Infinite Runner – Documentation

Notre application est un jeu dont l'intérêt est de divertir l'utilisateur. Dans ce jeu, vous incarnez un alien qui traverse plusieurs décors, et devez le faire survivre le plus longtemps possible, dans des niveaux à difficulté choisie par l'utilisateur. Le but est de créer une expérience qui s'adapte selon les goûts en terme de difficulté des uns et des autres.

Diagramme de cas d'utilisation

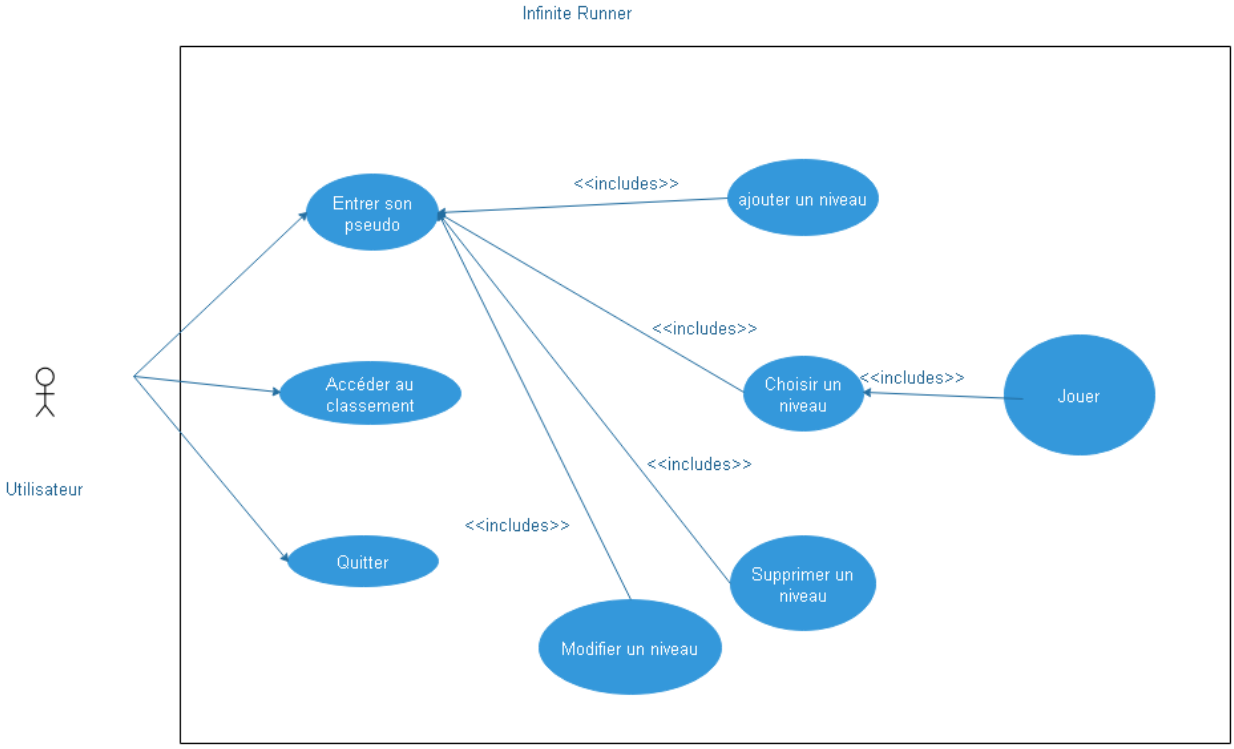
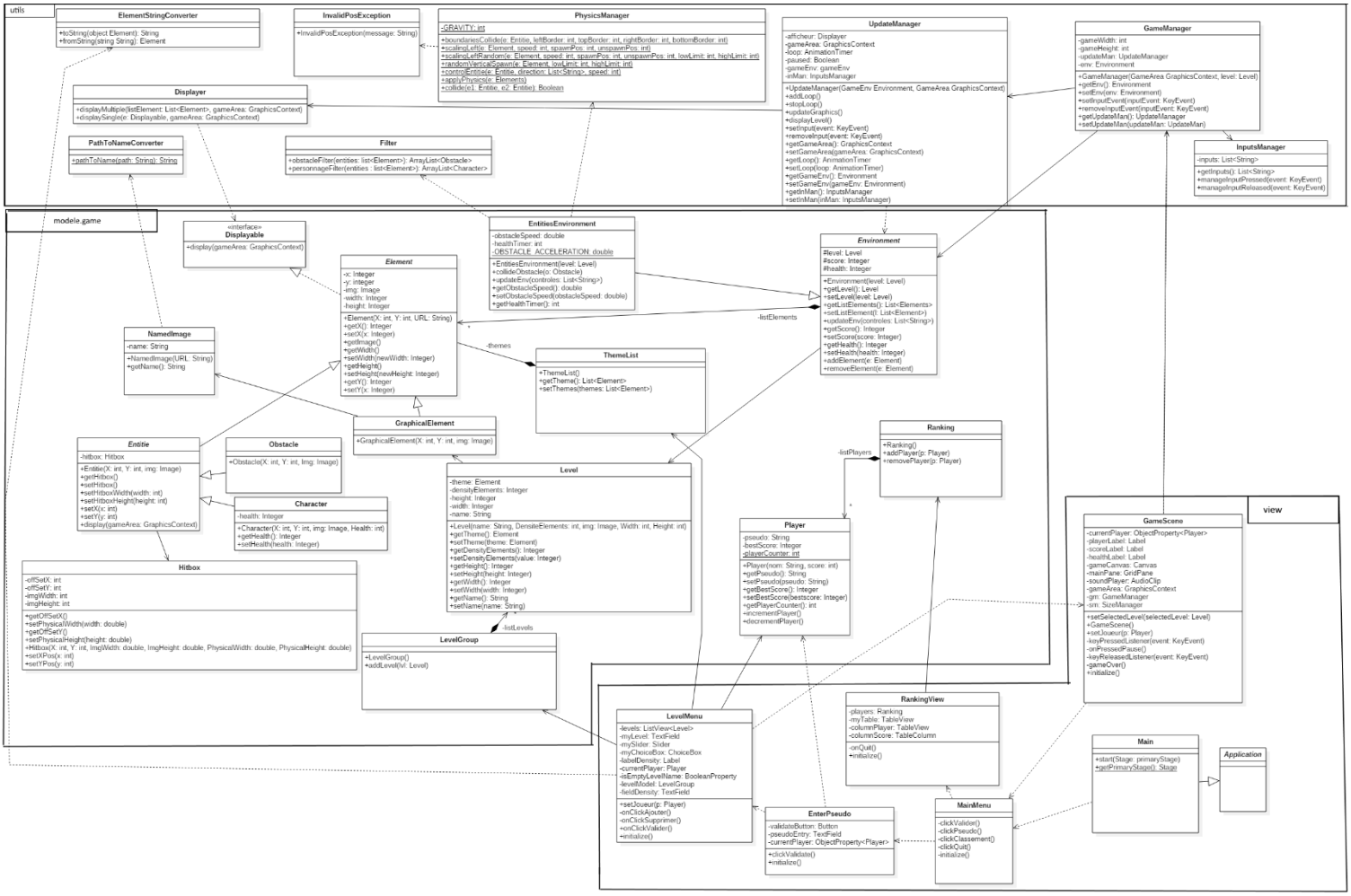


