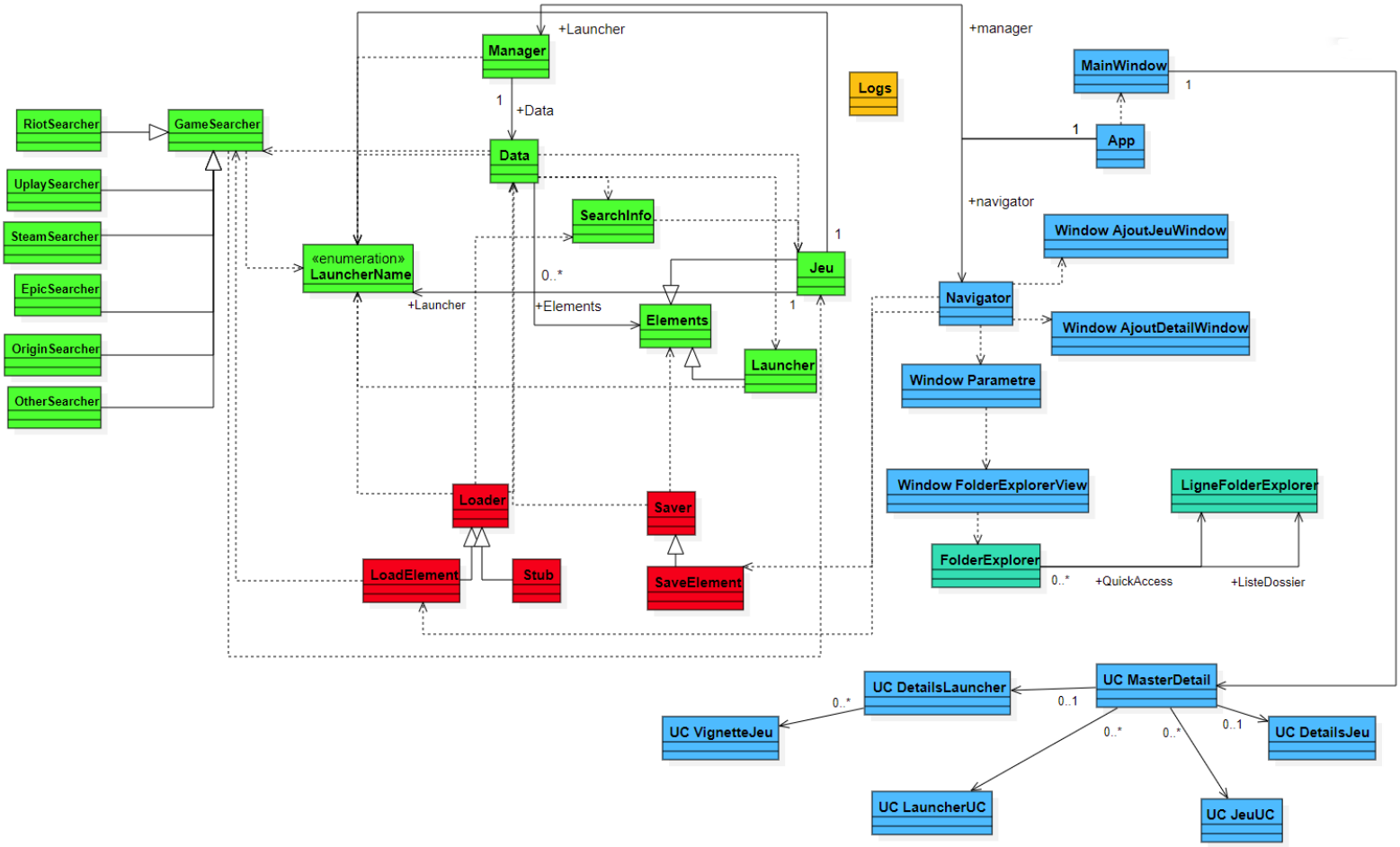
Documentation C#

-Diagramme de classe

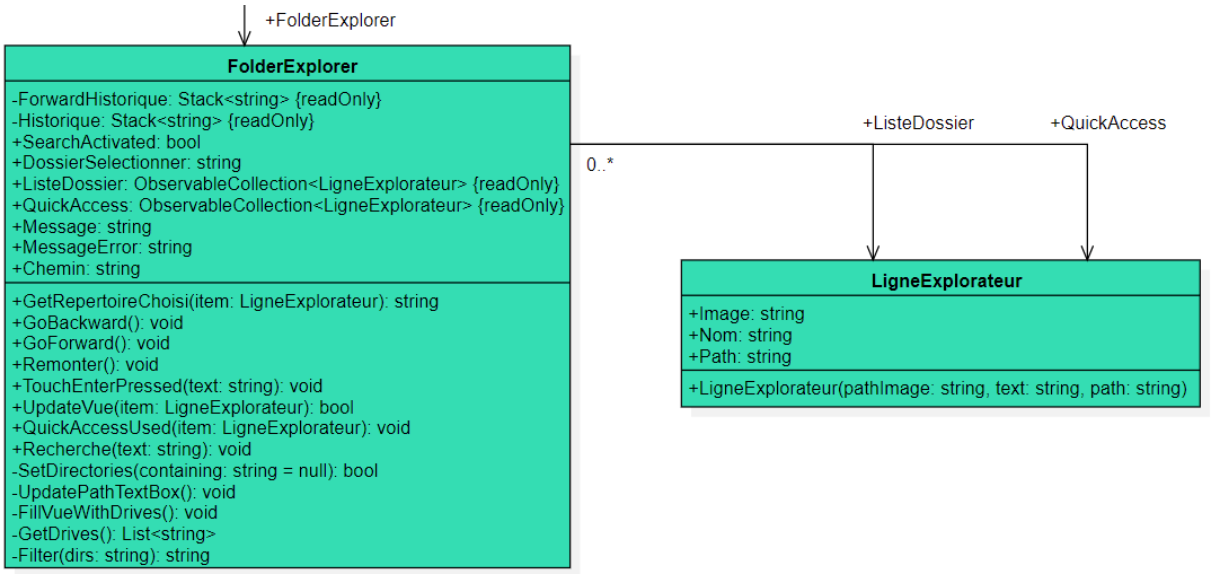
Pour notre application nous avons décidé d'utiliser le patron de conception structurel de la « façade ». L'utilisation de ce patron nous permet d'avoir une seule interface avec la vue. Donc pour toutes les actions de la vue autre que la navigation, une fonction de notre manager est lancée pour répondre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

