



# INFO0607 - Projet

Recommandation de Jeux-Vidéos

Benjamin VILLA, Ophélie KAMINSKI - L3 Informatique
Enseignant : Maxime GUERY
Université de Reims Champagne-Ardenne - UFR SEN
Année 2020/2021

# Table des matières

# Données

Pour ce projet, nous avons décidé d'utiliser un jeu de données provenant de kaggle.com[1] qui rassemble l'ensemble des ventes de jeux vidéos dans le monde jusqu'au 22 décembre 2016. Ces données ont été recueillies auprès de VGChartz qui est un site web publiant régulièrement l'état de cette industrie au travers entre autres de ses ventes. On y a de plus adjoint des informations quant à la réception critique de chaque jeu ainsi que son évaluation ESRB lorsqu'elles étaient disponibles.

| Name | Platform | Year_of_Release | Genre | Publisher | NA_Sales | EU_Sales | JP_Sales | Other_Sales | Global_Sales | Critic_Score | Critic_Count | User_Score | User_Count | Developer | Rating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Wii Sports | Wii | 2006 | Sports | Nintendo | 41.36 | 28.96 | 3.77 | 8.45 | 82.53 | 76 | 51 | 8 | 322 | Nintendo | E |
| Super Mario Bros. | NES | 1985 | Platform | Nintendo | 29.08 | 3.58 | 6.81 | 0.77 | 40.24 | | | | | | |
| Mario Kart Wii | Wii | 2008 | Racing | Nintendo | 15.68 | 12.76 | 3.79 | 3.29 | 35.52 | 82 | 73 | 8.3 | 709 | Nintendo | E |
| Wii Sports Resort | Wii | 2009 | Sports | Nintendo | 15.61 | 10.93 | 3.28 | 2.95 | 32.77 | 80 | 73 | 8 | 192 | Nintendo | E |
| Pokemon Red/Pokemon Blue | GB | 1996 | Role-Playing | Nintendo | 11.27 | 8.89 | 10.22 | 1 | 31.37 | | | | | | |
| Tetris | GB | 1989 | Puzzle | Nintendo | 23.2 | 2.26 | 4.22 | 0.58 | 30.26 | | | | | | |
| New Super Mario Bros. | DS | 2006 | Platform | Nintendo | 11.28 | 9.14 | 6.5 | 2.88 | 29.8 | 89 | 65 | 8.5 | 431 | Nintendo | E |
| Wii Play | Wii | 2006 | Misc | Nintendo | 13.96 | 9.18 | 2.93 | 2.84 | 28.92 | 58 | 41 | 6.6 | 129 | Nintendo | E |
| New Super Mario Bros. Wii | Wii | 2009 | Platform | Nintendo | 14.44 | 6.94 | 4.7 | 2.24 | 28.32 | 87 | 80 | 8.4 | 594 | Nintendo | E |
| Duck Hunt | NES | 1984 | Shooter | Nintendo | 26.93 | 0.63 | 0.28 | 0.47 | 28.31 | | | | | | |

Comme visible ci-dessus pour les 10 premières entrées, les données sont représentées par un CSV avec entête comportant au total 11563 jeux.

Concernant l'entête, nous pouvons retrouver 16 propriétés par jeux :

- Name : représentant le nom du jeu en question
- Platform : définissant la console sur laquelle le jeu est sorti
- Year_of_Release : indiquant l'année de sortie
- Genre : précisant de quel genre de jeu il est question
- Publisher : définissant l'éditeur du jeu
- NA_Sales, EU_Sales, JP_Sales, Other_Sales, Global_Sales : représentant les ventes du jeu (en millions d'exemplaires) respectivement aux USA, en Europe, au Japon, dans le reste du monde et les ventes totales
- Critic_Score : décrivant la réception critique du jeu d'après une note sur 100 délivrée par le site Metacritic et combinant les différentes reviews de journalistes
- Critic_Count : rapportant le nombre de reviews de journalistes
- User_Score : décrivant la réception du jeu par les joueurs d'après une note sur 10 combinant les différents avis d'utilisateurs
- User_Count : rapportant le nombre d'avis de joueurs
- Developer : définissant le développeur du jeu
- Rating : prévenant du label d'évaluation donnée par l'ESRB pour le jeu

Par la suite, nous allons devoir sélectionner seulement une partie de ce jeu de données afin de répondre au mieux à la problématique qui va suivre.

