VALLEIX Benjamin

BONAZ Clément

Documentation du Projet Jumper

Contexte :

Notre application Jumper est un jeu 2d développé en JavaFx, l'utilisation de notre jeu est assez simple. Après avoir lancé l'app il suffit de rentrer un pseudo et de cliquer sur jouer, le jeu consiste à obtenir le score le plus élevé en sautant au-dessus des obstacles en appuyant sur la barre espace du clavier. Si le personnage touche un obstacle alors la partie se termine, le score de la partie est comparé au meilleur score du jeu afin d'être enregistré ou non et un bouton pour relancer la partie apparaît à l'écran.

Diagramme de Paquetage :

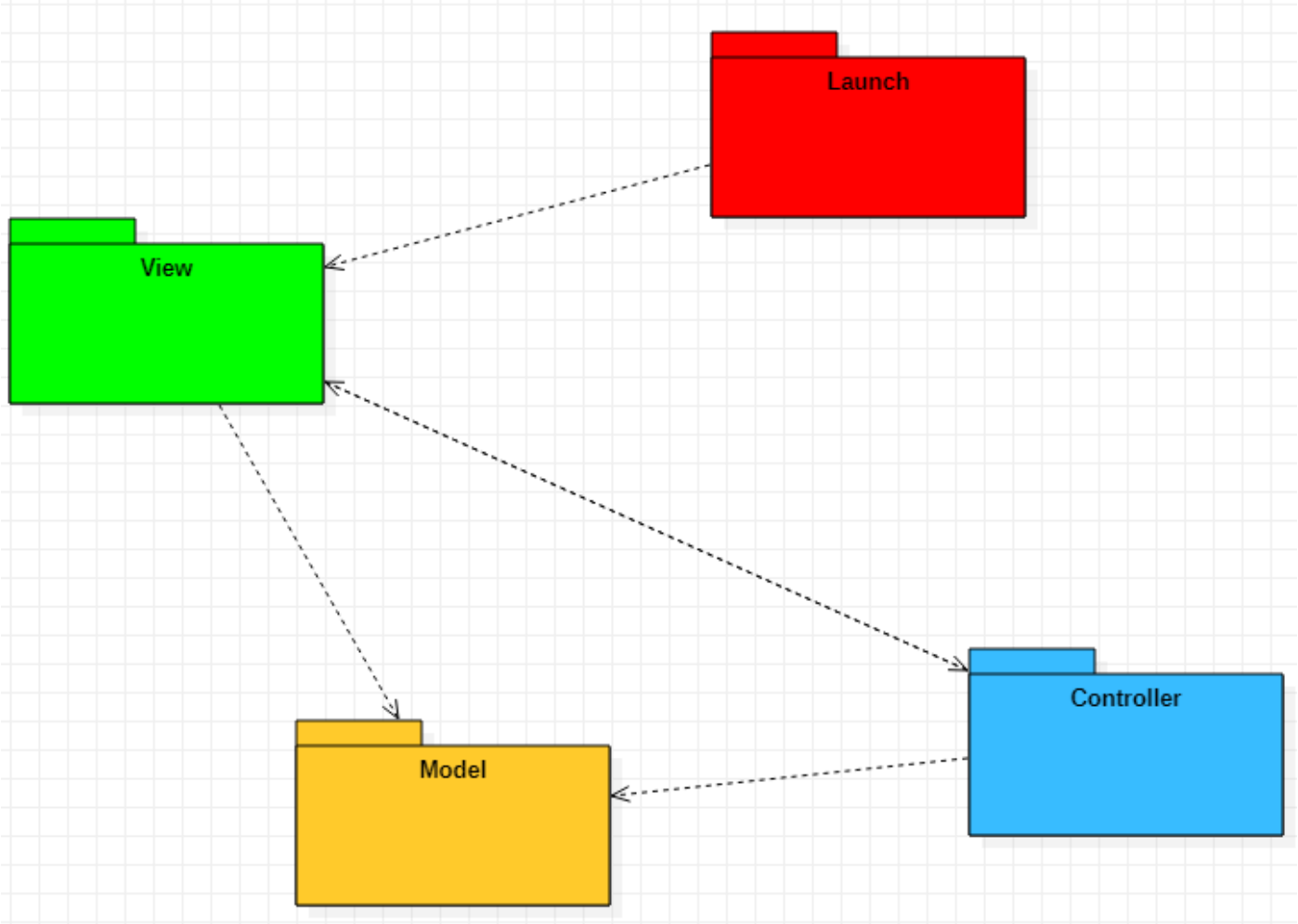
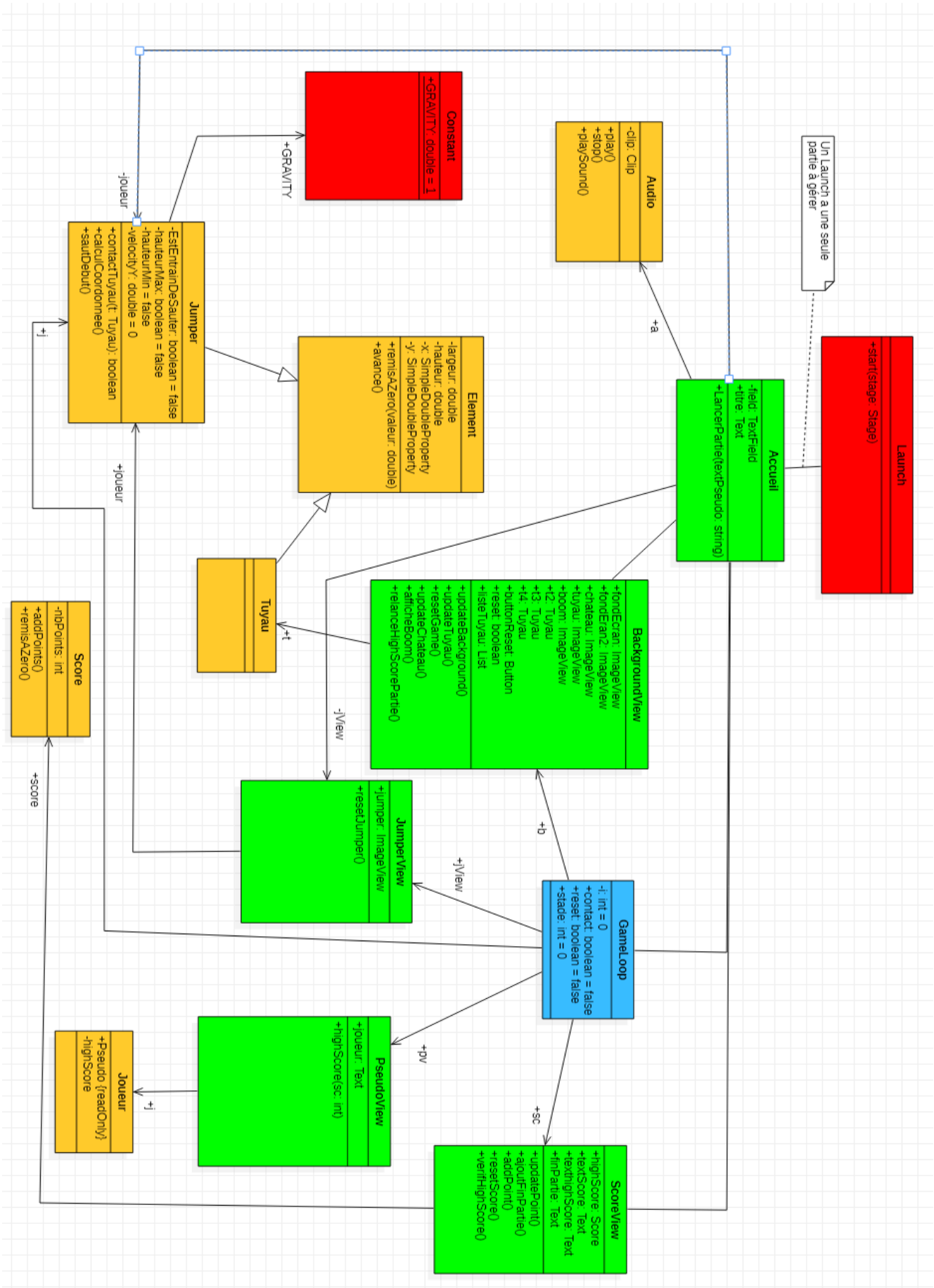
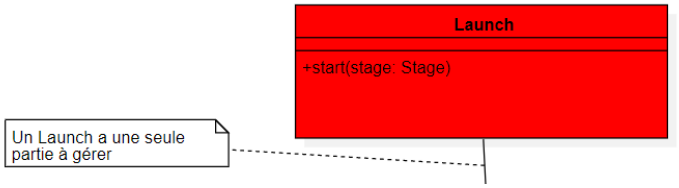