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

Diagramme de classe simplifié :



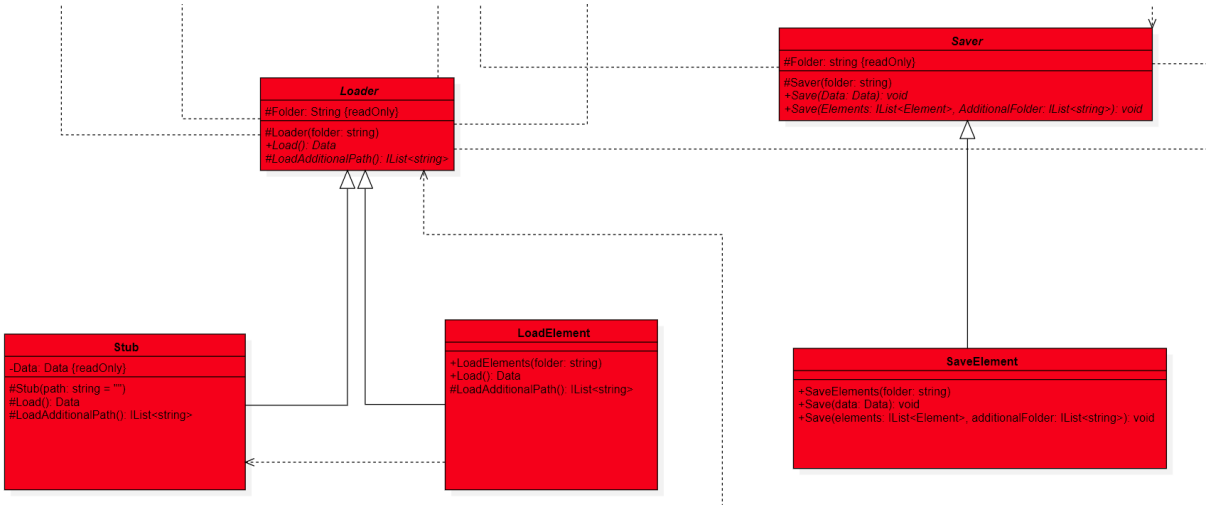
Paquet FolderExplorer :



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

LigneExplorateur représente une ligne dans l'explorateur c'est-à-dire une image et un nom. Il contient aussi son chemin correspondant. De cette manière on retourne le chemin du LigneEplorateur sectionné dans la vue (fonction GetRepertoireChoisi()).