[1] Lien du jeu de données : https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings

# Problématique

En accord avec notre jeu de données présenté dans la section précédente, nous avons jugé qu'il serait pertinent de modéliser un système de recommandation afin de permettre à quiconque ayant déjà joué à un jeu de se voir proposer d'autres jeux qui pourront tout autant lui convenir.

Dès lors, il est important de réussir à définir si il est vraiment possible de créer un tel système et de s'assurer que celui-ci vienne à réellement fonctionner sur nos données.

Comment est-il possible de créer des listes de recommandations de jeux d'après les goûts de l'utilisateur ?

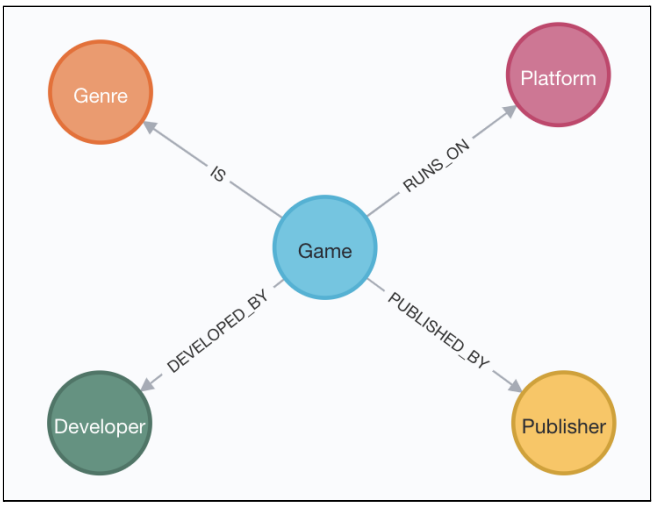
Voyons alors une tentative de réponse au travers de l'usage du Système de Gestion de Base de Données Neo4j combinée à son plugin Graph Data Science qui vont nous permettre de rechercher des similarités dans ces données.

# Modélisation

Initialement, le jeu de données contient une multitude de propriétés qui ne sont pas toutes autant appropriées dans l'idée de chercher des similarités entre les jeux, nous devons alors les réduire au strict nécessaire. De plus, toutes les entrées ne sont pas forcément remplies donc on préférera compter sur celles complètes.

## Le modèle

Le choix opéré sur le nettoyage des données nous amène à ce modèle :




5 labels de noeuds :

- Game : un jeu
- Genre : un genre de jeu
- Platform : une console de jeux
- Developer : un développeur de jeux
- Publisher : un éditeur de jeux

4 types de relations :

- Game → Genre : un jeu fait partie d'un genre
- Game → Platform : un jeu est exécuté sur une console de jeux
- Game → Developer : un jeu a été développé par un développeur de jeux
- Game → Publisher : un jeu a été publié par un éditeur de jeux

Dans l'idée de recommander des jeux similaires, la première piste nous amène à chercher des jeux du même genre. Ensuite, nous avons remarqué que lorsqu'un jeu pouvait plaire alors fréquemment les jeux du même développeur voire éditeur pouvaient amener à une recommandation logique. De même, on souhaitera davantage proposer des jeux sur une console que l'utilisateur possède déjà, impliquant alors que si un jeu sort sur plusieurs plateformes, on devra dissocier chaque version.

