Labo WEM - Steamer

Lien GIT : https://github.com/lewisjaggi/wem

1. Contexte et objectifs du projet

L'objectif est de fournir un outil pour rechercher un jeu sur Steam en fonction de ces préférences et de ses habitudes.

On se chargera de fournir à l'utilisateur une liste de recommandation pertinente en fonction de sa requête, de ses jeux, des jeux de ses amis, ainsi que des avis utilisateurs.

2. Données

Sources

Sur Kaggle se trouve un dataset d'un ensemble d'environ 40'000 jeux Steam avec des données. Ce dataset contient les infos générales de chaque jeu.

La Steam Web API est utilisée pour récupérer les informations utilisateurs. Nous récupérons la liste des jeux d'un utilisateur et celle des amis. Si un jeu ne se trouve pas dans la base de données, nous effectuons une autre requête pour obtenir les détails du jeu qui est ensuite ajouté.

droit d'utilisation

Ce dataset est libre de droit avec une licence CCO : Public Domain disponible sur le site de Kaggle à l'adresse suivante : Steam games complete dataset

Cette source n'a que 10 mois, on peut donc la considérer encore comme suffisamment récent car les nouvelles sorties sont aujourd'hui suffisamment mises en avant.

Steam Web API est libre d'utilisation.

Description

Dataset

Voici la structure des données qui nous est fournis, elle comporte 20 colonnes et plus de 40'000 entrées (jeux) :

Colonne	Description
url	Url du jeu
types	Type de l'article
name	Nom du jeu
desc_snippet	Description courte du jeu
recent_reviews	Avis récent

Colonne	Description
all_reviews	Tous les avis
release_date	Date de sortie
developer	Développeur du Jeu
publisher	Publieur du jeu
popular_tags	Tags
game_details	Details du jeu
languages	Langages supportés
achievements	Nombre de succès
genre	Genre(s) du jeu
game_description	Description du jeu
mature_content	Description du contenu mature en jeu
minimum_requirements	Spécification matériel minimum pour le jeu
recommended_requirements	Spécification matériel recommandé pour le jeu
original_price	Prix sans réduction
discount_price	Prix avec réduction

Steam Web API

La requête de la liste des jeux retourne la liste des id des jeux ainsi que d'autres informations comme le temps de jeu que nous n'utilisons pas.

La requête d'un jeu retourne un objet semblable à ceux du dataset.

La requête de la liste d'amis retourne la liste des id des amis, ainsi que le type de relation et la date de la mise en relation.

Extraction (méthodes)

Nous utilisons un script en python afin de lire, désérialiser et filtrer le fichier CSV afin de les insérer dans une base de données NoSQL MongoDB.

Pour la Steam Web API, les réponses retournées des requêtes sont déjà extraites, traitées et rendues sous forme JSON qui est ensuite désérializé. Il se peut toutefois qu'il soit impossible de récupérer les données. Cela dépend des configuration de confidentialité du compte steam.

3. Etat de l'art

Il existe des centaines d'algorithmes de recommendations qui sont répartis dans différentes catégories :

- Recommandation Personnalisée

- Recommandation Objet
- Recommandation Sociale
- Recommandation Hybride

Le choix du type d'algorithme dépend de la collecte d'informations des utilisateurs. La collecte données explicite recueille les indications de l'utilisateur comme lorsqu'il remplit un questionnaire, like article, etc. La collecte de données implicites repose sur l'observation des comportements de l'utilisateur comme la liste des jeux qu'il a regardé après une recherche.

Recommandation Personnalisée

La recommandation Personnalisée se base sur la collecte de données implicites d'un utilisateur pour analyser son intérêt pour un objet.

Par exemple, si l'Adsense de Google repère qu'un utilisateur va souvent sur des forums de Dongeons & Dragons, il va proposer plus de publicités pour des livres de fantaisie.

Nous ne nous étendrons pas plus sur ce type de recommandation, car nous ne l'avons pas utilisé dans le cadre du projet.

Recommandation Objet

La recommandation Objet, au contraire des autres types de recommandation, n'est pas centrée sur l'utilisateur, mais centrée sur l'item. C'est un algorithme Content-Based.

Le principe ici est de comparer la similarité entre les caractéristiques d'un objet et les intérêts d'un utilisateur.

