

## Contexte

Pour le bien de la compréhension de ce contexte d'application, nous allons vous introduire au terme « launcher ». Un launcher est un logiciel accueillant des jeux vidéo. Il permet d'acheter des jeux, les télécharger, les mettre à jour, les désinstaller, etc... Un launcher sert donc à la gestion des jeux vidéo acheté ou obtenu par l'utilisateur. Parmi les plus connus, on peut citer : Steam, Epic Games, Uplay, Origin ou encore Riot.

Aujourd'hui, les gens possèdent de plus en plus de jeux vidéo. Et cette augmentation passe par la multiplication du nombre de launcher qui proposent chacun leurs exclusivités. Ce qui oblige l'utilisateur à ouvrir plusieurs launcher au cours de la même journée. D'où l'idée de notre logiciel, réunir en un seul endroit tous les jeux de l'utilisateur.

Cette application permettra d'obtenir diverses informations sur les jeux de l'utilisateur. On pourra par exemple voir une image ou une vidéo de présentation du jeu ainsi qu'un court résumé du contenu du jeu.

L'application vise en priorité un public amateur de jeu vidéo.

Pour sélectionner un jeu, l'utilisateur cliquera sur le jeu de son choix dans la partie gauche de l'application. Les détails (partie droite de l'application) affichés seront : une image ou vidéos montrant le jeu (image de jaquette ou vidéo commerciale) et une description du jeu. L'utilisateur pourra même écrire une note lui permettant de se rappeler de certaines choses. Par exemple, se rappeler quelles sont les missions qu'il veut effectuer, mais n'a pas eues le temps de faire lors d'une précédente session de jeu.

L'interface aura un thème sombre et pas de thème lumineux, car les thèmes sombres sont plus agréables à regarder de nuit ou dans la pénombre.

L'application proposera des fonctionnalités intéressantes pour les joueurs telles que :

- La recherche d'un jeu via la barre de recherche permettra de faciliter une recherche spécifique dans le cas d'un utilisateur possédant beaucoup de jeux.

- Une recherche automatique des jeux vidéo présents sur l'ordinateur de l'utilisateur. Le menu paramètre pourra alors servir à renseigner des chemins de recherche supplémentaires.

- Le remplissage automatique des détails d'un jeu vidéo (partie droite de l'application), afin que l'utilisateur n'ait pas à renseigner manuellement tous les détails pour chaque jeu qu'il possède. En effet, si l'utilisateur possède beaucoup de jeu le remplissage des détails jeu par jeu peut prendre un temps considérable.

- L'ajout d'un bouton « lancer le jeu » dans la partie détails d'un jeu afin que l'utilisateur puisse lancer le jeu directement à partir de l'application.

## Personnas



Prénom : Jacques  
Age : 35 ans  
Profession : sans emploi  
Situation : célibataire

Jacques passe son temps libre à jouer sur son ordinateur. Il possède une tour récente faite pour le jeu. Il est à l'aise avec son ordinateur.

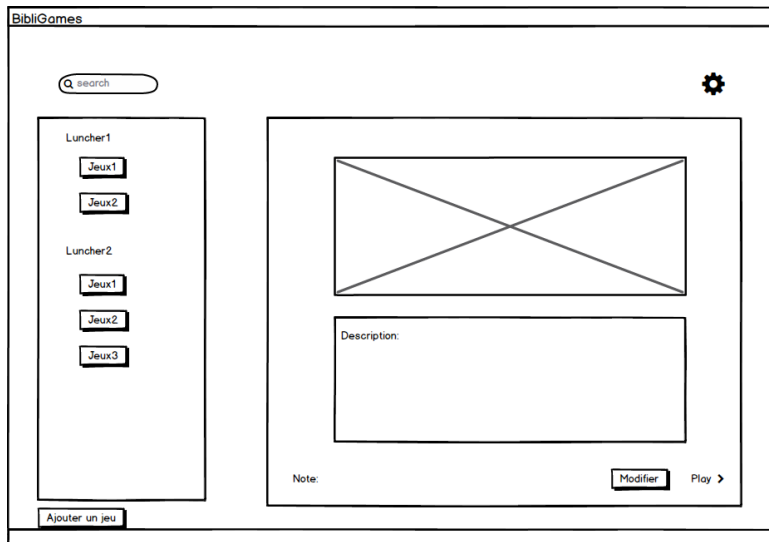
Il possède de nombreux jeux ce qui est assez encombrant. Il cherche donc une application ayant une vue d'ensemble de sa collection de jeux. Jacques attend de l'application qu'elle soit simple et épure et qu'il puisse facilement retrouver ses jeux voire même les lancer directement dans l'application.



Prénom : Timothée  
Age : 14 ans  
Profession : collégien

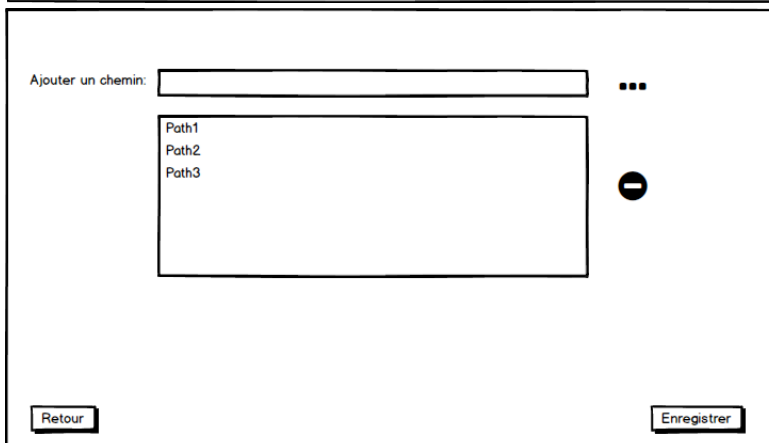
Prenons l'exemple de Timothée, un enfant de 14 ans qui ne joue que très occasionnellement (1 fois par mois environ) aux jeux vidéo. Il possède un ordinateur portable qui n'est pas récent. Le problème de Timothée est qu'il oublie quels jeux il possède et où ils se trouvent. Il se servirait donc de l'application pour pouvoir jouer à ses jeux de manière rapide sans chercher longtemps où est stocké le jeu sur son ordinateur. Si jamais l'application ne trouve pas ses jeux, il souhaite pouvoir rentrer manuellement le chemin vers ses jeux et cela, une seule fois. De plus grâce aux notes écrites dans le détail de ses jeux, il pourra savoir si ses jeux s'exécuteront de manière optimale sur son PC.

