

# Compte-rendu du projet d'analyse de données: applications au jeu Stardle

Charles Azais de Vergeron et Octave Feuilland

Mai 2025

## 1 Sommaire

blallalalaal

## 2 Introduction

### 2.1 Présentation du jeu Stardle

Le jeu Stardle est un jeu en ligne ([stardle.net](http://stardle.net)) basé sur le jeu Wordle / SUTO. Le but est de deviner chaque jour un vaisseau choisi au hasard parmi une liste d'environ 200 éléments.

Chaque vaisseau est défini par deux types de caractéristiques:

- Qualitatives

- Fabricant (Manufacturer)
- Type
- Role
- Année d'apparition (Release Date)
- En Jeu ou Non (Status)

- Quantitatives

- Nombre de personnes dans l'équipage (Crew)
- Valeur en \$ (Price)
- Valeurs en monnaie du jeu (Price In game)
- Capacité en tant que cargo (Cargo Capacity)
- Vitesse de croisière (SCM) et maximale (Max)
- Longeur (Length), Largeur (Beam) et Hauteur (Height)

Voici un exemple de partie du jeu :

Ship	Manufacturer	Role(s)	Length	Value	Cargo	Crew	Release year
Cyclone AA	TUMBRIL	Combat	6m	\$80	0 SCU	2 †	2018
Cyclone	TUMBRIL	Expedition	6m	\$55	1 SCU	2 †	2018
ZX	ORIGIN	Touring	5.5m	\$40	0 SCU	1 †	2023
Hurricane		Heavy Fighter	22m	\$210	0 SCU	2 †	2018
13 Pisces		Pathfinder	16m	Not available for sale	4 SCU	3 †	2019
Freelancer	AMISC	Medium Freight	38m	\$110	66 SCU	4 †	2016

Figure 1: Exemple d'une partie du jeu Stardle

Nous avons 3 possibilités de résultats: Vrai, Faux ou Proche (uniquement pour les caractéristiques quantitatives).

## 2.2 Objectif du projet

L'objectif de ce projet est d'analyser les données du jeu Stardle afin de créer un algorithme capable de trouver un vaisseau pris au hasard dans la base de données en un minimum de tentatives et de déterminer une combinaison d'essais de vaisseaux qui permet de minimiser le nombre de tentatives.

Nous avons donc récupéré les données, recréé le jeu puis essayé de trouver un algorithme optimal pour déterminer le vaisseau le plus efficacement possible.

## 3 Récupération des données

La récupération des données est faite via le script Python `CreateShipDatabase.py`. Ce script utilise l'API officielle de Star Citizen Wiki pour récupérer les informations sur les vaisseaux.

### 3.1 Structure du script

Le script de collecte de données se décompose en plusieurs étapes principales :

#### 1. Collecte des données

Nous utilisation l'API <https://api.star-citizen.wiki/> pour obtenir la liste complète des vaisseaux. Pour chaque vaisseau, le script récupère les données via l'API et extrait toutes les caractéristiques – dans la configuration "All" – et uniquement celles du jeu Stardle – dans la configuration "Stardle" – cf le code de `CreateShipDatabase.py`

Dans la suite du rapport, "Stardle" désigne la configuration du jeu et "All" la configuration complète.

#### 2. Stockage des données

- Création de deux fichiers JSON :

- `shipList.json` : liste des vaisseaux et leurs liens
- `shipDB_All.json` : base de données complète des vaisseaux
- `shipDB_Stardle.json` : base de données avec les catégories du jeu Stardle

## 4 Nettoyage et préparation des données

Pour ajouter de la complexité, nous avons mélangés les deux base de données. Stardle est la base du projet. Nous avons extrait de "All" les valeurs de scm,max, length, beam. Nous avons aussi rajouté une colonne "price in game" qui correspond à la valeur du vaisseau dans le jeu (via le site Erkul.com)

Pour la gestion des valeurs manquantes, nous avons étudié les dépendances entre les colonnes. Nous avons remarqué que certaines colonnes étaient très corrélées entre elles. Par exemple, le prix \$ (sans valeurs manquantes) et celle

du prix en jeu (avec beaucoup des valeurs manquantes). Nous avons tenté de remplir les valeurs manquantes en utilisant une régression linéaire et une régression quadratique mais ce ne fut pas concluant. Nous avons donc appliqué l'algorithme suivant:

---

**Data:** CSV avec valeurs manquantes aux colonnes status,type,manufacturer  
**Résultat:** Colonne sans valeurs manquantes

```
groupby by status,type,manufacturer for vaisseaux in names do
    if mean(groupby by status,type,manufacturer) ≠ Nan and price_in_game == 0 then
        | price_in_game[vaisseaux] = mean(groupby by status,type,manufacturer)
    else
        | price_in_game[vaisseaux] = mean
    end
end
```

---

## 5 Algorithme de jeu

Dans un premier temps, nous avons pensé à une dichotomie en utilisant le script du jeu pour un utilisateur. Cependant, la théorie des graphes s'appliquait assez bien à notre problème. Nous avons créé un graphe dont les vaisseaux sont les noeuds et la distance entre eux :

Si la marque ou le nombre d'équipage ou le type ou le rôle ou la date de sortie ou le statut sont différents, la distance augmente de 1 (choix arbitraire). Puis nous prenons la racine carrée. C'est donc comparable à une distance euclidienne binaire: cf `detailed_paths_history.csv` pour avoir toutes les distances.