Diagramme UML



Description

Ce diagramme possède un point d'entrée, la classe Main, qui appelle le contrôleur de la vue MainMenu. C'est cette vue qui va permettre à l'utilisateur de faire son choix entre l'accès au classement, le lancement du jeu ou tout simplement de quitter l'application.

Si l'utilisateur choisit d'afficher le classement, une vue RankingView s'affiche. Son contrôleur fait appel à la classe Ranking pour afficher une liste de la classe Player, nom et meilleur score.

Sinon, l'utilisateur doit entrer son pseudo via la vue EnterPseudo, qui appelle ensuite la vue LevelMenu. Pour afficher la liste des niveaux, celle-ci va appeler LevelGroup, classe contenant une liste de Level. Chaque level possède un attribut de la classe Element, car l'image de fond de chacun est un GraphicalElement, ceci afin de permettre à l'utilisateur de choisir lui-même le fond du niveau parmi une liste prédéfinie.

Une fois le niveau sélectionné, le contrôleur de la vue GameScene est appelé. Celui-ci possède un GameManager, qui possède une instance de la classe EntitiesEnvironment, et un UpdateManager créé à partir de cette instance.

Le but de l'updateManager est de gérer la mise à jour de l'environnement en lui passant les inputs que InputsManager a détecté lors des entrées clavier. Il utilise un Displayer qui va servir à afficher les différents objets de l'environnement. Pour cela, ce dernier va faire appel à une interface Displayable implémentée par la classe Element. Ainsi, chaque Displayable sera affiché (display) a la demande de l'UpdateManager. Il met ainsi à jour l'affichage du jeu.

La classe Environment gère les positions des différents éléments (obstacles, personnages...) via la classe Element. Elle en possède une liste. La classe EntitiesEnvironment, qui hérite d'Environment, a pour responsabilité de gérer les collisions et les positions de l'environnement dans lequel se trouve un personnage et des obstacles. Elle fait cela au moyen d'une classe PhysicsManager, qui fait apparaitre aléatoirement les obstacles, positionne le personnage suivant l'entrée clavier, détecte une collision entre deux éléments. Elle va par exemple décrémenter la vie du personnage si le PhysicsManager l'avertit qu'il est touché.

Patrons de conception

Nous avons implémenté le patron de conception Médiateur dans notre application avec comme médiateur la classe Environment et comme collègues la classe Element. Nous avions besoins de connaître la position de chacun de nos éléments dans jeu pour pouvoir détecter des collisions entre ces éléments. Ainsi en récupérant la position de chacun des éléments dans le jeu, le médiateur les régule et évite une dépendance entre tous les éléments présents dans l'environnement. Il permet de plus d'affecter un comportement à l'ensemble des éléments qu'il possède.

Nous avons également implémenté le patron de conception Observateur afin de surveiller la vie du personnage pendant le jeu depuis la vue GameScene. Il est ainsi possible d'afficher la vie du personnage sans attente active et de changer de vue lorsque les points de vie atteignent zéro.