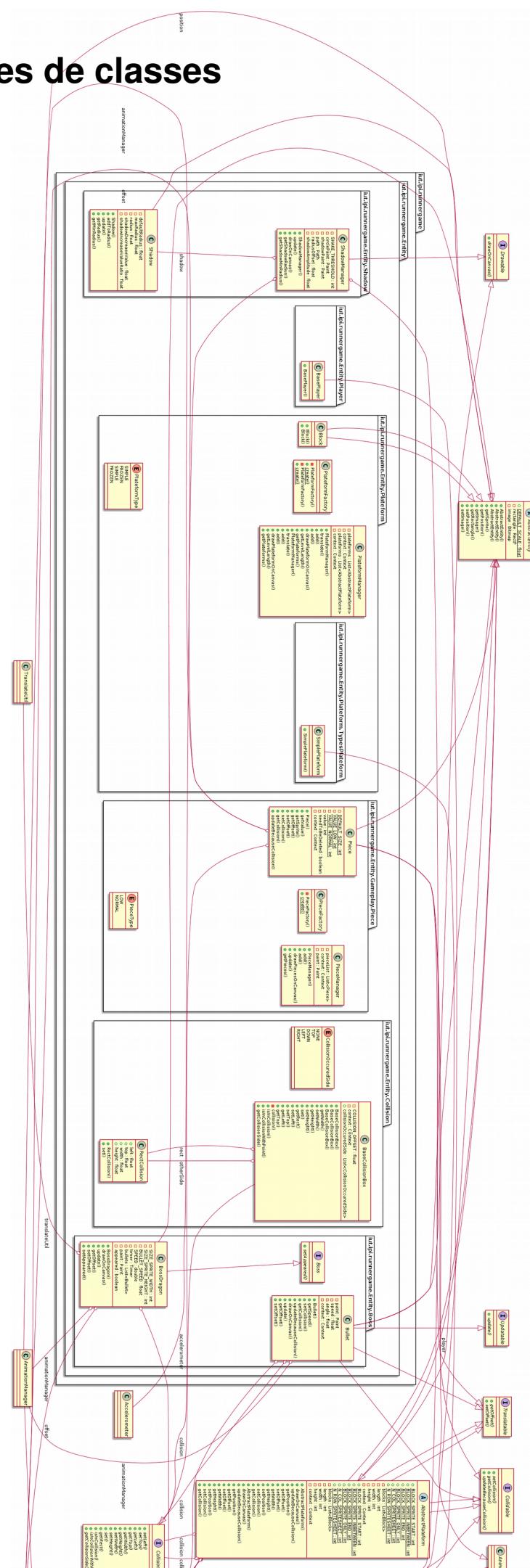
Le jeu est 2d-scroller, ce qui signifie que le héros peut se déplacer à droite, gauche et sauter. Le but est donc de faire le plus gros score pour concurrencer ses amis. Pour faire le plus haut score, il faut récupérer un maximum de pièces et aller le plus vite possible sans se faire tuer par le boss ou tomber des plateformes...

Via son niveau unique, le jeu comprend un ensemble complètement animé, avec un style graphique homogène et un harmonisation des couleurs. Une surprise, un boss, arrive au bout de 30 secondes de jeu. Plusieurs types de pièces sont collétables pour augmenter son score, avec des petites pièces qui rapportent évidemment moins que les grandes. A la mort du personnage et après l'abandon du joueur, un écran de fin de jeu s'affiche récapitulant le meilleur score.