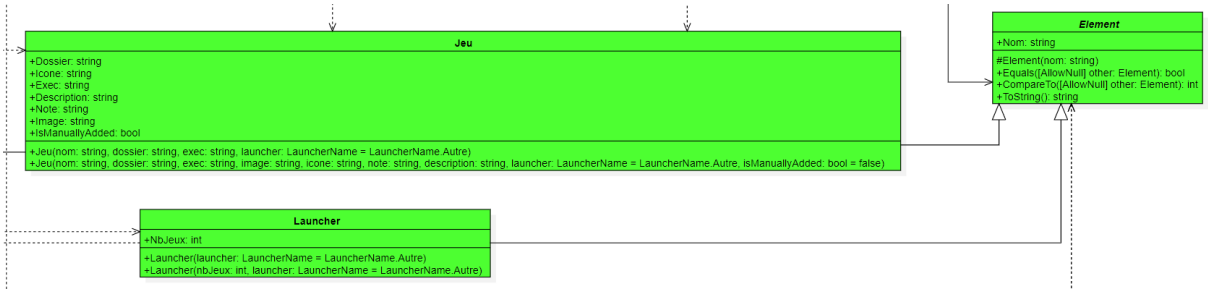
Paquet DataManager :



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.

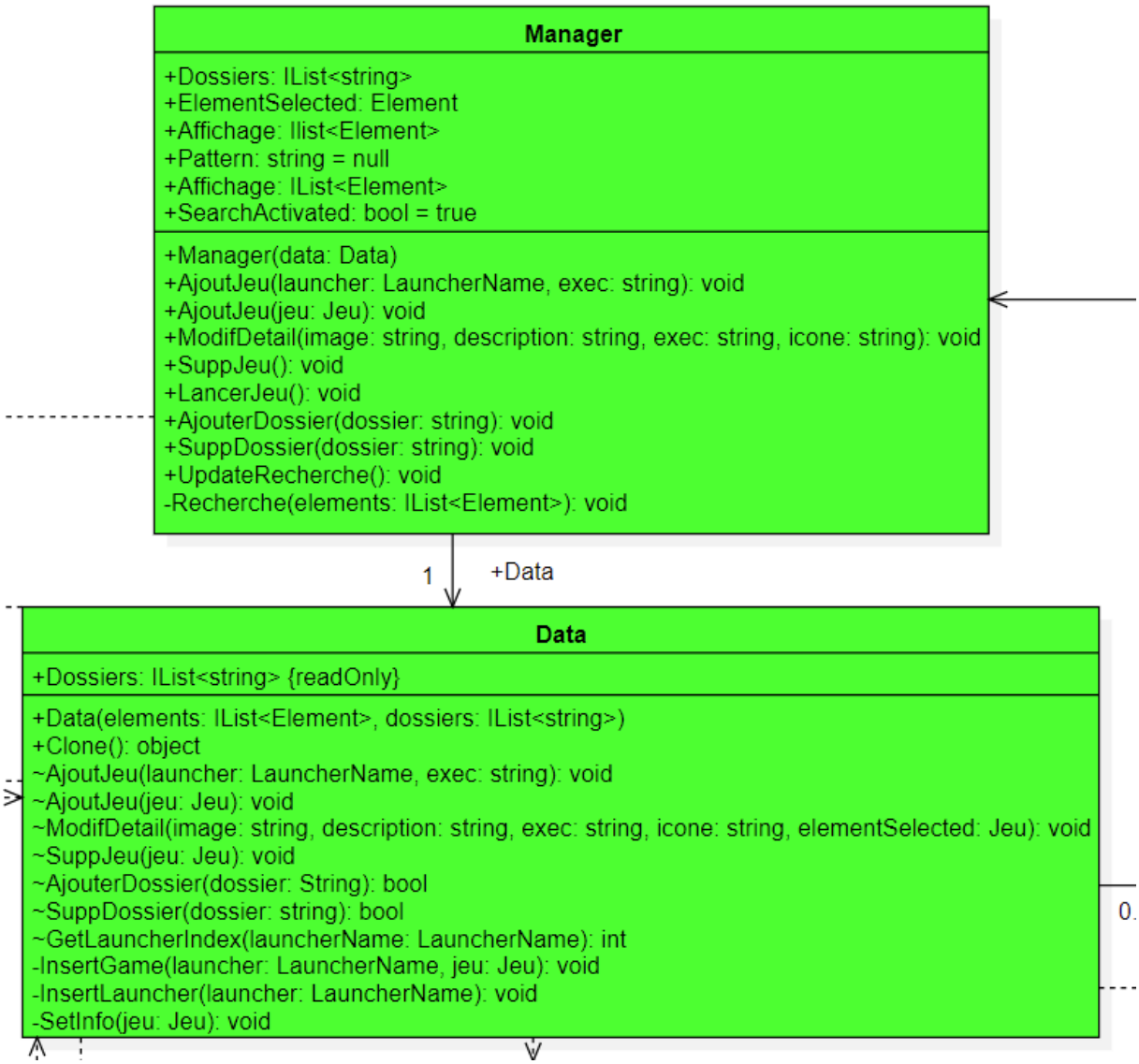
Paquet Modele :