Cela nous donne le graphe suivant pour les types des vaisseaux :

??

Nous considérons l'équipage et l'année de sortie comme des données qualitatives car elles sont discrètes et non continues comme peuvent l'être la masse ou le prix par exemple.

### 5.1 Algorithme Stardle Auto

L'algorithme `stardle_auto()` est une version automatisée du jeu Stardle qui utilise une approche basée sur les distances entre vaisseaux pour optimiser la recherche. Voici son fonctionnement :

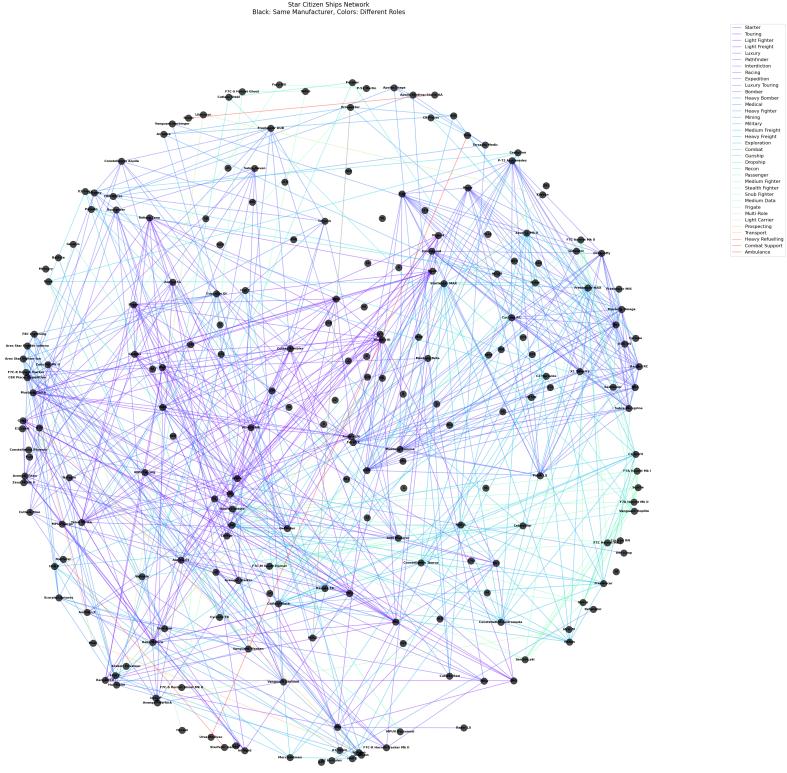


Figure 2: Exemple d'une partie du jeu Stardle

---

**Data:** Base de données des vaisseaux, Matrice des distances pré-calculées  
**Résultat:** Nombre de tentatives et liste des vaisseaux essayés  
 verifier\_ship retourne les informations comparatives entre deux vaisseaux.  
 get\_ship\_distance retourne la distance entre deux vaisseaux depuis la matrice des distances.  
 $\text{ship\_secret} \leftarrow$  Choisir un vaisseau aléatoire dans la base\_de\_données  
 $\text{tentatives\_max} \leftarrow 6$   
 $\text{visited\_ships} \leftarrow$  liste vide  
 $\text{current\_ship} \leftarrow$  Choisir un vaisseau aléatoire initial  
**for** *essais de 1 à tentatives\_max* **do**  
 | Ajouter current\_ship à visited\_ships  
 | resultat  $\leftarrow$  verifier\_ship(ship\_secret, current\_ship)  
 | **if** resultat == "Bravo" **then**  
 | | **return** (attempt, visited\_ships)  
 | **else**  
 | | closest\_ships  $\leftarrow$  []  
 | | **for** ship dans base\_de\_données **do**  
 | | | **if** ship  $\notin$  visited\_ships **then**  
 | | | | distance  $\leftarrow$  get\_ship\_distance(ship, ship\_secret)  
 | | | | **if** distance  $\neq$  None **then**  
 | | | | | Ajouter (ship, distance) à closest\_ships  
 | | | **end**  
 | | **end**  
 | **end**  
 | **if** closest\_ships non vide **then**  
 | | Trier closest\_ships par distance croissante current\_ship  $\leftarrow$  closest\_ships[0]

## 6 Conclusion

Nous avons construit un dataset pour jouer au jeu Stardle et créer un algorithme joueur. Nous voulions déterminer un choix optimal de vaisseau de départ. Cependant, l'algorithme détermine le bon vaisseaux en deux essais. De plus, dans certains cas de vaisseaux proches trop égaux dans leur données, l'algorithme ne les trouve pas (5 cas sur 5000 tentatives ).