Diagramme de Classe :

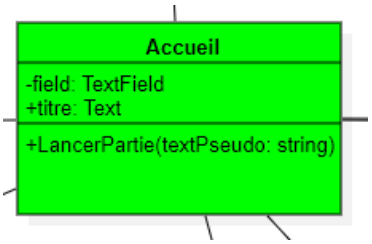


Launch :



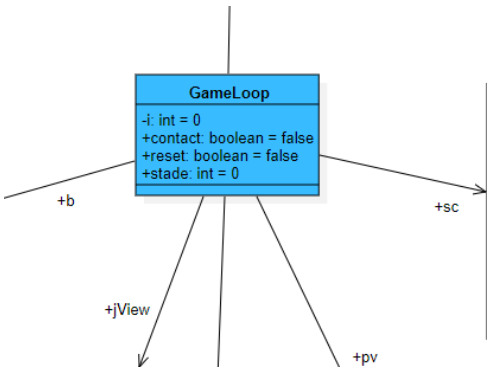
La classe Launch appartient au package Launch, étend la classe application de javafx et implémente la méthode start, cette méthode créée un objet Stage qui est la fenêtre principale de notre application, dans cette fenêtre nous allons charger un objet scene qui va afficher les objets graphiques que nous voulons.

Accueil :



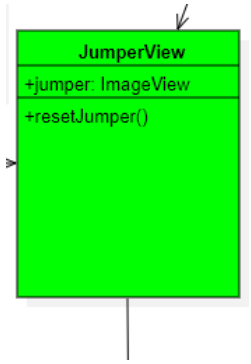
La classe Accueil appartient au package View, cette classe possède deux attribut FXML field et titre, mais aussi un attribut joueur de type Jumper, jView de type JumperView, a de type Audio et elle contient la méthode LancerPartie. LancerPartie ferme la fenêtre et charge tous les objets graphiques de notre app dans un objet scene avant de créer un objet GameLoop. On charge le score maximum de l'app depuis le fichier de persistance puis une fois que tous les objets graphiques sont stockés dans scene on la donne à Stage puis on affiche de nouveau la fenêtre de l'app. Enfin, on déclare un event qui sera réalisé chaque fois que la touche barre espace du clavier et pressé.