## Sketch



Vue 'principale' de l'application, elle permettra la sélection d'un jeu et l'affichage de ces détails.

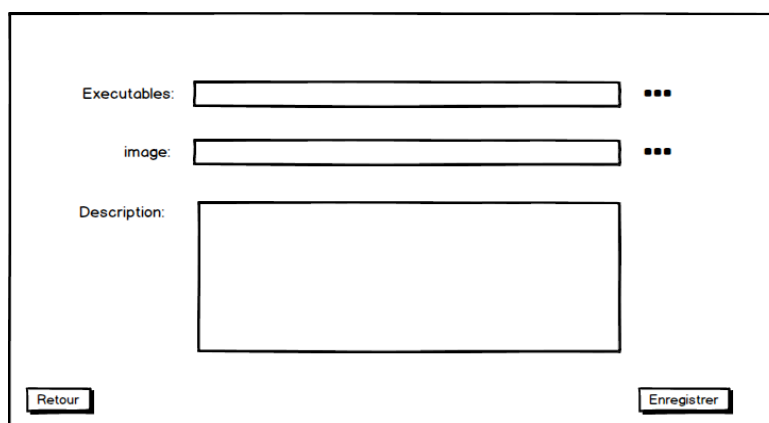
Elle permettra aussi d'ajouter un jeu, ouvrir les paramètres, lancer un jeu ou modifier les détails d'un jeu.



Vue 'paramètre' de l'application, elle servira à spécifier les chemins de recherches automatiques.

En cliquant sur les 3 petits points, on ouvrira l'explorateur de fichier. L'utilisateur pointera alors l'emplacement du dossier qu'il souhaite.

Après la sélection d'un chemin, on pourra cliquer sur 'moins' pour le supprimer.



Vue 'ajout détails' de l'application, elle permettra à l'utilisateur de saisir le chemin de l'exécutable, un chemin pour une image et une description.

En cliquant sur les 3 petits points, on ouvrira l'explorateur de fichier. L'utilisateur pointera alors l'emplacement de l'exécutable ou de l'image qu'il souhaite.

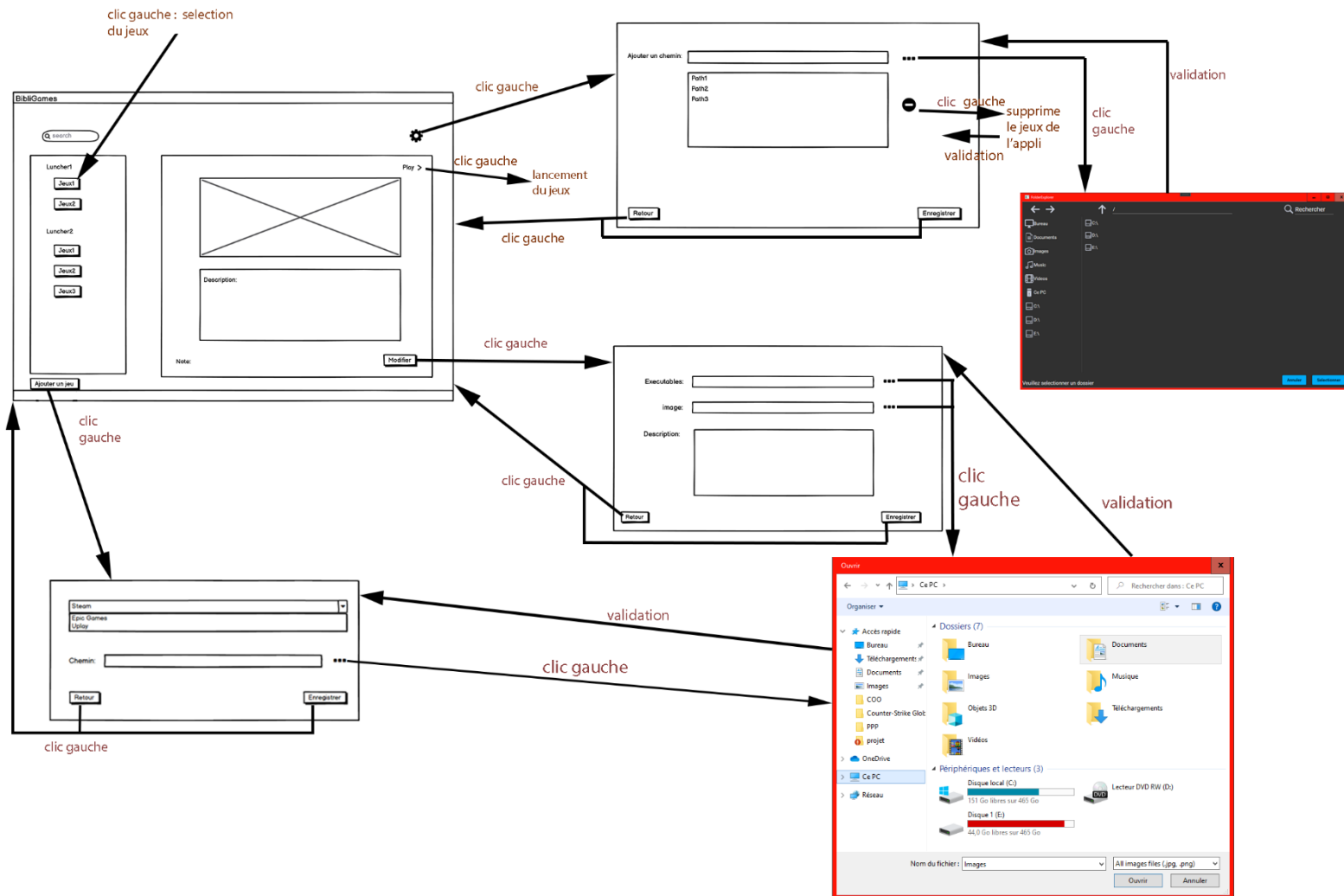
The screenshot shows a software interface for adding a game. At the top, there is a dropdown menu with 'Steam' selected, and below it, a list of launchers: 'Epic Games' and 'Uplay'. Below this is a text input field labeled 'Chemin:' followed by a file explorer icon (three dots). At the bottom, there are two buttons: 'Retour' on the left and 'Enregistrer' on the right.

Vue ‘ajout d’un jeu’ de l’application, elle permettra à l’utilisateur de renseigner un jeu.

