Description du diagramme de cas de la modélisation :

Le Manager possède un attribut de type IPersistanceManager, qui lui permet de charger les données de manière statique (Stub) ou bien depuis des fichiers (DataContract). Il possède aussi tous les Utilisateurs, les ContenusVideoludique en liste. Il possède aussi bien entendu l’utilisateur courant, pour savoir ce que l’on doit afficher.

La classe Utilisateur se trouve dans le package user, ainsi que le marathon, car un marathon est intimement lié à un utilisateur. Tout les ContenusVideoludiques déjà vu par l’utilisateur sont stockés dans ListCVvu, et Linq permet de connaitre le nombre de Contenu, de Film, de Serie (Anime exclu), et d’Anime déjà vu. Linq permet toujours de récupérer la liste filtrée selon la classe, mais aussi selon le fait que le contenu possède ou non un genre global ou d’anime. L’élément renvoyé sera un IEnumerable, renvoyé directement par Linq. Cela permet de construire ce qui est souhaité à partir de ceci (List, LinkedList…).

Marathon est possédé en un unique exemplaire par Utilisateur. Celui-ci possède une durée maximale de sa liste de lecture ListContenu. On peut ajouter et supprimer des IEstAjoutableAuMarathon (cf partie IEstAjoutableAuMarathon). On peut ajouter des thèmes aux dictionnaires, possédant en clé un genre, et en value la liste de tout les ContenuVideoludique. Cela permet de simplifier grandement l’algorithmie de création de la playlist de lecture, en permettant notamment d’ajouter à la playlist plus facilement un contenu vidéoludique dont le thème est déterminé à l’avance. De plus, cela permet d’enlever les contenus vidéoludiques déjà ajouté, afin d’éviter d’ajouter deux fois le même contenu.

La plupart des fonctions sont explicités plus en précision dans le diagramme de séquence, présentant la création de la liste de lecture du Marathon.

AddEpisodeLecture permet de répondre au souci de ne pas ajouter toute la playlist, mais seulement au maximum 3 épisodes pour chaque

Il existe 3 enums, deux pour les genres des contenus vidéoludiques, et une pour les langues.

Comme dis précédemment, ils sont dans des packages à la racine du projet de modélisation, pour être logique en réutilisation.

GenreGlobal possède un type none, car les Animes (possédant un genre global), peuvent ne pas en avoir en réalité, et c’est plus rapide que d’écrire « GenreAnime? Genre = null; ».

GenreAnime ne possède pas de none, car un anime possède forcément un genre d’animes.

IEstDescriptible est une interface, avec pour contrat le fait de posséder un string pour décrire la classe, ainsi qu’une méthode pour get la description.

IEstAjoutableAuMarathon permet de répondre au souci d’ajouter des ContenusVideoludique, et des Episodes dans la liste de lecture ListContenu des Marathons. L’interface est préférée au fait de construire a partir des 3 épisodes max choisis au hasard une nouvelle série, car plus léger en terme d’instanciation (pas d’intenciation d’une série) ; et aussi car plus simple.

Tous les films, séries et animes dérivent, directement ou non, de la classe abstraite ContenuVideoludique. Cela permet déjà de compacter le code, car le film et la série partages de nombreuses caractéristiques communes (titre, réalisateur…). De plus, cela permet de stocker dans une unique liste tout les contenus vidéoludiques, plutôt qu’une liste pour les films et une pour les séries. Cela permet donc de manipuler films et séries de manières communes. Anime dérive de série, car un anime est globalement une série, avec un genre d’anime en plus.

Film ajoute juste une liste d’acteurs, et ensuite Serie possèdes une liste de saisons, possédant eux même une liste d’épisodes. La saison ne possède pas réellement de date de début ou de fin, ni de durée, Linq est utilisée afin de déterminer la date minimale et maximale, ainsi que la durée totale des épisodes de la saison.

Anime ne possède rien de nouveau, c’est juste une série, avec une liste de GenreAnime en plus.