Les intérêts peuvent plus où moins correspondre directement à une caractéristique. Par exemple, un client qui veut acheter une voiture et a comme critère l'aspect "famille". Cela signifie qu'il n'est pas intéressé par une voiture avec beaucoup de cheveaux et profilés, mais une voiture 5 places avec un grand coffre et des gadgets de confort comme la climatisation, etc. Dans notre cas, l'intérêt de l'utilisateur est donné de manière explicite au travers d'un formulaire et correspond directement à une caractéristique. Il cherche un jeu avec le genre "Action" et certains jeux ont comme caractérstique le genre "Action".

Pour analyser la similarité des intérêts avec les critères, il y a plusieurs méthodes :

- Coefficient de Dice
- Indice de Jacquart
- Cosinus de similarité
- Etc.

Pour choisir une de ces méthodes, il y a plusieurs critères comme la rapidité, l'accuracy, etc. Il y en a un très important dans notre cas. Toutes les caractéristiques n'ont pas la même importance. Une caractéristique qui apparaît dans beaucoup de jeux identifie moins un jeu qu'une caractéristique qui n'apparaît que dans deux ou trois jeux. Il y a aussi une notion d'importance de la caractéristique au sein même d'un jeu. Si la caractéristique est mise en avant et apparaît plusieurs fois, elle est plus importante pour le jeu que si elle n'apparaît qu'une fois en fin de liste. Une technique généralement utilisé est le calcul du tf-idf vu en cours de DataManagement. Si on regarde les méthodes proposées, il n'est pas possible d'utiliser le coefficient de Dice, ni l'indice de Jacquart, car ces méthodes se basent sur des valeurs booléenes. Soit la caractéristique est là, soit non. Par

contre le cosinus de similarité travaillent très bien avec un tf-idf. On génère un vecteur de tf-idf pour un jeu et un autre pour un jeu fictif généré à partir du formulaire de recherche et on calcule le cosinus de similarité.

Il existe d'autres types d'algorithme de recommandation objet qui se basent sur la collecte de données utilisateurs implicites comme la méthode de retour de pertinence de Rocchio.

Par exemple, si on voit que l'utilisateur va souvent regarder les jeux Warhammer, l'algorithme va mettre en avant les jeux Warhammer.

Nous ne nous étendrons pas plus sur ce type de recommandation, car nous ne l'avons pas utilisé dans le cadre du projet.

Recommandation Sociale

La recommandation Sociale se base sur le comportement des utilisateurs similaires. Ces algorithmes utilisent généralement un mécanisme basé sur le voisinage proche. pour le faire, il effectue une corrélation entre des utilisateurs ayant des préférences et intérêts similaires.

Il existe différentes méthodes pour calculer cette corrélation:

- Coefficient de corrélation de Pearson r
- Coefficient de corrélation de Spearman
- Corrélation de Ringo

Ainsi, on analyse l'intérêt des voisins proches pour un jeu. On corrèle leur résultat. Les voisins avec des goûts similaires à l'utilisateur vont lui recommander le jeu et s'ils ont des goûts opposés, ils vont déconseiller le jeu.