-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 :



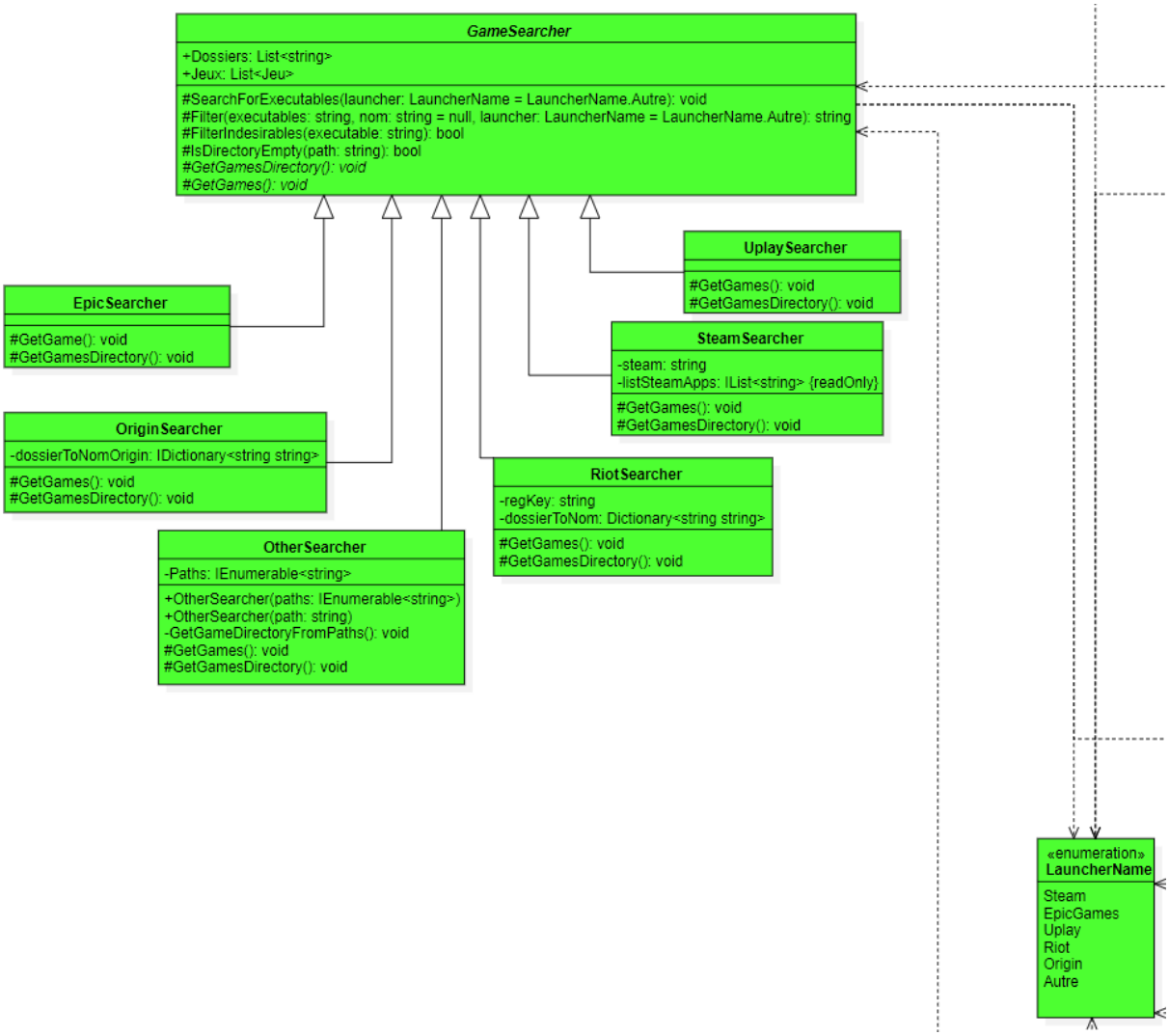
Le Manager relie la vue au model. Il dépend de Data qui permet de lancer les fonctions affectant les données. Data modifie les données et Manager ne sert que d'interface entre la vue et Data.

