

Projet "Flop Flop"

24.01.2022

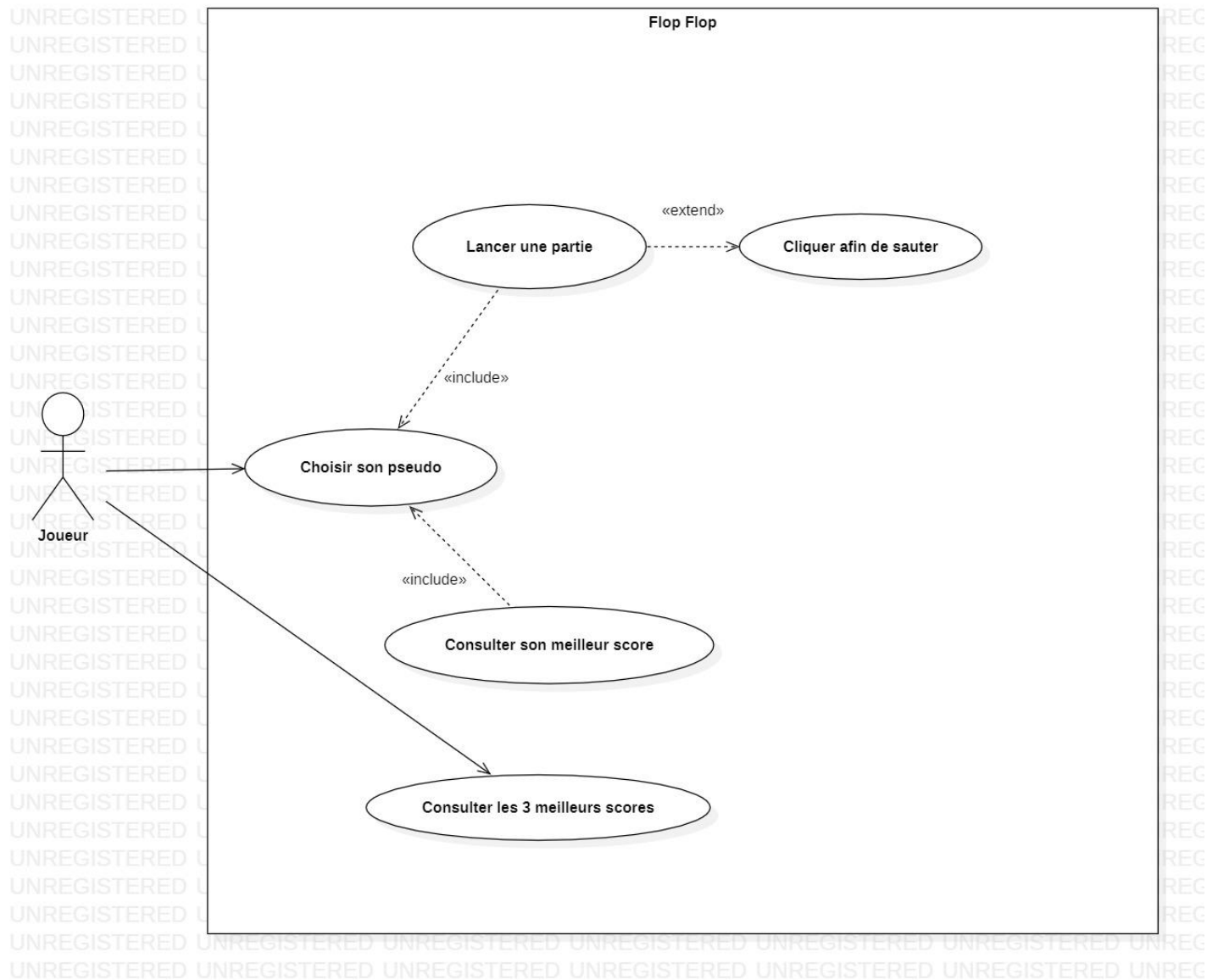
MARTEL Baptiste - SIRVEN Samuel

Projet de Conception et Programmation objets avancées

Langages : Java, FXML

Durée : 7 semaines

Diagramme de cas d'utilisation



Scénario de jeu

1. Le joueur doit rentrer un pseudo
2. Il clique ensuite sur le bouton "START"
3. Il appuie sur la touche "ESPACE" pour faire déplacer l'oiseau et passer entre les tuyaux

4. Dès qu'il heurte un tuyaux ou que l'oiseau sort de la fenêtre de jeu, celui-ci s'arrête
5. Il peut ensuite cliquer sur "HOME" ou "RESTART" pour revenir à la fenêtre principal ou rejouer

Sans rentrer de pseudo, le joueur peut consulter les 3 meilleurs scores en cliquant sur le bouton "SCOREBOARD" sur le menu principal.