Dans le menu déroulant l’utilisateur choisira avec quelle launcher associer le jeu.

En cliquant sur les 3 petits points, on ouvrira l’explorateur de fichier. L’utilisateur pointera alors l’emplacement de l’exécutable.

# Storyboard



## Diagramme de cas



## -Description du diagramme de cas

Nom cas : Chercher un jeu

Acteur : Utilisateur

Scénario nominal :

- 1 l'utilisateur cherche un jeu dans la barre de recherche
- 1.1 le système affiche les jeux correspondant à la recherche

Nom cas : Voir les détails d'un jeu

Acteur : Utilisateur

Scénario nominal :

- 1 l'utilisateur clique sur un jeu dans la partie master
- 1.1 le système affiche les détails du jeu dans la partie détail

Nom cas : Lancer un jeu

Acteur : Utilisateur

Cas inclus : Voir les détails d'un jeu

Scénario nominal :

- 1 l'utilisateur clique sur le bouton jouer
- 1.1 le système lance le jeu

Scénario alternatif :

- 1.1 le système ne trouve pas le jeu et lance une erreur.

Nom cas : Ajout d'une note

Acteur : Utilisateur

Cas inclus : Voir les détails d'un jeu

Scénario nominal :

- 1 l'utilisateur rentre ou supprime du texte
- 1.1 le système sauvegarde la modification

Nom cas : Modifier les détails d'un jeu

Acteur : Utilisateur

Cas inclus : Voir les détails d'un jeu

Scénario nominal :

- 1 l'utilisateur ajoute/modifie un exécutable et/ou une photo et/ou une description
- 1.1 le système modifie les éléments modifiés
- 2 l'utilisateur valide la modification
- 2.1 le système enregistre les modifications

Scénario alternatif :

- 2 l'utilisateur annule les modifications
- 2.1 le système annule les modifications

Nom cas : Afficher les jeux

Acteur : Utilisateur

Scénario nominal :

- 1 le système affiche les jeux de l'utilisateur dans la partie détail

Nom cas : Ajouter un jeu

Acteur : Utilisateur

Scénario nominal :

1. l'utilisateur clique sur le bouton « Ajouter un jeu »
  - 1.1. le système affiche la page pour ajouter un jeu
2. l'utilisateur entre le chemin du jeu
  - 2.2. le système ajoute le jeu à l'application

Scénario alternatif:

- 3 l'utilisateur annule son choix
  - 3.1 le système n'effectue pas l'ajout du jeu

Nom cas : Ajout d'un chemin de recherche

Acteur : Utilisateur

Scénario nominal :

1. l'utilisateur entre le chemin du dossier
  - 1.1. le système recherche des nouveaux jeux

Nom cas : Suppression d'un chemin de recherche

Acteur : Utilisateur

Scénario nominal :

1. l'utilisateur sélectionne un chemin du dossier
  - 1.1. le système supprime le chemin de recherche de l'application