GameLoop :



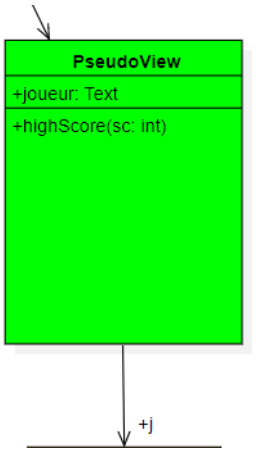
GameLoop fait partie du package Controller et est une classe qui possède plusieurs attribut, i qui est de type int et static, contact et reset de type boolean et stade de type int. Cette classe implémente un thread qui correspond à la boucle de jeu tant que l'app n'est pas fermé. A chaque tour de boucle les méthodes d'update de backgroundView, ScoreView et j afin mettre à jour les éléments graphiques, après avoir mis a jour les éléments graphique on test si j est en contact avec un obstacle. S'il y a contact la partie est fini et on affiche à l'écran le score final du joueur et un bouton pour lancer une nouvelle partie.

JumperView :



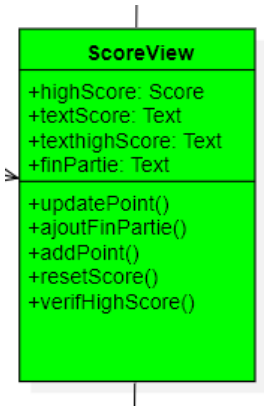
JumperView fait partie du package View, étend la classe Parent et possède 2 attributs qui sont jumper de type ImageView et joueur de type Jumper. Dans cette classe, on crée un objet graphique qui représente le joueur et on bind les attributs X et Y de jumper sur les atribut X et Y de joueur afin que si les attributs de joueur changent alors ceux de jumper aussi. JumperView possède une méthode appelée resetJumper qui est une méthode qui réinitialise les propriétés de joueur (et de jumper grâce au data binding) a leurs valeurs initiales.

PseudoView :



PseudoView fait partie du package View, étend la classe Parent et possède 2 attributs qui sont joueur de type text et j de type Joueur. Cette classe nous permet d'afficher le pseudo du joueur et possède une méthode highScore qui permet de définir le highScore du Joueur j.

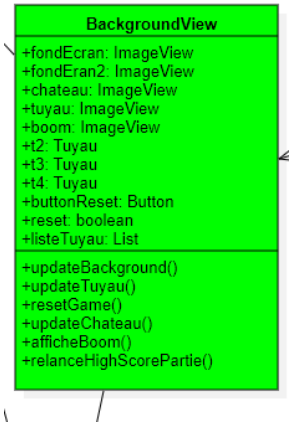
ScoreView :



ScoreView fait partie du package View, étend la classe Parent et possède 5 attributs qui sont highScore et score de type Score, textScore, texthighScore et finPartie qui sont de type Text. Cette classe contient la méthode

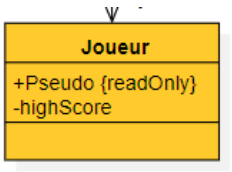
updatePoint qui mets à jour textScore en récupérant le contenu de score, ajoutFinPartie qui fait apparaître à l'écran le message de fin de partie, addPoint qui appelle la méthode addPoint de score, resetScore qui reset le score et fait disparaître le message de fin de partie et la méthode verifHighScore. verifHighScore compare le score final du joueur et le highScore, si le score et supérieur au highScore alors il est enregistré.

BackgroundView :



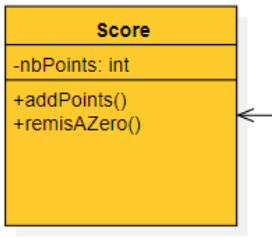
BackgroundView fait partie du package View, étend la classe Parent et possède 12 attributs qui sont fondEcran, fondEcran2, chateau, tuyau, boom qui sont de type ImageView, t, t2, t3, t4 de type Tuyau, listeTuyau de type List et buttonReset de type Button. Dans son constructeur, on crée tous les objets graphiques liés au background de l'app, il y a notamment l'objet tuyau pour lequel on bind ses propriétés X et Y sur celle d'un des tuyaux de type Tuyau choisi au hasard. La méthode updateBackground met à jour la position des attributs fondEcran et fondEcran1 en modifiant leurs coordonnées X de 5. updateChateau met lui aussi la position du château à jour, updateTuyau met à jour la position du tuyau en appellant la méthode avance de t, afficheboom fait apparaître l'objet graphique boom. resetGame reset les coordonnées X et Y de tous les objets graphiques géré par BackgroundView. relanceHighScore récupère le highScore contenu dans le fichier de Persistance HighScore.xml pour ensuite retourner un score.