La requête Cypher exécutée pour obtenir ce modèle est :

```
CALL db.schema.visualization()
```

## Création de la base de données

La base de données est tout d'abord vidée de ses nœuds pour être sûr qu'aucune trace de données résiduelles pourrait troubler nos résultats :

```
MATCH (n)
DETACH DELETE n;
```

Par la suite, les données sont chargées depuis le CSV :

```
LOAD CSV
WITH HEADERS FROM 'file:///dataset.csv' AS line
```

Les jeux ne sont pris en compte que si les données dont nous avons besoin (Developer, Publisher, Platform et Genre) sont renseignées :

```
WITH line
WHERE line.Developer IS NOT NULL
AND line.Publisher IS NOT NULL
AND line.Platform IS NOT NULL
AND line.Genre IS NOT NULL
```

Puis, différents nœuds sont créés, chacun unique et identifiés : les jeux, les développeurs, les éditeurs, les plateformes et les genres. Ceux-ci possèdent également des attributs : leur année de sortie (year), leur score (score) et leurs ventes au niveau mondial (sales) :

```
CREATE (game:Game {
  name: line.Name,
  year: toInteger(line.Year_of_Release),
  sales: toFloat(line.Global_Sales),
  score: toInteger(line.Critic_Score)
  })
MERGE (dev:Developer {name: line.Developer})
MERGE (pub:Publisher {name: line.Publisher})
MERGE (plat:Platform {name: line.Platform})
MERGE (genre:Genre {name: line.Genre})
```

Enfin, les relations sont établies entre chaque jeu :

```
CREATE (game)-[:DEVELOPED_BY]->(dev)
CREATE (game)-[:PUBLISHED_BY]->(pub)
CREATE (game)-[:RUNS_ON]->(plat)
CREATE (game)-[:IS]->(genre);
```

# Statistiques du graphe

## Diamètre

```
MATCH (n),(m)
WHERE n <> m
WITH n,m
MATCH path = shortestPath((a)-[*]-(b))
RETURN length(path) AS diameter
ORDER BY diameter DESC
LIMIT 1
```

Le diamètre du graphe représente l'excentricité maximale du graphe que l'on peut reformuler comme étant le plus long des plus courts chemins entre nœuds du graphe. Ainsi, il nous suffit de calculer les plus courts chemins pour chaque couple possible de nœuds, récupérer ensuite leurs longueurs respectives et enfin les classer du plus grand au plus petit. On affichera uniquement la première entrée.

## Densité

```
MATCH (v)
WITH toFloat(count(v)) AS V
MATCH ()-[e]->()
WITH V, toFloat(count(e)) AS E
RETURN E / (V * (V - 1)) AS density
```

La densité du graphe détermine à quel point celui-ci s'approche de sa forme complète où tout couple de sommets disjoints est relié par une arête et dont la densité vaut 1. Le calcul pour un graphe dirigé comme le nôtre est obtenu d'après la formule :

$$D = \frac{|E|}{2\binom{|V|}{2}} = \frac{|E|}{|V|(|V|-1)}$$

avec `E` l'ensemble des arêtes et `V` l'ensemble des sommets.

## Degrés

```
MATCH (n)
RETURN
    n.name,
    size(()-[]->(n)) AS indegree,
    size((n)-[]->()) AS outdegree,
    size((n)-[]-()) AS total
```

Chaque nœud possède un certain degré qui représente le nombre d'arêtes entrantes (`indegree`) et sortantes (`outdegree`) que l'on obtient simplement en faisant le compte de chacun des deux types d'arêtes via l'appel à la fonction `size()`. Dans le cas présent on admet la présence d'une propriété `name` afin d'identifier à quel nœud correspond le résultat.

## Nombre de noeuds par label

```
MATCH (n)
RETURN DISTINCT labels(n) AS label, count(labels(n)) AS amount
```

## Nombre de relations par type

```
MATCH ()-[r]->()
RETURN DISTINCT type(r) AS type, count(type(r)) AS amount
```

# Traitements

## L'algorithme Node Similarity

### Historique

L'algorithme Node Similarity est un algorithme, basé sur le coefficient de Jaccard, qui détermine la probabilité de similarité entre plusieurs éléments. Ce coefficient, nommé d'après le biologiste suisse Paul Jaccard, permet "d'évaluer la similarité entre deux ensembles"[2].