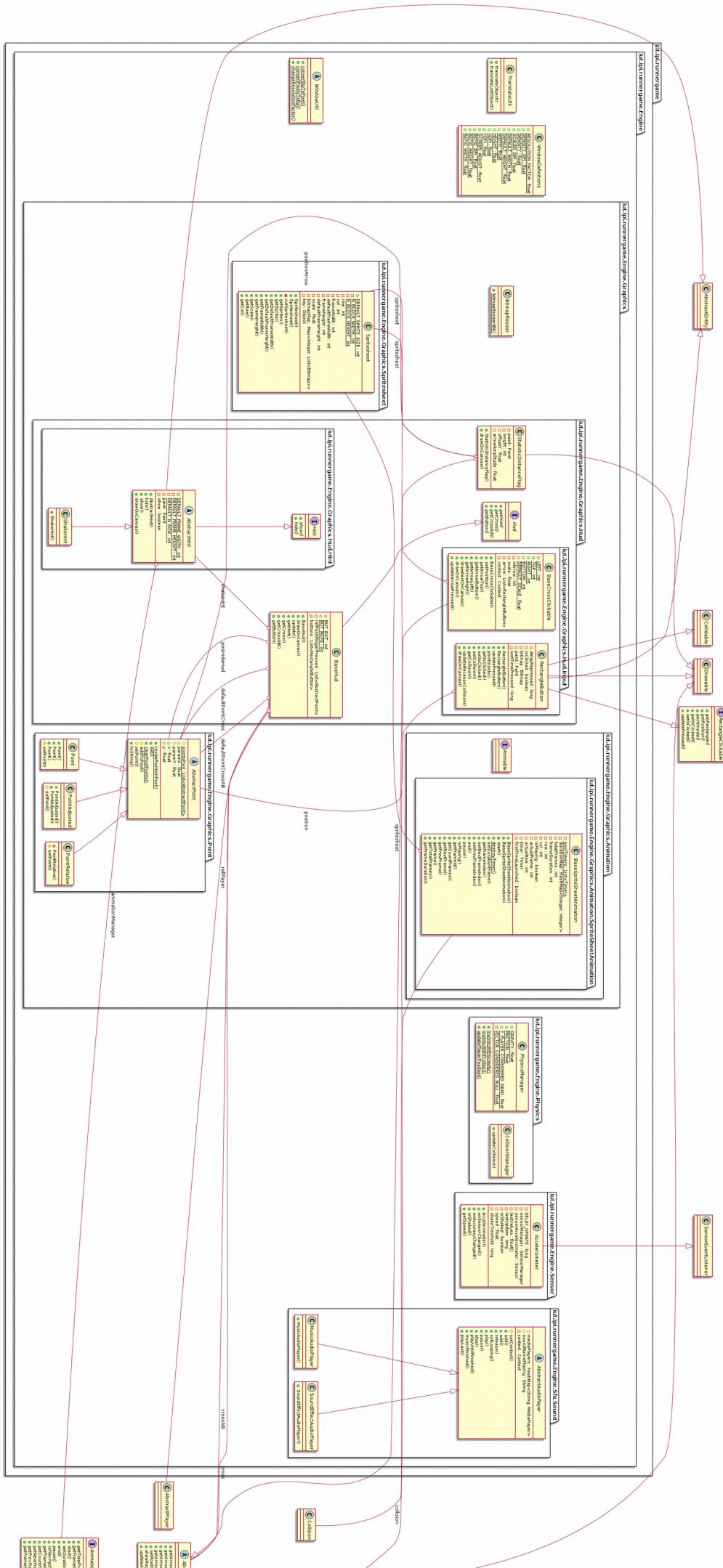
Le jeu fonctionne sur tout les types de téléphones (malheureusement les tablettes ont du mal...)

De plus, un éditeur de niveau est fourni en Python pour permettre de créer son propre niveau et de l'importer dans le jeu, en recompilant le code.