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

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

InsertLauncher() insert 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.

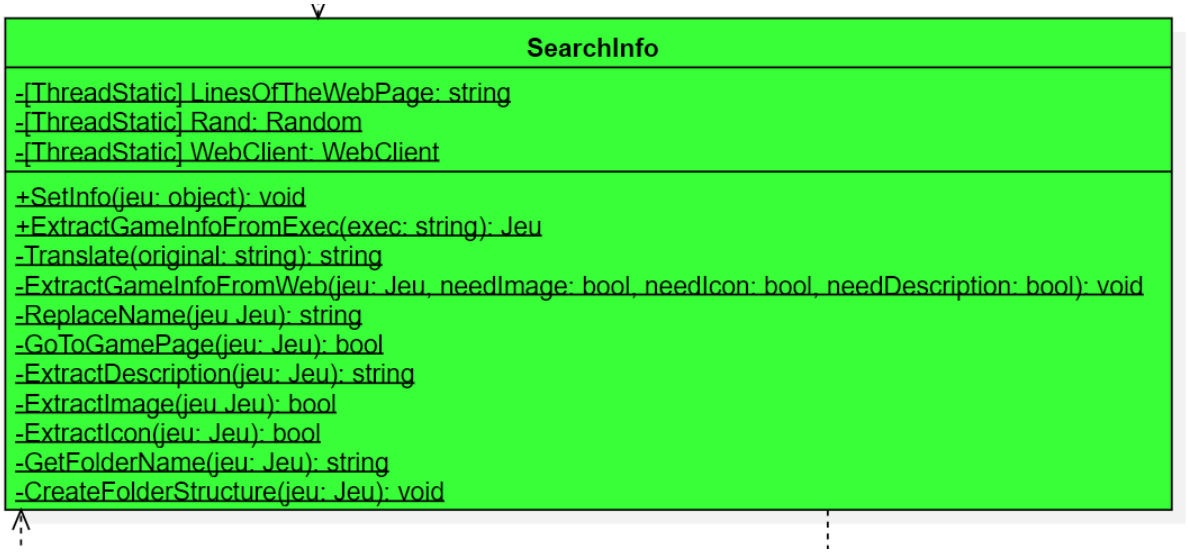
GetExecutableAndNameFromGameDirectory s'appuie sur la recherche de GetAllGameDirectory pour renvoyer une liste de jeu déjà avec certaine info pré-rempli (Dossiers/Nom/Exécutables/Launcher).

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 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 peux prendre jusqu'à 2 sec par jeu voir plus).

SetInfo reçoit un objet car il doit être compatible avec le deleguate 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 recu par SetInfo.