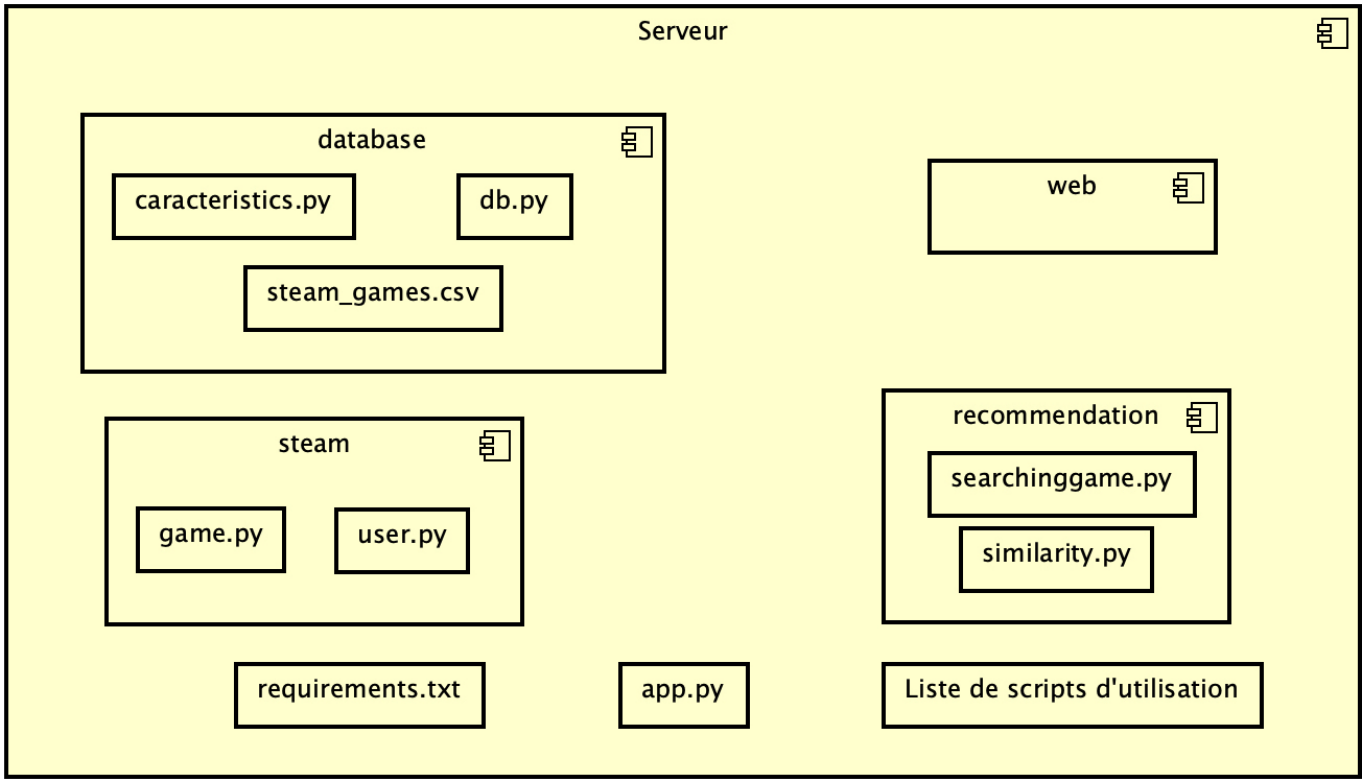
Recommandation Hybride

Cette recommandation consiste à combiner les pratiques des différents types de recommandation.

4. Architecture globale

Nous avons utilisé Python comme langage. Pour le traitement des données que nous avons stockées dans une base de données MongoDB, nous avons utilisé numpy et Scikit-learn.

Pour fournir un environnement à l'application, ainsi qu'une interface d'utilisation pour les utilisateurs, nous avons utilisé Flask qui intègre la base de données. L'interface fournit à l'utilisateur est une simple page html avec un formulaire mis en place à l'aide de Bootstrap.



Notre serveur contient différents composants. Le composant database contient les éléments liés à la base de données, le fichier CSV source, le fichier caracteristics.py qui contient les caractéristiques conservées lors de la validation et le fichier db.py qui contient le code pour instancier les objets liés à MongoDB et pour peupler la base de données.

Le composant steam contient les classes pour effectuer les requêtes à la Steam Web API et récupérer les informations sur les utilisateurs et les jeux.

Le composant recommendation qui contient la classe du jeu fictif appelée Searching Game et le fichier contenant les méthodes utilisées pour l'algorithme de recommandation comme le calcule du tf-idf.

A côté, il y a un fichier requirements.txt qui contient la liste des modules python nécessaires au bon fonctionnement du site, une liste de script pour installer et démarrer le serveur. Ainsi que le fichier app.py qui est le controlleur du serveur web.

5. Fonctionnalités

L'utilisateur peut choisir d'intégrer l'utilisation de la Steam Web API qui récupère ses données et ceux de ses amis steam. Pour ce faire, il peut choisir de mettre ou non son id steam dans le champs. Un petit lien l'amène a un tutoriel pour trouver son id steam.

L'utilisateur peut faire une sélection de filtres pour sa recherche de jeu. Il peut tout d'abord filtrer sur une gamme de prix et un niveau d'avis positif minimum.

Il peut ensuite sélectionné une langue ou plusieurs langues. S'il ne sélectionne pas de langue, le filtre n'est pas utilisé.

Il peut ensuite sélectionnés les caractéristiques du jeu qui l'intéresse au travers des genres, des tags populaires et des game détails. S'il ne sélectionne aucune caractéristique, nous considérons qu'il ne recherche rien et nous retournons une page sans jeu.