### Choix

Nous avons choisi cet algorithme car nous souhaitons effectuer une recommandation; il faut donc déterminer quels jeux sont les plus similaires entre eux. Ainsi, en partant d'un jeu qu'un joueur apprécie, il sera possible de sélectionner ceux qui lui ressemblent le plus, et donc qui seront plus susceptibles de plaire au joueur.

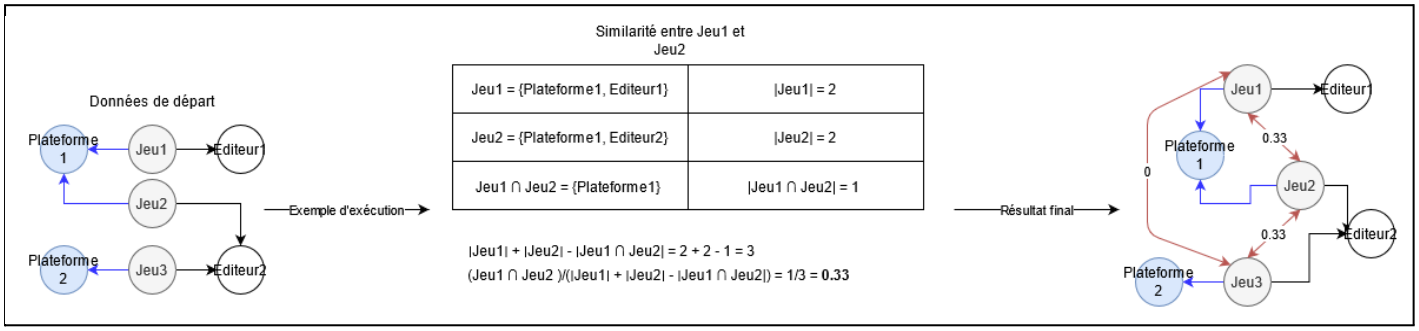
### Fonctionnement

L'algorithme Node Similarity s'appuie donc sur le coefficient de Jaccard, qui est donné par la formule ci-après :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Plus A et B possèdent de propriétés communes, plus le coefficient de Jaccard sera élevé. Dans le cadre de notre jeu de données, les propriétés sont les relations en commun entre les jeux (même éditeur, même développeur, même plateforme, même genre).

Le fonctionnement de l'algorithme peut être résumé par le schéma suivant :




[2] https://fr.wikipedia.org/wiki/Indice_et_distance_de_Jaccard

### Implémentation

Voici comment nous appelons l'algorithme de Node Similarity dans notre script.

```
:param config => ({
  similarityCutoff: 0.1,
  degreeCutoff: 1,
  nodeProjection: '*',
  relationshipProjection: {
    relType: {
      type: '*',
      orientation: 'NATURAL',
      properties: {}
    }
  },
  writeProperty: 'value',
  writeRelationshipType: 'SIMILARITY'
});

CALL gds.nodeSimilarity.write($config);
```

## L'algorithme Modularity Optimization

### Historique

L'algorithme Modularity Optimization est un algorithme de détection de communauté encore au stade de bêta. Il est basé sur l'estimation de la modularité des communautés dans un graphe.[3]