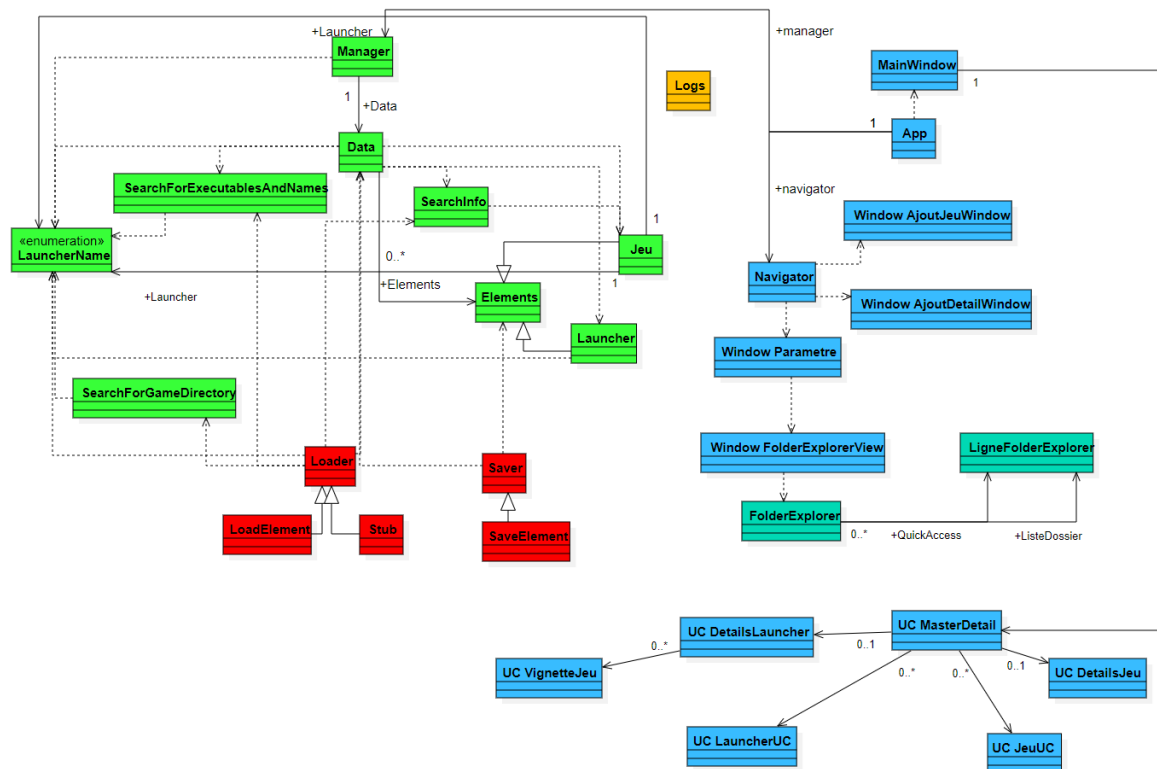


## -Diagramme de classe

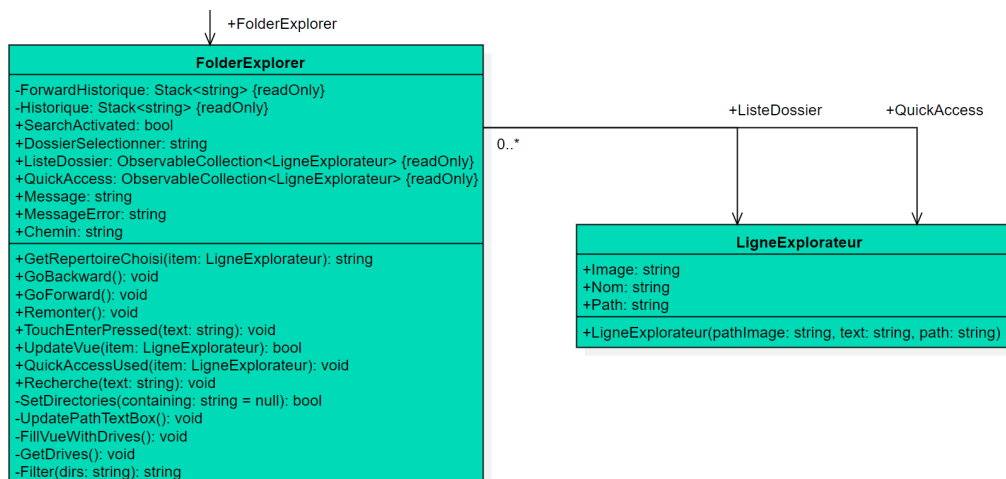
Pour notre application nous avons décidé d'utiliser le patron de conception structurel de la « façade ». L'utilisation de ce patron nous permet d'avoir une seule interface avec la vue. Donc pour toutes les actions de la vue autre que la navigation, une fonction de notre manager est lancée pour répondre à l'action demandée par l'utilisateur.

Ce manager est situé dans le paquet Modele dans la classe Manager.

### Diagramme de classe simplifié :

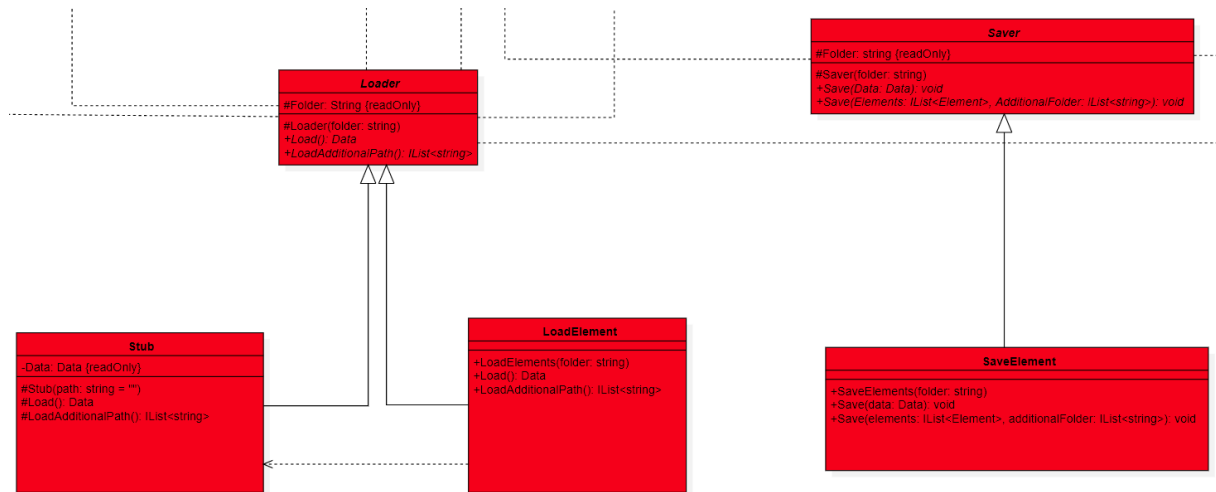


### Paquet FolderExplorerLogic :



Ces 2 classes implémentent la logique d'un explorateur de dossier

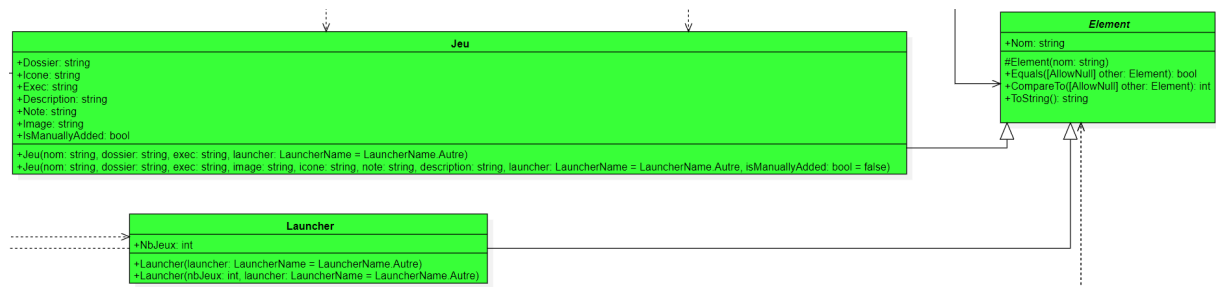
## Paquet DataManager :



La classe Loader et Saver sont 2 classes abstraites qui servent à la sérialisation. LoadElement et SaveElement implémentent les fonctions de sérialisation, et Stub sert à charger un jeu de test.