Joueur :



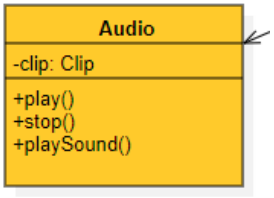
Joueur fait partie du package Model et possède 2 attributs qui sont pseudo de type String et highScore de type int. Le nom donné en paramètre au constructeur de joueur permet de définir le Pseudo.

Score :



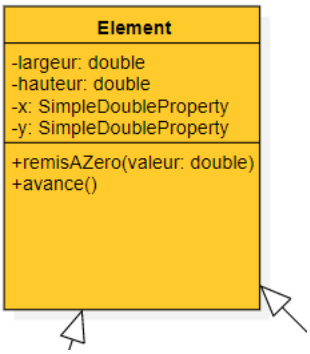
Score fait partie du package Model et possède un seul attribut nbPoints de type int, la méthode addPoints incrémente le score de 1 à chaque fois qu'elle est appelée. Enfin la méthode remisAZero réinitialise nbPoints à 0.

Audio :



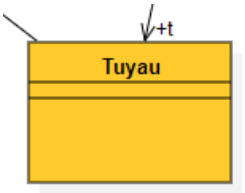
Audio fait partie du package Model et possède un seul attribut clip de type Clip. Dans le constructeur, on initialise clip en chargent le fichier. waw choisis. La méthode play lance le clip et la méthode stop arrête le clip, enfin la méthode playSound qui prend en paramètre une chaîne de caractère créé un audio s et lance la méthode play de s.

Element :



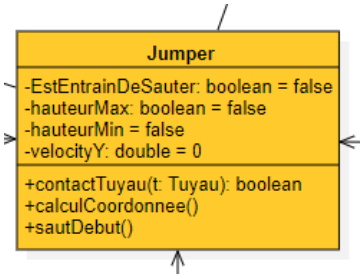
Element fait partie du package Model et possède 4 attributs qui sont largeur, hauteur de type double et x, y qui sont de type SimpleDoubleProperty. La méthode remisAZero set x à la valeur donnée en paramètre de la méthode, enfin la méthode avance soustrait 5 à la valeur de x à chaque appel de la méthode.

Tuyau :



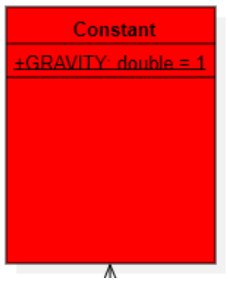
Tuyau fait partie du package Model et hérite de Element.

Jumper :