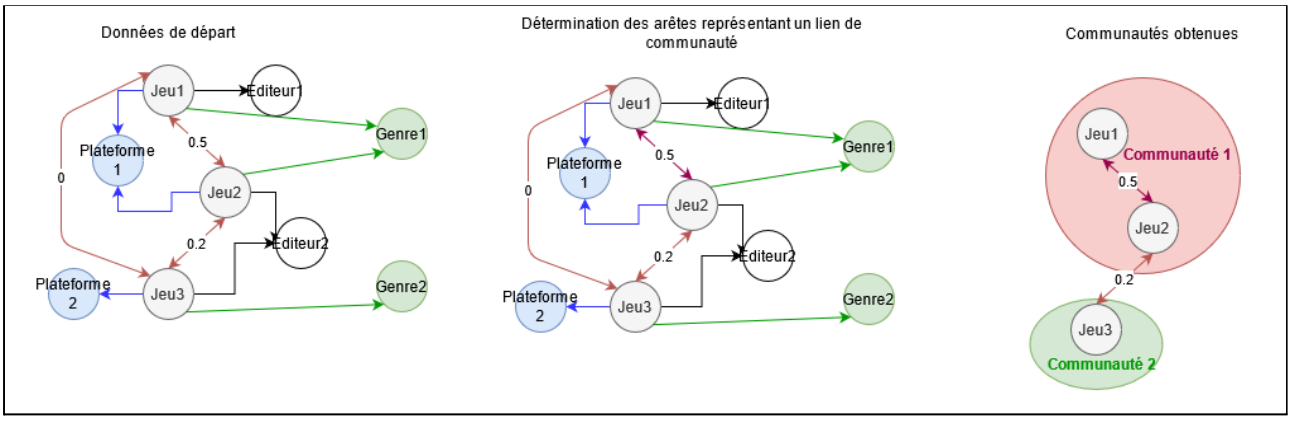
### Choix

Après avoir appliqué à nos données l'algorithme Node Similarity, nous voulions pouvoir regrouper les nœuds ayant de fortes similarités. Après avoir testé plusieurs algorithmes de détection de communautés, l'algorithme Modularity Optimization est celui dont les résultats nous ont semblé les plus cohérents et les plus exploitables.

[3] https://neo4j.com/docs/graph-data-science/current/algorithms/modularity-optimization/

## Fonctionnement

L'algorithme de Modularity Optimization peut être schématisé de la façon suivante :




## Implémentation

L'algorithme est appelé ainsi dans notre script :

```
:param config => ({
  nodeProjection: 'Game',
  relationshipProjection: {
    relType: {
      type: 'SIMILARITY',
      orientation: 'NATURAL',
      properties: {
        value: {
          property: 'value',
          defaultValue: 1
        }
      }
    }
  },
  relationshipWeightProperty: 'value',
  seedProperty: '',
  maxIterations: 10,
  tolerance: 0.0001,
  writeProperty: 'community'
});
CALL gds.beta.modularityOptimization.write($config);
```

# Analyse des résultats

Maintenant que le traitement des données a été opéré, nous pouvons afficher les résultats afin de voir ce qui a été possible d'exploiter à partir de nos données.

N'ayant pas trouvé comment exécuter le script principal tout en permettant un affichage dynamique, les requêtes à venir seront à exécuter à la main dans le Browser de Neo4j.

Ainsi, avec le calcul du degré de similarité effectué par Node Similarity nous pouvons afficher pour chaque jeu X, son jeu Y qui lui est le plus recommandé :