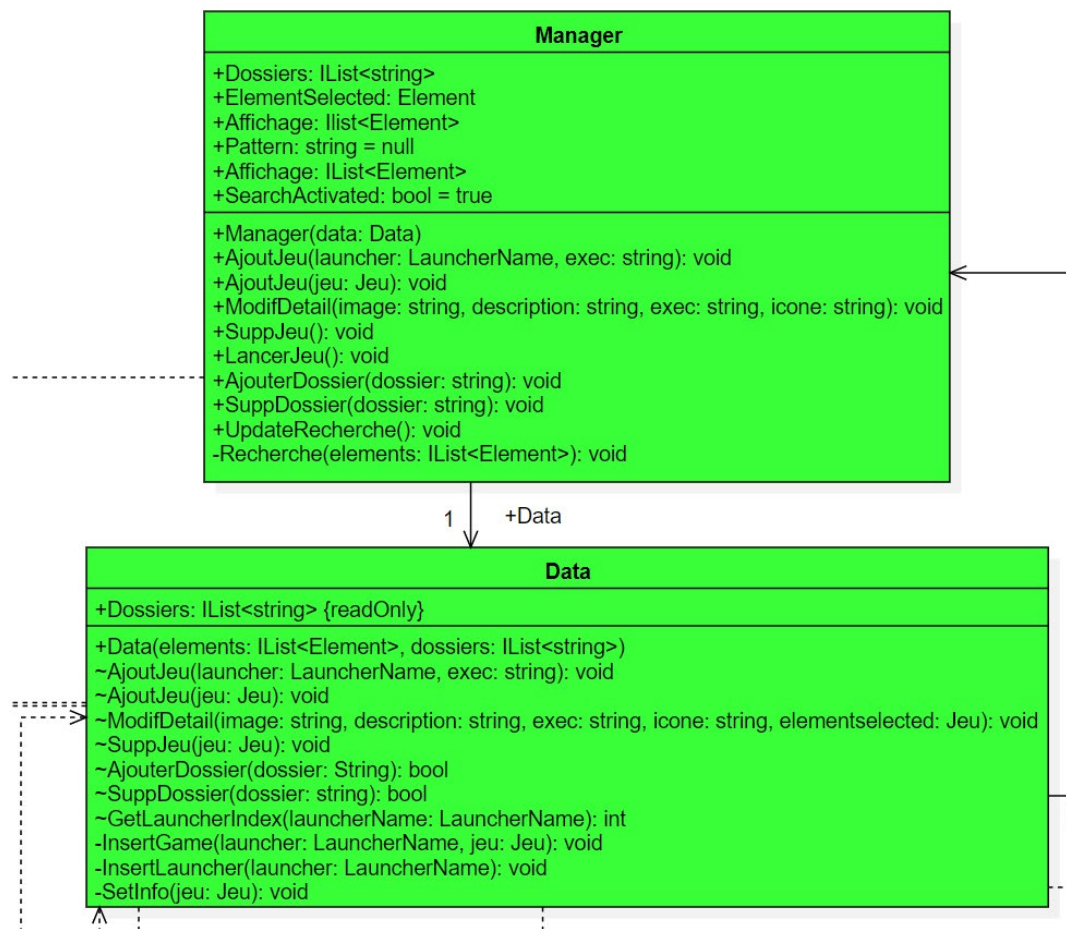
## Paquet Modele :

### -Partie Element :



Jeu et Launcher héritent d'Elément. Elément possède un attribut Nom et permet de manipuler les Jeu et Launcher au sein d'une seule liste via polymorphisme. Jeu regroupe les informations principales d'un jeu et Launcher possède le nombre de jeux qu'il contient. L'attribut IsManuallyAdded permet de savoir si le jeu a été ajouter par l'utilisateur ou s'il est trouvé par l'application.

### -Partie Manager/Data :



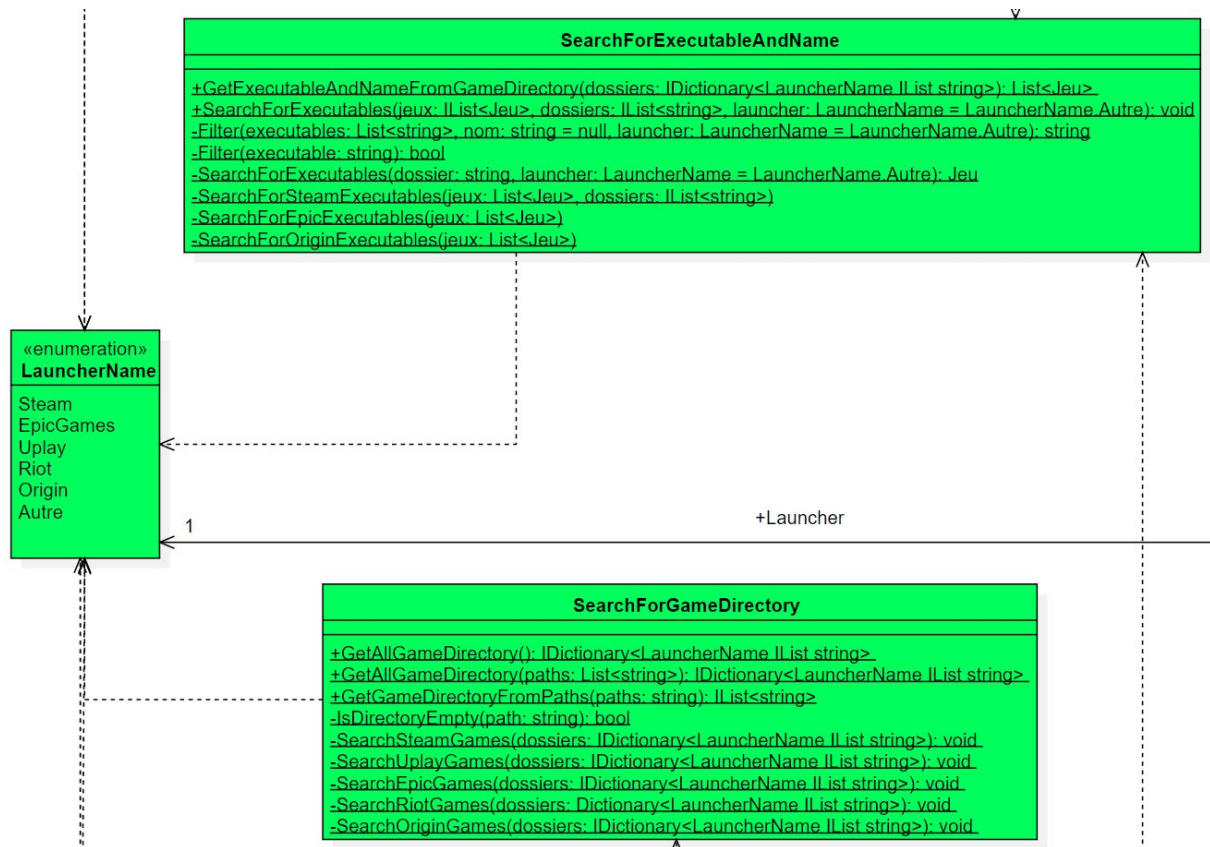
Le Manager relie la vue au model. Il dépend de Data qui permet de lancer les fonctions affectant les données. Data modifie les données et Manager ne sert que d'interface entre la vue et Data.

SetInfo() appelle la fonction SearchInfo.SetInfo() dans un nouveau thread.

InsertGame() insère les jeux par ordre alphabétique dans leurs différents launchers

InsertLauncher() insère les launchers par ordre alphabétique en laissant le launcher « Autre » toujours en dernier.

## -Partie SearchForExecutableAndName/LauncherName/SearchForGameDirectory



**SearchForExecutableAndName** permet de récupérer l'ensemble des jeux présent sur l'ordinateur du client à partir des dossiers récupérés par **SearchForGameDirectory**. **LauncherName** regroupe les principaux noms de launchers. **SearchForGameDirectory** permet de chercher les dossiers contenant des jeux.