6. Techniques, algorithmes et outils utilisés

Algorithme d'extraction de données

Le fichier CSV source de données n'est pas très bien normalisé:

- Cellules vides
- Cellules à NaN
- Cellules qui devraient contenir un nombre, mais qui contiennent du textes
- Cellules contenant une liste d'attributs non-normalisées
    - Tous les objets n'ont pas la même liste de caractéristiques
- Etc.

Il a fallut mettre en place un système qui nettoie les valeurs et génères des dictionnaires pour les listes d'attributs, ceci afin de normaliser la base de données.

Une fois l'objet normalisé, on pré-processes un maximum de traitement pour l'algorithme de prédiction. On calcule le tf-idf qu'on stocke dans la base de données de chaque objet. On calcule est intègre en tant qu'attribut de l'objet, son score d'avis positif.

On stocke séparément les données utilisées séparément des objets comme les caractéristiques avec leurs statistiques d'utilisation. Ceci afin de récupérer l'information plus rapidement pour le formulaire ainsi que le calcule du tf-idf de la requête.

Nous avons mis en place un système de validation des caractéristiques conservées pour chaque objet. Nous expliquons pourquoi dans le chapitre d'optimisation.

Algorithme de recommandation

L'algorithme a évolué au fil des itérations. Nous sommes partis sur une recommandation de type Objet, car nous n'avons aucune donnée utilisateur au départ. Il est possible de récupérer quelques données à l'aide de la Steam Web API quand les configurations de confidentialité des comptes steam le permet.

Nous avons donc mis en place un formulaire qui nous permet de générer un jeu factice en fonction des intérêts que nous donne de manière explicite l'utilisateur. Nous calculons ensuite la similarité entre ce jeu factice et la liste des jeux de la base de données à l'aide du cosinus de similarité des tf-idf des caractéristiques des jeux.

Le vecteur de tf-idf d'un jeu se base sur les genres, les tags populaires et les game détails, pour chacun d'entre eux on applique la formulaire suivante:

```
tf-idf = nombre d'apparition de la caractéristique dans le jeu * log(nombre total
de jeu / nombre de jeu contenant la caractéristique) * ranking + 1
```

La première partie correspond au calcule classique d'un tf-idf, mais nous avons intégré deux choses. Le ranking qui est calculé à partir de la position des apparitions de la caractéristique. Le "+1" a été intégré, car si les valeurs sont trop petites, la perte de précision au fil de l'algorithme fait qu'on obtient des scores à 0 régulièrement.

Le calcule du cosinus de similarité se fait de manière classique avec numpy pour paralléliser le calcul.

Nous avons ensuite généré un score en fonction du pourcentage d'avis positif des utilisateurs au pro-rata du nombre de votes. Ce score a une valeur entre 0 et 1.

Puis, nous avons décidé d'intégrer un score pour chaque jeu en fonction du nombre de caractéristiques en commun avec le jeu factice. Ce score vaut entre 1 et 2.

A ce stade-là, nous avons le coeur de l'algorithme qui correspond a une recommandation Objet. Nous avons ensuite intégré une recommandation Personnalisée en fonction de la bibliothèque de jeu de l'utilisateur. Nous créons un autre jeu factice en générant une sorte de moyenne de jeu à partir de cette bibliothèque. Puis nous faisons à nouveau un cosinus de similarité avec l'ensemble des jeux pour calculer un score.

Nous avons finalement intégré une recommandation Sociale basée sur la bibliothèque des jeux des amis. Nous améliorons le score d'un jeu si ce jeu est présent dans la bibliothèque d'un ami. Nous avons intégré le coefficient de Pearson r, souvent utilisé pour la similarité utilisateur, pour que le score soit affecté en fonction des goûts communs entre l'utilisateur et son ami. Si leur goût s'oppose le score est baissé, mais s'ils ont beaucoup de jeux communs, cela augmente fortement le score.

Une analyse sur les clics des liens des jeux recherchés avait été envisagé, mais abandonné à cause de temps de recherche trop long au profit d'optimisations.

Optimisations

Arrivé à un certain développement de l'algorithme, nous obtenions des timeout de requêtes au bout de quinze minutes d'attentes environ. Nous avons par conséquent dû fortement optimisé notre algoritme pour qu'au final la pire requête possible s'en ajout de jeu dans la base de données dure environ 1 minute et 50 secondes.