Architecture du projet :

Pour commencer, notre application se divise en trois package distincts à qui s'ajoutent un package de tests.

Package "view"

Contient le code-behind des 4 vues de notre projet : "MainWindow" sur laquelle le joueur peut lancer une partie ou consulter les 3 meilleurs scores réalisés, "Game" qui représente notre vue de jeu, "ScoreBoard" et "Erreur" qui s'affiche lorsque l'utilisateur ne rentre pas de pseudo et clique sur "START".

Package "model"

Ce package contient toutes les classes de notre modèle contenues dans leurs package respectifs contenant généralement elles-mêmes et leur point d'extensibilité qui sont à base de notre jeu. Parmi ceux-ci, il y a le package "manager" qui contient les gestionnaires de notre jeu. Il contient une classe Manager qui contient la plupart de nos fonctionnalités telles que la création d'un monde au lancement d'une partie par exemple. Il y a ensuite la classe FXController qui gère le binding des propriétés d'objet du modèle sur les propriétés de composants de la vue, ainsi que l'initialisation des vues, généralement avec l'ajout de boutons et d'ImageView en déléguant à une classe Renderer. De plus, il y a une classe Navigator qui gère le changement de scène de notre fenêtre principal, et une classe ScoreChecker qui va mettre à jour le score actuel du joueur dans une partie.

Package "persistance"

Ce package contient deux interfaces ILoad et ISave qui sont respectivement associées aux méthodes de chargement d'une liste de Joueur à partir d'un fichier et de sauvegarde d'une liste de Joueur. Nous avons une classe SaverBinaire qui implémente ISave et redéfinit la méthode dataSave en sauvegardant la liste de Joueur dans un fichier binaire. Deux classes implémentent l'interface ILoad et la méthode loadData : LoaderBinaire qui charge une liste de Joueur à partir d'un fichier

binaire et Stub qui instancie une liste de Joueur pour faire office de test pour le binding sur notre vue.

Description du diagramme de classes :

Tout d'abord, nous avons généralement dans chaque sous-package, une classe qui représente un point d'extensibilité.

Nous avons tout d'abord la classe Element qui représente le point d'extensibilité des éléments qui seront animés sur notre vue : l'oiseau (Bird) et les tuyaux (Obstacle). Le Background, quant à lui, reste fixe. Les Element possèdent donc une Position (x,y), une longueur, une largeur ainsi qu'une URL pour accéder à l'image qui les représente.

Cette animation se fera d'abord dans les Classes du sous-package Animation dédiés aux Element : celles-ci sont des listeners d'une boucle pour effectuer tous les N millisecondes une action.


La classe AnimationBird va appeler la classe de déplacement associé pour soit faire voler l'oiseau si l'utilisateur a appuyé sur espace, sinon faire chuter l'oiseau. La classe AnimationObstacle va permettre d'appeler le déplacement de l'obstacle et d'instancier tous les 600 signaux reçu de "créer un obstacle", en appelant une méthode de la classe CreatorRandom du sous-package "creator".

Ces classes de déplacement sont contenues dans le sous-package "displacer". Elles font appel aux classes du sous-package "collider" qui vont vérifier si l'oiseau est entré en collision avec un obstacle. Pour le savoir, le collider doit connaître le World actuel qui représente une liste d'Element.

Les classes Animation sont des listeners des classes du sous-package "boucle" qui sert à envoyer un signal, comme dit précédemment, toutes les N millisecondes. Nous avons donc implémenté un patron de conception Observateur.

Notre Manager va donc permettre de "créer des mondes" en utilisant la classe CreatorRandom et de gérer le début et la fin du jeu, en observant la propriété booléenne "gameOver" de la classe AnimationBird qui sera mise à "true" lorsqu'un déplacement n'est pas possible. Il pourra donc lancer les boucles, et stopper les animations lorsque celle-ci passe à "true".

Plus du côté vue, nous avons le sous-package "render" qui contient une interface fonctionnelle RendererSupplier, implémentée par la classe Renderer, qui ajoute et les Element à la vue et créer le binding entre eux et leurs composants. Celui-ci sera géré dans la classe FXController du sous-package "manager" qui gère tous nos éléments graphiques,



et les événements de la vue comme l'appui sur le bouton restart. Il appellera donc des méthodes du manager à la réception d'un événement.

Enfin les classes du sous-package "log" contiennent juste la liste des Player, et permettent de chercher, ajouter, ou mettre à jour un Player. Cette liste sera instanciée à l'appel de la méthode `dataLoad` du Manager.