**GetAllGameDirectory** permet de retourner tous les emplacements de jeux connu (trouver grâce aux fonctions **Search...Games**). Si la fonction prend un paramètre alors elle cherchera les jeux aussi dans les dossiers passer en paramètre.

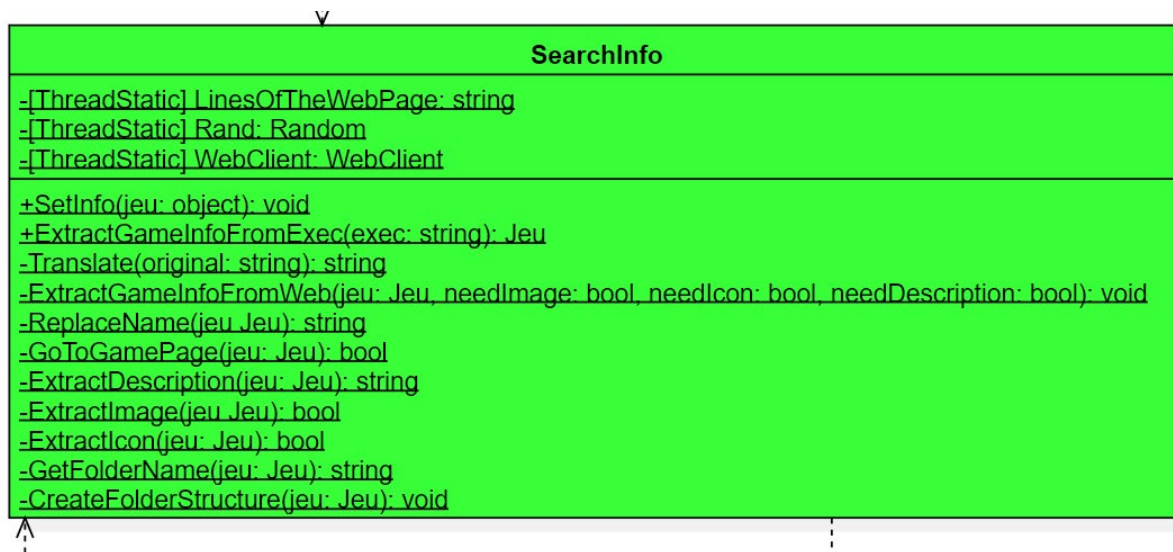
**GetExecutableAndNameFromGameDirectory** s'appuie sur la recherche de **GetAllGameDirectory** pour renvoyer une liste de jeu déjà avec certaine info pré-rempli (Dossiers/Nom/Exécutables/Launcher).

Pour récupérer ces infos la fonction s'appuie sur les fonctions **SearchFor...Executables**.

**SearchForExecutables** fait la meme chose mais à partir d'une liste de dossiers ou d'un dossier unique.

**LauncherName** est une enum qui contient les noms des principaux launcher.

## -Partie Eléments/ SearchInfo



SearchInfo permet d'aller chercher les informations d'un jeu sur internet tel que l'icône, la description et l'image. La classe possède des attribut ThreadStatic pour permettre de multithreadé la recherche d'info (qui peut prendre jusqu'à 2 sec par jeu voir plus).

SetInfo reçoit un objet car il doit être compatible avec le delegate ParameterizedThreadStart pour pouvoir lui donner un jeu en paramètre. SetInfo va regarder quelle sont les infos manquantes.

ExtractGameInfoFromExec sert à récupérer des infos (dossier, nom) à partir de l'exécutable.

Tanslate s'appuie sur un serveur de traduction pour traduire de l'anglais vers français.

ExtractGameInfoFromWeb lance les extractions nécessaires à partir des infos reçu par SetInfo.

ReplaceName sert à obtenir le nom du jeu modifié convenant au site qu'on utilise pour aller chercher les infos (IGDB).

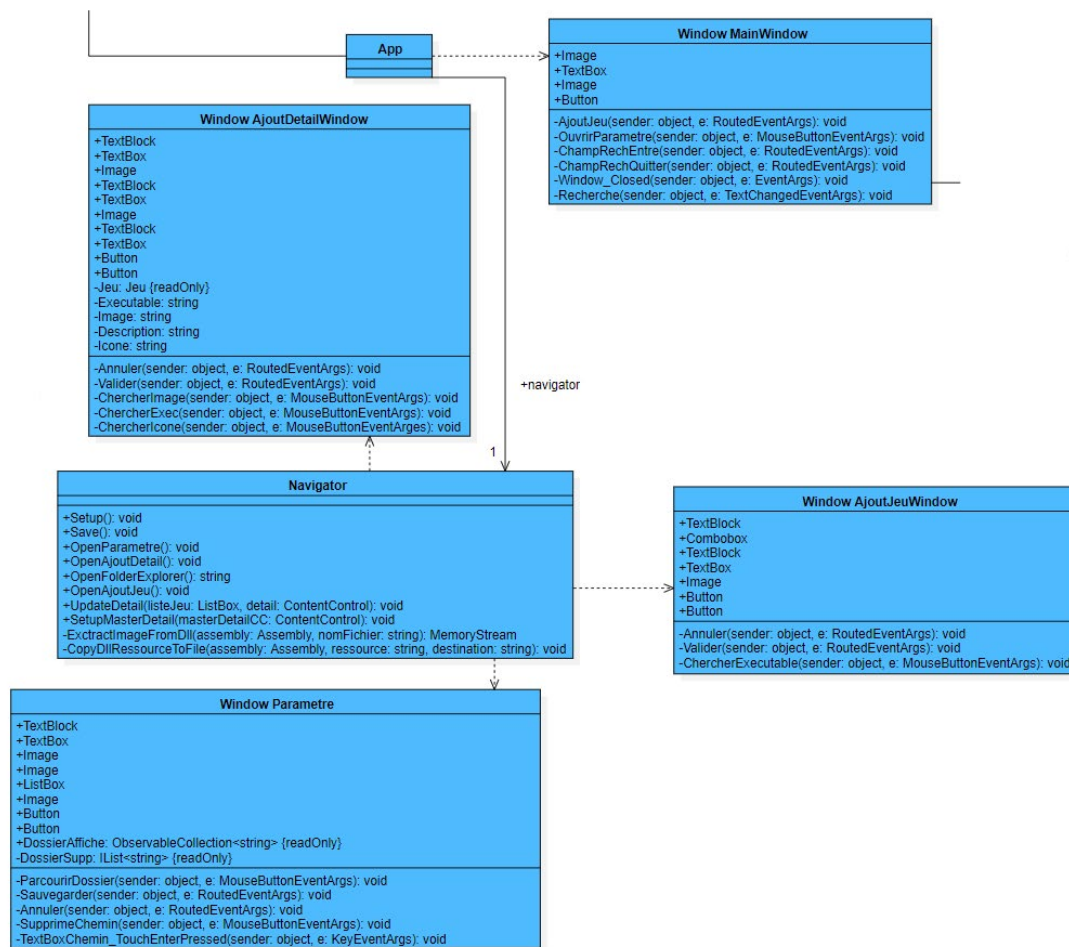
ExctractDescription/Image/Icon servent à extraire les infos.