Nous avons commencé par pré-processé un maximum, notemment les tf-idf des jeux et le calcul du score des avis utilisateurs.

Nous avons ensuite mis en place des filtres pour diminuer le nombre de jeux traiter. Nous filtrons sur la langue, au niveau minimum d'avis positif, sur une gamme de prix, ainsi que sur le fait que les jeux doivent contenir au moins une caractéristique commune.

Le calcul du cosinus de similarité des vecteurs prenaient énormément de temps, car les vecteurs avaient environ 440 dimensions. Il a donc été décidé de sélectionner une certaine liste de caractéristique quitte à ce qu'un jeu n'aie pas de caractéristique. Cela à réduit les vecteurs à 57 dimensions.

optimisation accès base de données

Pour optimiser le traitement des calule sur le server, nous voulions récupérer la totalité des données du la base de données avec la methode :

```
<Class>.objects()
```

maleureusment, cette methode nous retournais un objet de type `QuerySet`. Nous n'avions donc pas toutes les données, mais un objet lazy qui ne récupère une donnée que dés que l'on y accède (Instance de document). Nous avons donc utilisé l'appel suivant afin de récupérer toutes les données en une fois.

```
<Class>.objects().as_pymongo()
```

Au lieu de renvoyer des instances de Document cette methode renvoy les valeurs brutes de pymongo.

Elle est particulièrement utile si l'on a pas besoin de déréférencement et que l'on se soucie de la vitesse de récupération des données, ce qui était notre cas.

Optimisation multithreading

Une grande partie du traitement de la requête consiste à calculer le "cosinus similarity" entre le vecteur de la requête et les jeux présents dans la base de données. Ces calculs prenaient beaucoup de temps, c'est pourquoi nous avons décidé de l'effectuer en parallèle. Pour le traitement en parallèle, on utilise l'objet "Pool" du package "multiprocessing" de Python. Cet objet permet de créer des nouveaux processus python qui s'exécutent sur différent cœur du processeur. En créant le Pool, il est possible d'appeler une fonction à chaque fois qu'un processus est initialisé. Cette fonction permet de créer des variables globales qui pourront être utilisé dans les processus. Dans notre algorithme, on crée un Pool avec un nombre de processus qui correspond au nombre cœur de la machine. Chaque processus appelle la fonction d'initialisation qui crée les variables globales utiles aux calculs. Ensuite, on effectue un "imap_unordered" qui exécute un map en parallèle non ordonné (un map non ordonné est plus rapide, car il ne garde pas l'ordre de la liste initiale). Le map applique la fonction de calcul de "cosinus similarity" sur toute la liste de jeu.

7. Planification, organisation et suivi répartition du travail

WorkPackage

	Gestion de projet		Analyse					Conception	Fonctionnalités		
	Planification	Meetings de suivi	Choix des technologies	Mise en place d'un outil de versionning	Mise en place de l'hébergement du site	Etat de l'art des algorithmes de propositions	Analyse de Steam Web API	Use case de recherche de jeux	Définition de la structure de la base de données	Définition des paramètres du formulaires	Définition de la présentation du résultat
Délivrables:	**Diagramme de Gantt**	**Aucun**	**Environnement de développement**			**Documentation**		**Diagramme Use Case**	**Diagramme de base de données**	**Maquettes**	**Maquettes**

	Implémentation										Tests			
	Script			Frontend			Backend			Database				
	Récupération des données du fichier CSV	Traitement, mise en forme et formatage des données	Insertion des données dans la base de données	Formulaire	Envoi des données	Traitement des données retournées	Traitement des données du formulaire	Filtrer les données selon le formulaire	Création de cluster selon des critères de recherche	Récupération des données depuis la Steam Web API	Mettre en place la base de données	Tests du formulaire	Tests des requêtes à la base de données	Tests de la Steam Web API	Tests de l'aglorithme
	Exécuter le script et ajouter les données à la base de données			Affichage des résultats de la recherche			Retour d'une liste de propositions							
Délivrables:	**Script**			**Fichiers HTML/CSS/JS**			**Code source**			**Base de données**	**Documentation des protocoles de tests**			