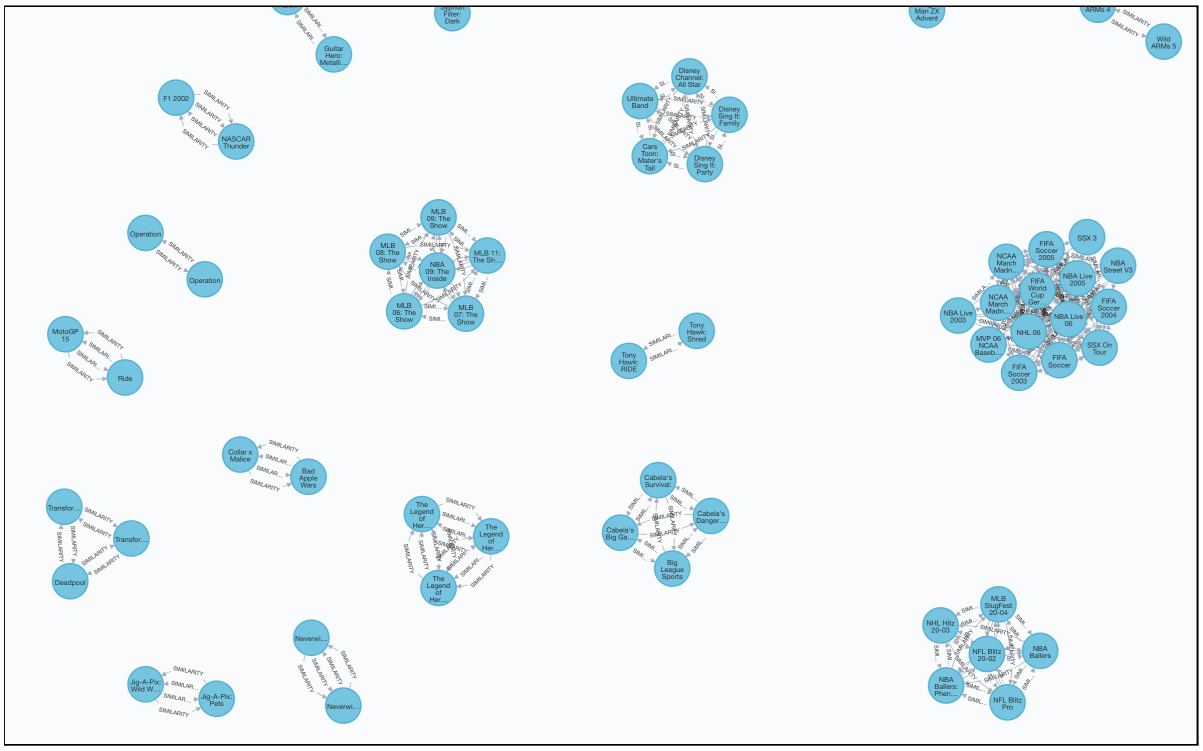
```
MATCH (from)-[rel:SIMILARITY]-(to)
WHERE exists(rel.value)
RETURN
    from.name AS `Jeu de base`,
    to.name AS `Jeu le plus recommandé`,
    rel.value AS similarity
ORDER BY similarity DESC
LIMIT 50;
```

Seuls les 50 premiers résultats sont affichés ici afin d'obtenir un affichage rapide :

| | Jeu de base | Jeu le plus recommandé | similarity |
|---|---|---|---|
| 31 | "Shin Megami Tensei: Strange Journey" | "Shin Megami Tensei: Devil Survivor" | 1.0 |
| 32 | "Army of Two: The Devil's Cartel" | "Battlefield: Hardline" | 1.0 |
| 33 | "Patapon 3" | "Patapon" | 1.0 |
| 34 | "MLB 07: The Show" | "MLB 11: The Show" | 1.0 |
| 35 | "Tak and the Guardians of Gross" | "Bratz: Girlz Really Rock" | 1.0 |
| 36 | "Guitar Hero: Van Halen" | "Guitar Hero: Metallica" | 1.0 |

On remarque alors que les jeux recommandés sont pour la majorité très pertinents car ils reprennent bien les différents paramètres que l'on avait choisi au départ soit le genre, la plateforme, le développeur et l'éditeur. Cependant, il demeure un certain nombre de faux positifs qui restent cela dit négligeables.

En affichant les relations des noeuds, on constatera qu'ils forment des groupes que l'on va rassembler ensuite avec Modularity Optimization :



Une fois l'algorithme exécuté, nous obtenons un affichage avec la requête :

```
MATCH (node:Game)
WHERE exists(node.community)
WITH node.community AS community, collect(node.name) AS list
RETURN DISTINCT
      community AS `Communauté`,
      size(list) AS `Taille`,
      list AS `Liste de recommandations`
ORDER BY community ASC
LIMIT 50;
```