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

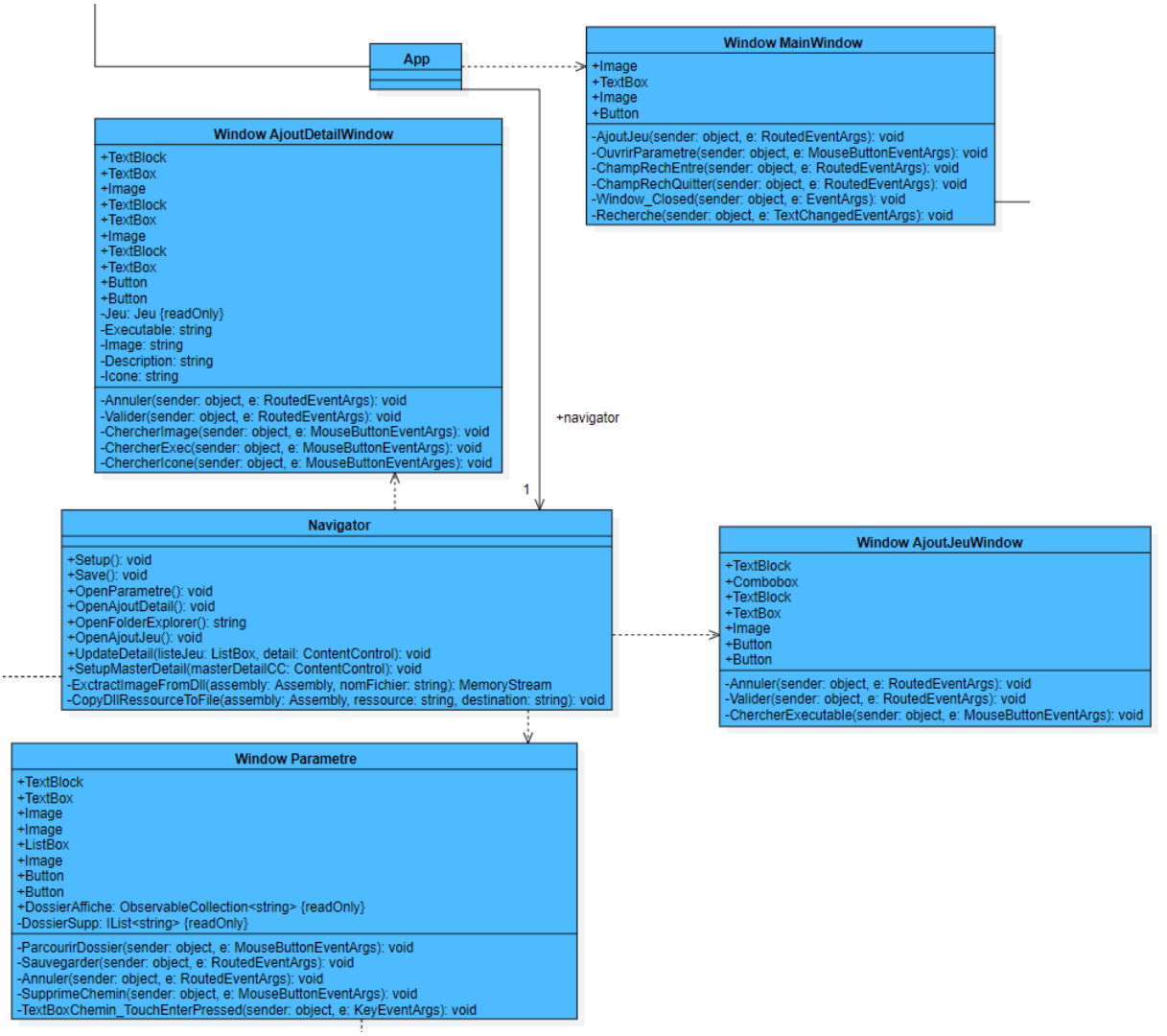
ExctractDescription/Image/Icon servent à extraire les infos.

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.

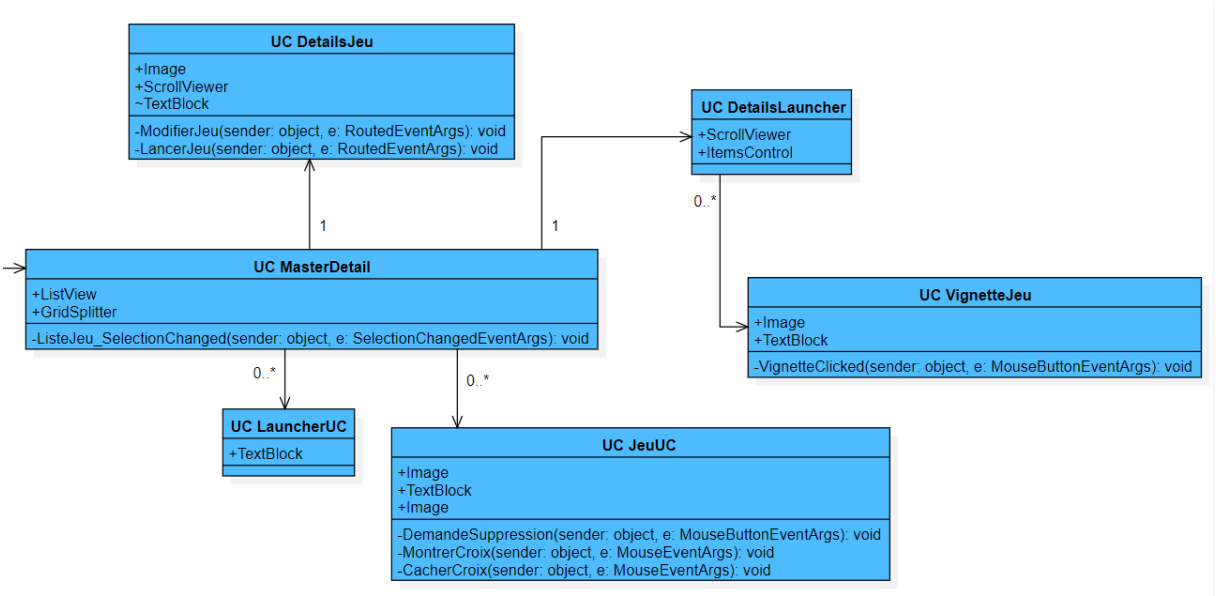
La MainWindow est la fenêtre d'accueil de l'application, c'est elle qui possède le Master Detail. Window_Closed lance Navigator.Save().