Diagramme de Gantt

			01/05/2020 Cahier des charges SP-10	08/05/2020 SP-11	15/05/2020 SP-12	22/05/2020 Ascension SP-13	29/05/2020 SP-14	05/06/2020 Rendu 08.06.2020 SP-15
Gestion de projet		Planification						
		Meetings de suivi						
Analyse		Choix des technologies						
		Mise en place d'un outil de versionning						
		Mise en place de l'hébergement du site						
		Etat de l'art des algorithmes de propositions						
		Analyse de Steam Web API						
Conception		Use case de recherche de jeux						
Fonctionnalités		Définition de la structure de la base données						
		Définition des paramètres du formulaires						
		Définition de la présentation du résultat						
Implémentation	Script	Récupération des donnée du fichier CSV						
		Traitement, mise en forme et formatage des données						
		Insertion des données dans la base de données						
		Exécuter le script et ajouter les données à la base de données						
	Frontend	Formulaire						
		Envoi des données						
		Traitement des données retournées						
		Affichage des résultatas de la recherche						
	Backend	Traitement des données du formulaire						
		Filtrer les données selon le formulaire						
		Création de cluster selon des critères de recherche						
		Récupération des données depuis la Steam Web API						
		Retour d'une liste de propositions						
	Database	Mettre en place la base de donnée						
Tests		Tests du formulaire						
		Tests des requêtes à la base de données						
		Tests de la steam Web API						
		Tests de l'aglorithme						

Absence de Charles-Lewis Jaggi pour la protection civile

8. Conclusion

Nous sommes plutôt satisfaits du résultat, malgré de nombeux soucis rencontrés. Nous avons pu entre-apercevoir la complexité d'un algorithme en temps-réel. Il est nécessaire de faire un choix entre la qualité de l'algorithme et la vitesse d'exécution. On ressent nettement le temps de l'accès à la base de données ou l'API.

Il y a de nombreux aspects qu'on peut améliorer comme utiliser les habitudes des utilisateurs qu'on monitore. Seul l'utilisateur sait ce qui lui plaît, on peut analyser cela.

On aimerait aussi plutôt utiliser des requêtes AJAX pour pouvoir fournir des feedbacks à l'utilisateur pour qu'il sache l'avancement.

De nombreuses données ont été mis de côté pour des améliorations de filtrage et de recommandation.

Nous avons été frustré par le jeu de données auquel nous avions accès. Le dataset n'est pas normalisé avec des données vides, nulles ou non structurées et il y avait beaucoup d'incohérences comme des smileys dans le champs du prix de certains jeux. L'API posait aussi soucis, nous avions accès à peu de données utiles, il y avait des changements d'id des objets durant les requêtes, ainsi que des id identiques pour des jeux différents.

Les résultats que nous obtenons à la fin de notre recherche sont plutôt cohérent et les jeux retournés sont des jeux que l'on pourrait tout à fait acquérir de par la correspondance avec ce que l'on recherche.

Pour terminer, ce fut un projet fort intéressant, mais complexe à réaliser en si peu de temps.

9. Références

- Les algorithmes de recommandation [en ligne] disponible sur : https://www.podcastscience.fm/dossiers/2012/04/25/les-algorithmes-de-recommandation/ [Consulté le 11 juin 2020].
- Recommender Systems with Python — Part I : Content-Based Filtering [en ligne] disponible sur : https://heartbeat.fritz.ai/recommender-systems-with-python-part-i-content-based-filtering-5df4940bd831 [Consulté le 11 juin 2020].
- MongoEngine User Documentation [en ligne] disponible sur : https://docs.mongoengine.org [Consulté le 11 juin 2020].
- MongoDB Documentation [en ligne] disponible sur : https://docs.mongodb.com/ [Consulté le 11 juin 2020].
- Steam Web API [en ligne] disponible sur : https://developer.valvesoftware.com/wiki/Steam_Web_API [Consulté le 11 juin 2020].
- PyMongo 3.9.0 documentation [en ligne] disponible sur : https://api.mongodb.com/python/current/tutorial.html [Consulté le 11 juin 2020].
- WebAPI - Official TF2 Wiki [en ligne] disponible sur : https://wiki.teamfortress.com/wiki/WebAPI [Consulté le 11 juin 2020].
- Python multiprocessing documentation [en ligne] disponible sur : https://docs.python.org/fr/3.7/library/multiprocessing.html [Consulté le 11 juin 2020].
- Steam games complete dataset [en ligne] disponible sur : https://www.kaggle.com/trolukovich/steam-games-complete-dataset [Consulté le 11 juin 2020].