Seuls les 50 premiers résultats sont affichés ici afin d'obtenir un affichage rapide :

| | | | |
|---|---|---|---|
| 94 | 1035 | 40 | ["The Sims 2: Pets", "The Sims 2", "The Sims 4: Get Together", "Top Gun: Combat Zones", "The Sims: Unleashed", "The Sims: Vacation", "The Sims 4", "The Sims: Livin Large", "The Sims", "The Sims: Bustin' Out", "The Sims: House Party", "The Sims: Makin' Magic", "The Sims: Hot Date", "The Urbz: Sims in the City", "The Sims: Superstar", "SimCity (2013)", "The Sims 2", "The Sims 2: Nightlife", "The Sims 2: Castaway", "Spore", "The Sims 2", "The Sims 2: Pets", "The Sims 2: Castaway", "The Sims: Bustin' Out", "The Sims: Bustin' Out", "The Sims 2: Open for Business", "Euro Truck Simulator", "The Sims", "The Sims 2: Pets", "The Sims 3: Ambitions", "The Sims", "The Urbz: Sims in the City", "Spore Hero", "The Sims 2", "The Sims 2: Castaway", "The Urbz: Sims in the City", "The Sims 2", "The Sims 2: Pets", "The Sims 2", "SimCity 4"] |
| 95 | 1053 | 26 | ["GTR Evolution", "GT Legends", "Test Drive Unlimited", "Ridge Racer Unbounded", "Valentino Rossi: The Game", "WRC: FIA World Rally Championship", "Grand Prix Legends", "Sega Rally Revo", "Ridge Racer 7", "Project CARS", "Test Drive Unlimited 2", "Test Drive Unlimited 2", "Test Drive Unlimited", "Test Drive Unlimited", "Test Drive Unlimited 2", "Ridge Racer", "Race Pro", "Ridge Racer Unbounded", "Project CARS", "18 Wheels of Steel: Extreme Trucker 2", "Ridge Racer Unbounded", "Valentino Rossi: The Game", "Assetto Corsa", "Split/Second", "Off-Road Drive", "RACE On"] |
| 96 | 1072 | 16 | ["Star Trek: Tactical Assault", "The Settlers: Rise of an Empire", "Tom Clancy's EndWar", "Anno 2070", "Tom Clancy's EndWar", "Tom Clancy's EndWar", "R.U.S.E.", "The Settlers 7: Paths to a Kingdom", "R.U.S.E.", "R.U.S.E.", "Anno 2205", "Age of Empires: Collector's Edition", "Tom Clancy's EndWar", "Worms World Party", "Heroes of Might and Magic V", "World in Conflict: Complete Edition"] |

Et enfin voici sa version permettant d'afficher la communauté pour un jeu donné, dont le titre devra être renseigné dans le script :

```
MATCH (src:Game),(dst:Game)
WHERE exists(src.community)
AND exists(dst.community)
AND src.name = 'Bayonetta'
AND dst.community = src.community
WITH src.name AS `Jeu de base`, src.community AS community,
collect(dst.name) AS list
RETURN DISTINCT
      `Jeu de base`,
      community AS `Communauté`,
      size(list) AS `Taille`,
      list AS `Liste de recommandations`
```

Il suffira de changer la 4ème ligne du script en spécifiant entre guillemets le nom du jeu dont l'on souhaite une liste de recommandations. Voici donc son résultat avec le jeu Bayonetta :