GetFolderName renvoie un nom de dossier correspondant au spécification de Windows à partir du nom du jeu.

CreateFolderStructure s'assure de créer le chemin dans lequel les infos seront placé.

## Paquet Vue :

### -Partie MainWindow



App possède un Navigator de cette manière le Navigator est accessible à l'entièreté de la Vue.  
App possède un Manager de cette manière le Manager est accessible à l'entièreté de la Vue.  
Navigator est la classe qui gère la navigation de la vue, c'est elle qui modifie les ContentControl et ouvre les fenêtres.

Navigator.Setup() permet de chargé l'application à partir de la sauvegarde (stub ou réel) et Save() lance la sauvegarde. UpdateDetail permet de changer le contenu du ContentControl passer en paramètre afin d'avoir l'affichage souhaité par rapport au type de l'élément sélectionné dans ListBox. SetupeMasterDetail met un MasterDetail dans le ContentControl passer en paramètre.

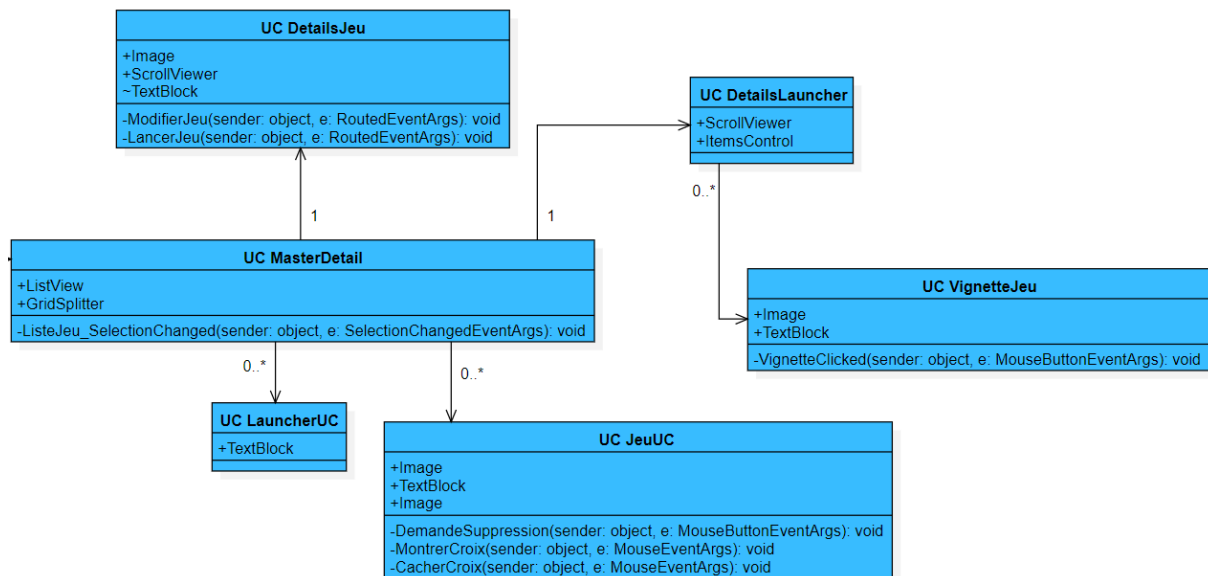
La MainWindow est la fenêtre d'accueil de l'application, c'est elle qui possède le Master Detail.  
Window\_Closed lance Navigator.Save().

AjoutDetailWindow permet de modifier les détail d'un jeu.

AjoutJeuWIndow permet d'ajouter un jeu.

Parametre permet d'ajouter un dossier contenant des dossiers de jeux.

## -Partie MasterDetail



MasterDetail permet de voir les différents launchers et jeux ainsi que leurs informations.

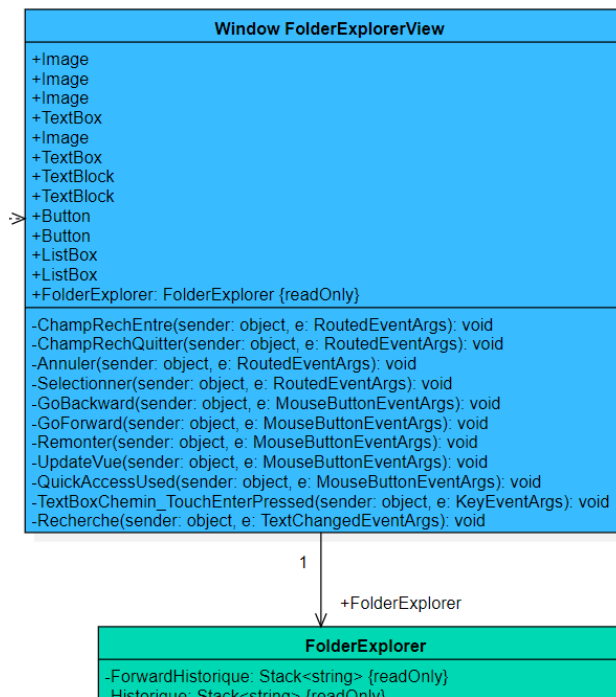
ListeJeu\_SelectionChanged() appelle Navigator. UpdateDetail()

DetailJeu et DetailLuncher corresponde à la partie détails de leurs types respectifs.

JeuUC et LuncherUC corresponde à la partie Master de leurs types respectifs.

VignetteJeu correspond à l'affichage d'un jeu dans le détail de launcher. VignetteClicked permet d'aller sur le jeu cliquer.