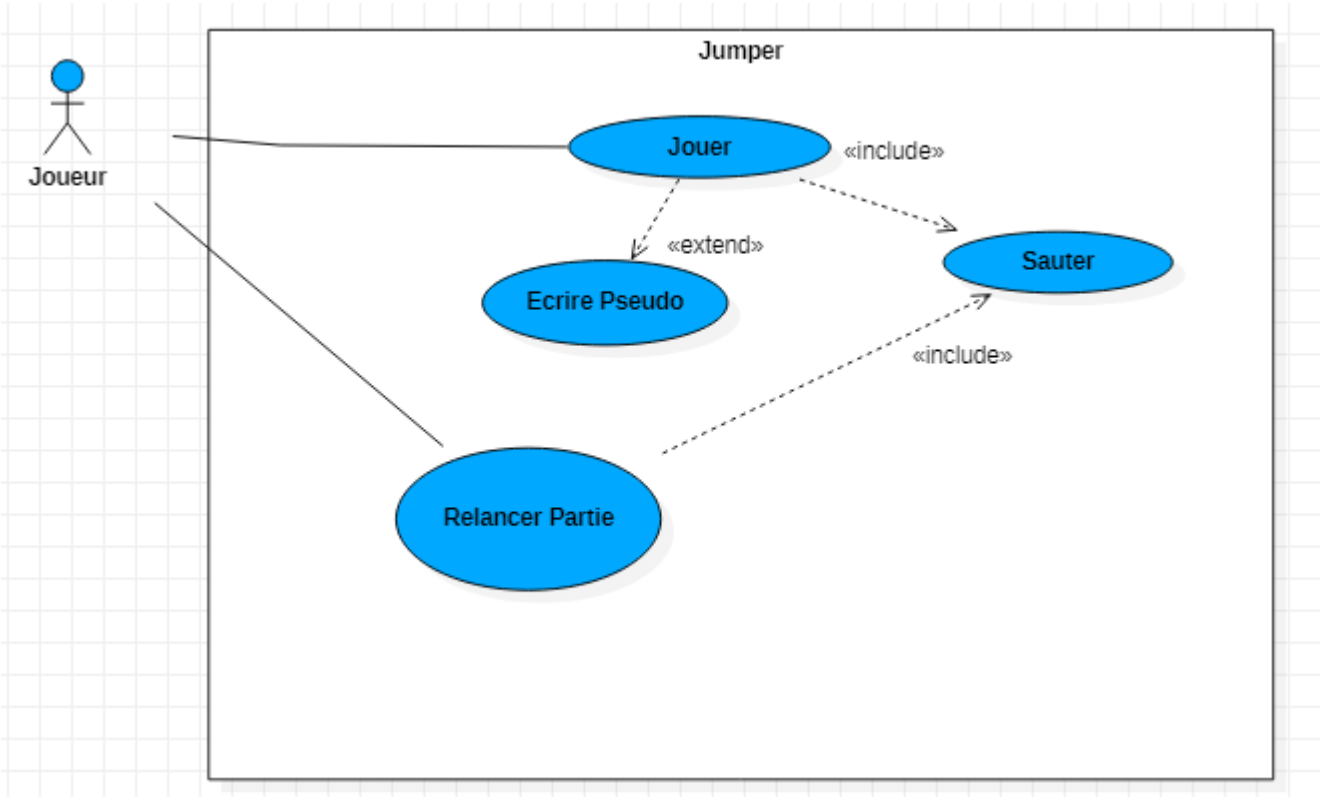
Jumper fait partie du package Model, hérite de Element et possède 4 attributs qui sont velocity de type double, EstEntrainDeSauter, hauteurMax et hauterMin de type boolean. La méthode contactTuyau vérifie si le tuyau t donné en paramètre est en contact avec jumper et retourne true s'il y a collision sinon false. La méthode calculCoordonnee calcule la coordonnée Y de jumper en fonction de velocity et de GRAVITY (constante de type double issu de la classe Constant), la méthode sautDebut initialise velocity a -6.

Constant :



Constant fait partie du package Launch et possède un attribut GRAVITY de type double initialisé a 1.

Diagramme de Cas d'utilisation :



L'acteur principal de notre application est le Joueur. En premier lieu, un Joueur doit entrer son pseudo avant de pouvoir jouer. Une fois le pseudo renseigné, la Partie se lance et cela inclut le fait qu'un Joueur peut faire sauter Jumper en appuyant sur la touche espace. La partie est une boucle de jeu, tant que jumper n'a pas touché un tuyau, la partie continue en boucle tout en accélérant au fur et à mesure que les points augmentent. Une fois la partie finis, un Joueur peut relancer une partie en appuyant sur le bouton relancePartie. Un Joueur peut relancer autant de partie qu'il le souhaite pour battre son meilleur score.