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



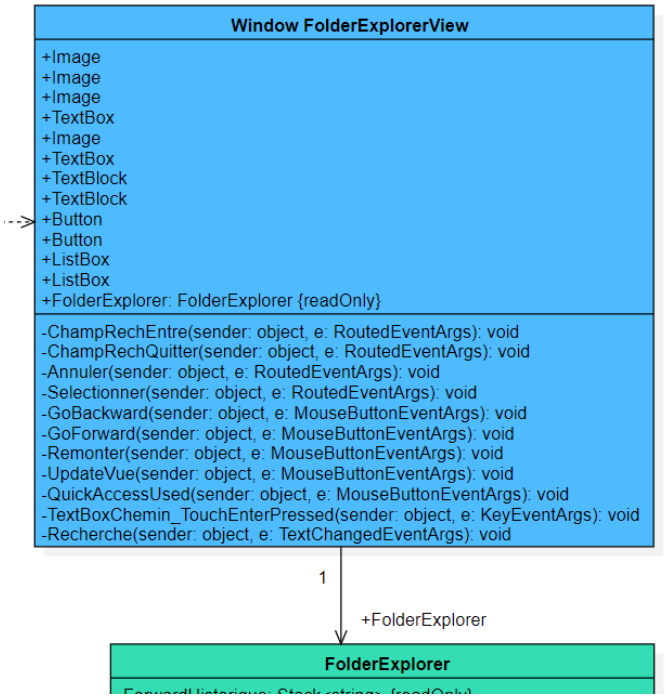
MasterDetail permet de voir les différents launchers et jeux ainsi que leurs informations. ListeJeu_SelectionChanged() appelle Navigator. UpdateDetail()

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.

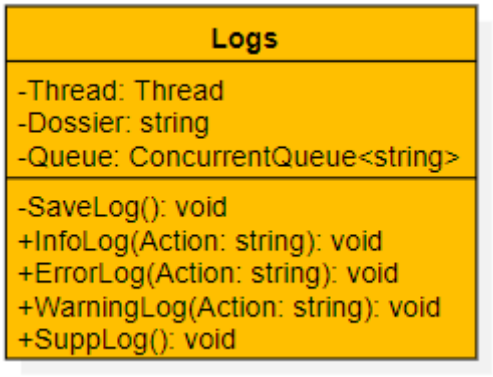
-Partie FolderExplorerView



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

Chaque évènement appel 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.