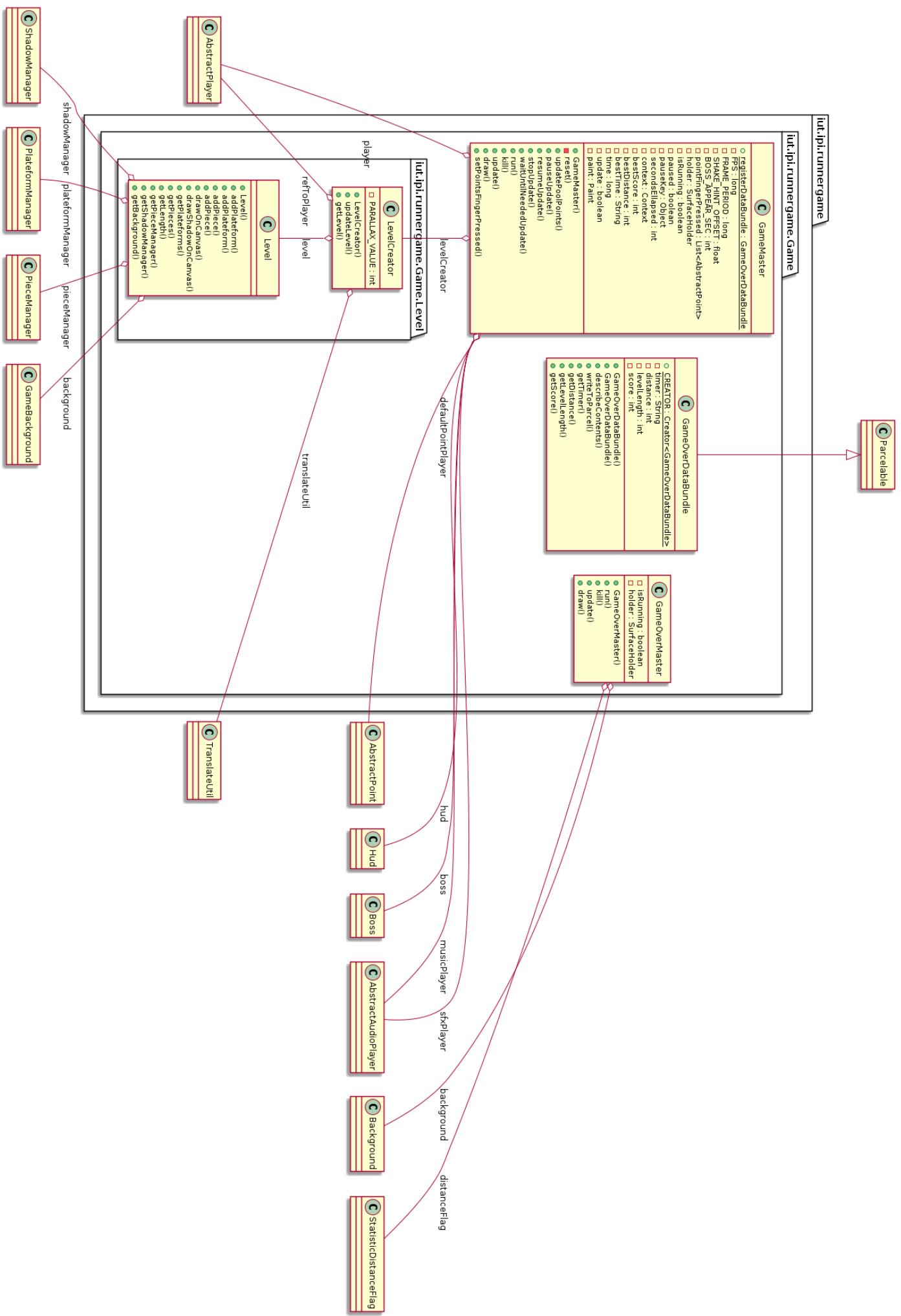
# Diagrammes de classes



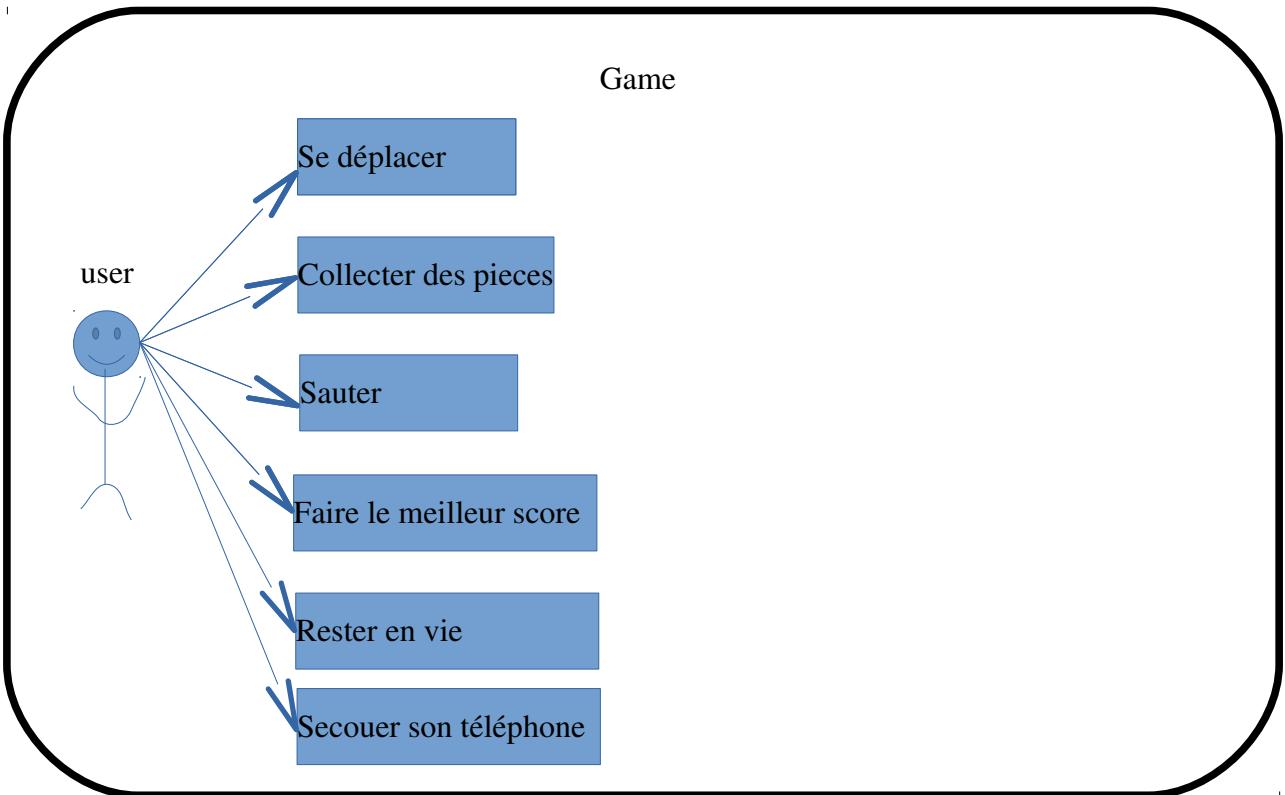
ENGINE's Class Diagram



## GAME's Class Diagram



# Diagramme cas d'utilisation



# Explication Diagramme UML

Le projet s'architecture sur 4 grosses parties : Game, Engine, Entity et Activity.

Le choix a été de faire que Engine soit complètement séparé, et donc complètement réutilisable sur d'autres projets à venir si jamais il y'en a besoin. Le but étant de faire un moteur complètement générique, et donc paramétrable.

Le moteur déclare entre autre quelques classes permettant la simplification de l'utilisation des canvas, comme par exemple gérer automatiquement le positionnement des entités avec un pool de points qui vont se mettre à jour automatiquement en fonction de la taille de l'écran. Tout est fait pour que par la suite il soit facile d'implémenter un nouveau concept au jeu.

Il comprend donc une couche « graphics », qui s'occupe du rendu, de charger les sprites, de les découper, gérer les animations etc.

Une autre couche « sfx » permettant de s'occuper des chargements des sons, de les jouer, de les mettre en pauses et donc de créer une ambiance sonore.

Puis une couche « physics » permettant de gérer les différentes interactions avec les entités, comme par exemple le fait que le joueur ne passe pas à travers les plateformes, ou qu'une pièce disparaîsse quand elle rentre en collision avec le joueur.

Entity est une surcouche au moteur, qui du coup reste propre au jeu. C'est dans ce package qu'est codé tous les éléments affichables à l'écran, tel que le joueur, le boss, les plateformes, les pièces, le fond d'écran qui bouge etc.... Tout est créé pour que l'ajout d'un composant qui puisse bouger, être animé ou mis à jour soit simple.

AbstractEntity est la base de tout ce qui est afficher à l'écran. Cela permet un control total et donc de très facilement ajouter une nouvelle entité.

Une factory a été faite pour gérer les plateformes, pour que si dans la mise à jour futur, une nouvelle type de plateforme doit être ajoutée, une simple modification de la factory permettra de l'intégrer.

Une factory pour les pièces fonctionnant comme celle des plateformes existe aussi.

Puis finalement, Game, qui s'occupe de toute la logique du jeu. Tout ça repose sur un Master, ici GameMaster, qui va s'occuper de réunir et de gérer les différentes parties ensemble.