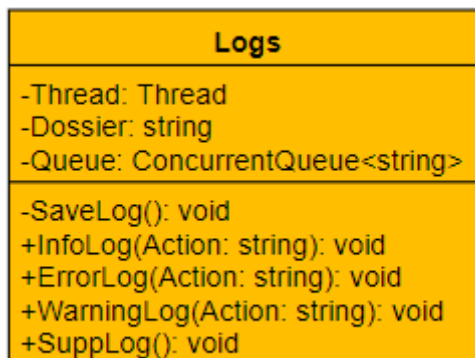
## -Partie FolderExplorerView



FolderExplorerView correspond à l’affichage d’un explorateur de dossier pour cela il s’appuie sur FolderExplorer présenter plus haut.

Chaque évènement appelle son équivalent dans le FolderExplorer.

## Paquet Logger :



La classe Logs sert à faire des Log de l’appli. Cette classe n’a pas de lien dans le diagramme de classe car elle pourrait être présente dans toutes les classes de l’appli. On laisse donc les développeurs l’utiliser quand bon leur semble.

Lorsqu’on ajoute une string dans la Queue on vérifie si le thread tourne ou pas et on relance le thread si jamais il était à l’arrêt.

InfoLog() ajoute une ligne concernant une info dans la Queue.

ErrorLog() ajoute une ligne concernant une erreur dans la Queue.

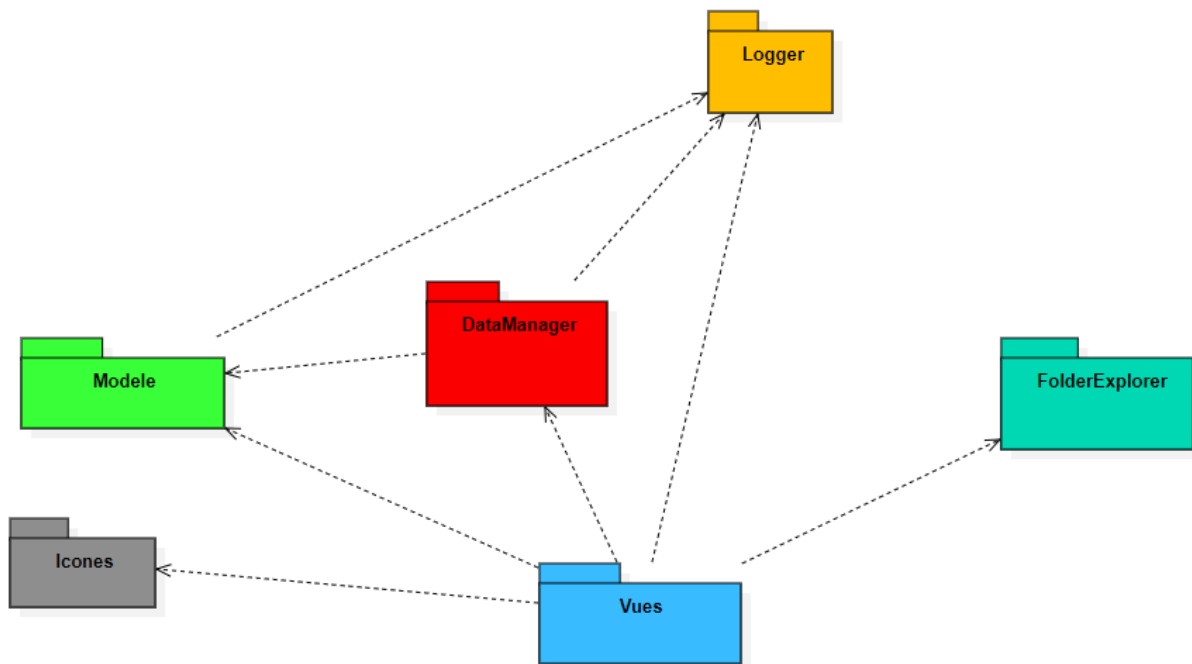
WarningLog() ajoute une ligne concernant une information importante dans la Queue.

SuppLog() réinitialise le fichier de log chaque jour.

SaveLog() est exécuté par le thread et tant que la Queue n’est pas vide il écrit dans un fichier (Dossier) le contenu de la Queue.



## -Diagramme de paquet



Les paquets FolderExplorer et Icones n'ont pas de dépendances, Vue est dépendant de tous les paquets. DataManager dépend de Modele et Logger et Modele depend uniquement de Logger.

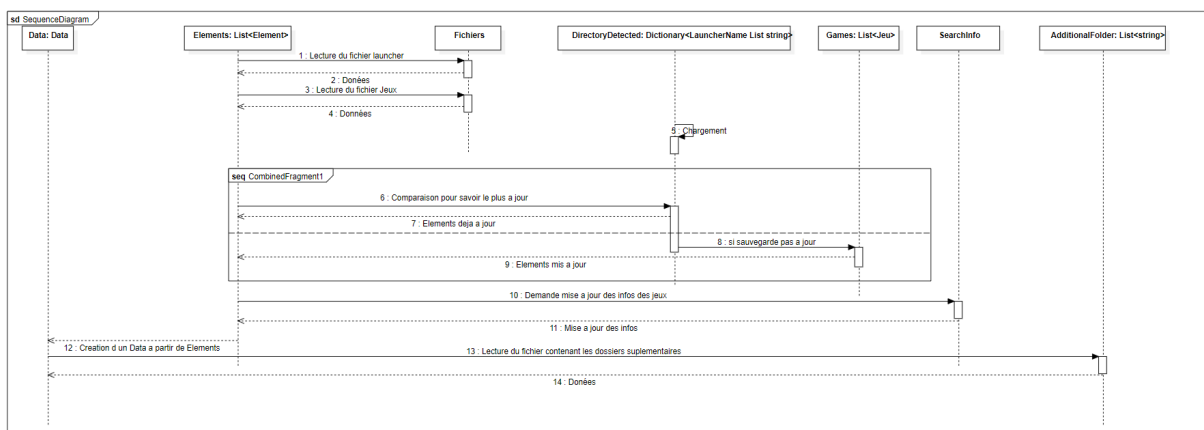
DataManager s'occupe de la persistance de l'application. Modele contient une classe Manager servant d'interface entre la logique de l'application et la Vue.

Le Logger aura pour but de permettre un débogage plus simple une fois l'application déployer chez le client.

Le paquet Icone permettra de mettre à jour les icones sans avoir à mettre à jour le paquet qui les contiens.

FolderExplorer implémente le comportement d'un explorateur de dossier.

## -Diagramme de séquence du loader



## -Participation personnelles

Tristan: Logger

Yoann: FolderExplorer (FolderExplorer+LigneFolderExplorer) + FolderExplorerView