| Jeu de base | Communauté | Taille | Liste de recommandations |
|---|---|---|---|
| "Bayonetta" | 18307 | 41 | ["El Shaddai: Ascension of the Metatron", "Iron Man 2", "The Incredible Hulk", "Deadly Premonition", "Afro Samurai", "Way of the Samurai 3", "Binary Domain", "Captain America: Super Soldier", "Country Dance: All Stars", "Anarchy Reigns", "Fairytale Fights", "El Shaddai: Ascension of the Metatron", "Jurassic Park: The Game", "Fist of the North Star: Ken's Rage", "The Cursed Crusade", "Thor: God of Thunder", "X-Blades", "Eragon", "Kung-Fu: High Impact", "Onechanbara: Bikini Samurai Squad", "Sonic Free Riders", "Bayonetta", "Sleeping Dogs", "Turok", "Terraria", "Iron Man", "Condemned 2: Bloodshot", "Ghostbusters: The Video Game", "Iron Man", "Super Monkey Ball 3D", "Rise of Nightmares", "The Golden Compass", "Sega Rally Revo", "Viking: Battle for Asgard", "Condemned: Criminal Origins", "Naughty Bear", "Iron Man 2", "The Golden Compass", "Golden Axe: Beast Rider", "Overlord II", "Sonic Boom: Shattered Crystal"] |
| "Bayonetta" | 296 | 17 | ["The Incredible Hulk", "Anarchy Reigns", "Captain America: Super Soldier", "Dengeki Bunko Fighting Climax", "Thor: God of Thunder", "Bayonetta", "Yakuza 3", "Virtua Fighter 5", "Yakuza 4", "Iron Man", "Yakuza 5", "Iron Man 2", "Condemned 2: Bloodshot", "Viking: Battle for Asgard", "Binary Domain", "Golden Axe: Beast Rider", "The Golden Compass"] |
| "Bayonetta" | 8926 | 16 | ["Bayonetta", "Yakuza 3", "Yakuza 4", "Iron Man", "Yakuza 5", "Iron Man 2", "Condemned 2: Bloodshot", "Viking: Battle for Asgard", "Binary Domain", "Golden Axe: Beast Rider", "The Golden Compass", "The Incredible Hulk", "Anarchy Reigns", "Captain America: Super Soldier", "Thor: God of Thunder", "Planet 51"] |
| "Bayonetta" | 9288 | 16 | ["Bayonetta", "Condemned 2: Bloodshot", "Iron Man", "Rise of Nightmares", "The Golden Compass", "Viking: Battle for Asgard", "Condemned: Criminal Origins", "Golden Axe: Beast Rider", "Iron Man 2", "The Incredible Hulk", "Binary Domain", "Captain America: Super Soldier", "Anarchy Reigns", "Thor: God of Thunder", "Planet 51", "Universe at War: Earth Assault"] |

Un autre exemple avec Halo 4 :

| Jeu de base | Communauté | Taille | Liste de recommandations |
|---|---|---|---|
| "Halo 4" | 6917 | 29 | ["Worms", "Gears of War", "Banjo-Kazooie", "Halo 3", "Halo: Reach", "Halo 4", "Gears of War 2", "Halo 3: ODST", "Gears of War 3", "Kinect Sports", "Gears of War", "Halo 5: Guardians", "Gears of War: Ultimate Edition", "Halo: The Master Chief Collection", "Halo: Combat Evolved Anniversary", "Crackdown", "Viva Pinata", "Kinectimals", "Gears of War: Judgment", "Gears of War 4", "Sunset Overdrive", "Crackdown 2", "Perfect Dark Zero", "Kinect Rush: A Disney Pixar Adventure", "Unreal Tournament III", "Viva Pinata: Trouble in Paradise", "The Gunstringer", "Project Sylpheed: Arc of Deception", "Homefront: The Revolution"] |
| "Halo 4" | 14637 | 15 | ["Project Sylpheed: Arc of Deception", "Halo 3", "Halo: Reach", "Halo 4", "Gears of War 2", "Halo 3: ODST", "Gears of War 3", "Gears of War", "Halo: Combat Evolved Anniversary", "Crackdown", "Gears of War: Judgment", "Crackdown 2", "Perfect Dark Zero", "Unreal Tournament III", "The Gunstringer"] |

Pour conclure, nous avons donc réussi à développer un système de recommandations basé sur l'appréciation initiale d'un jeu vidéo et en se référant aux différentes données disponibles. Les résultats restent améliorables mais sont globalement satisfaisants, c'est au niveau du choix de modélisation et de la prise en compte d'autres facteurs comme les ventes ou le score critique d'un jeu que l'on pourrait afficher des listes encore plus précises via des tris basés sur ces derniers.