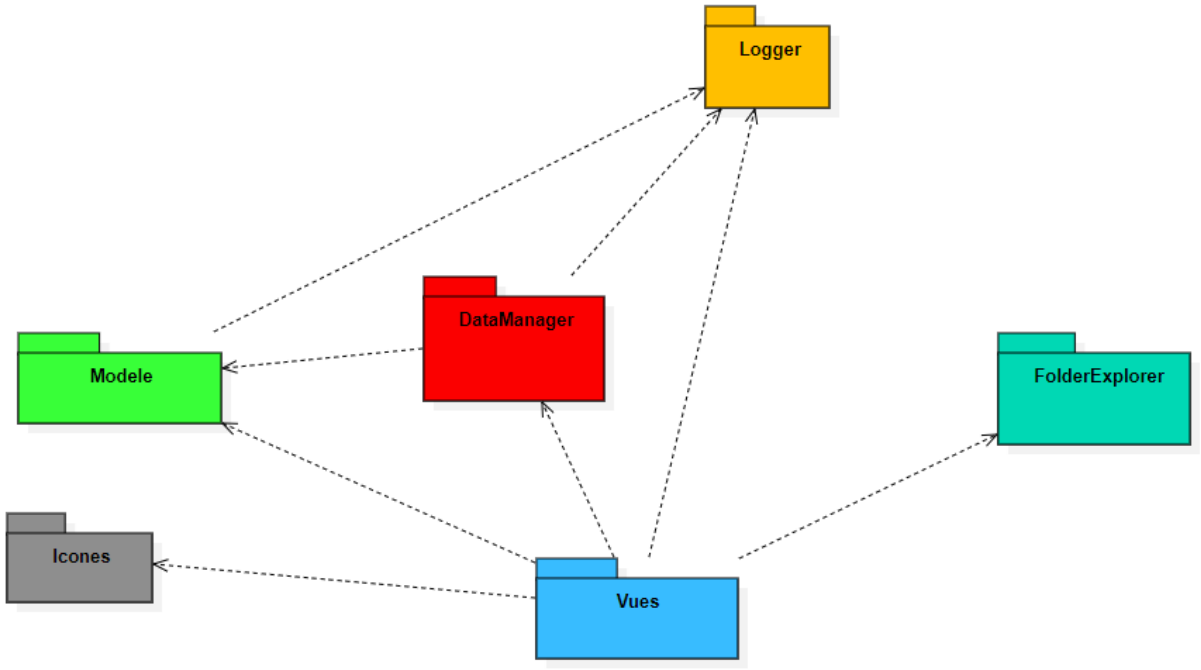
ErrorLog() ajoute une ligne concernant une erreur dans la Queue.

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.

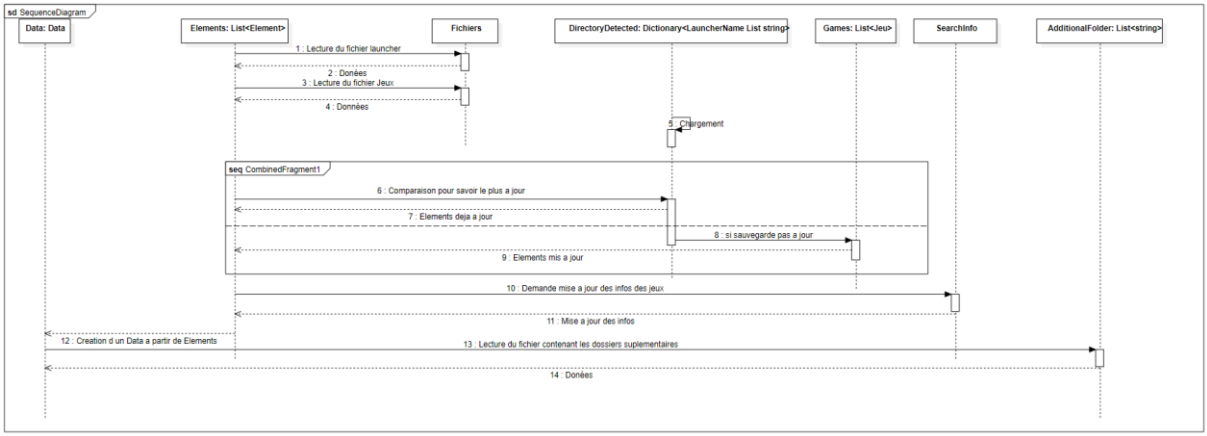
DataManager s'occupe de la persistance de l'application. Modele contient une classe Manager servant d'interface entre la logique de l'application et la Vue.

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


sd SequenceDiagram
Data: Data
Elements: List<Element>
Fichiers
DirectoryDetected: Dictionary<LauncherName List string>
Games: List<Jeu>
SearchInfo
AdditionalFolder: List<string>
1 : Lecture du fichier launcher
2 : Données
3 : Lecture du fichier Jeux
4 : Données
5 : Chargement
seq CombinedFragment1
6 : Comparaison pour savoir le plus a jour
7 : Elements deja a jour
8 : si sauvegarde pas a jour
9 : Elements mis a jour
10 : Demande mise a jour des infos des jeux
11 : Mise a jour des infos
12 : Creation d un Data a partir de Elements
13 : Lecture du fichier contenant les dossiers suplementaires
14 : Données