Connaissances et raisonnements : Projet Snow man

Étudiant :
NIANG Abdoul Aziz

Année : 2018-2019

7 janvier 2019

Table des matières

Introduction

Le but de ce projet est de réaliser une application permettant de jouer au jeu du "bonhomme de neige". Pour ce faire, il fallait constituer une ontologie qui représente les différents données du jeu en question. Puis développer un outil qui va interagir avec ces données pour pouvoir jouer à ce jeu.

Dans le corps de ce rapport, on va parler des différents outils utilisés pour la réalisation de l'application en premier lieu. En deuxième lieu, aborder les différentes réalisations effectuées. Et en dernier lieu, faire un bilan sur l'application.

I Outils utilisés

I.1 Les ontologies

Une ontologie constitue un modèle de données représentatif d'un ensemble de concepts dans un domaine et des relations entre ces concepts. Elle vise principalement à modéliser un ensemble de connaissances dans un domaine donné. On retrouve cette technologie dans beaucoup de données comme l'intelligence artificielle, le web sémantique, l'architecture de l'information, etc.

I.2 Stardog

Stardog est un triple-store conçu spécialement pour stocker et récupérer des données RDF. Toutes les données de ce type de base de données sont stockées sous forme de triplet qui est une association (sujet, prédicat, objet). De ce fait, il n'est pas nécessaire de créer des tables pour stocker les données contrairement aux bases de données relationnelles. De plus, on peut stocker un grand nombre de triplets.

I.3 Graphql

Graphql est un langage de requêtes développé par *Facebook* en 2012 et publié en 2015. Il permet d'interroger une base de données sans nécessairement connaître la structure des données de la base. La réponse obtenue suite à une requête est au format *JSON*. Ce langage est implémenté pour de nombre langages tels que Python, JavaScript, etc.

I.4 NodeJS

Node.js est une plateforme logicielle libre et événementielle en JavaScript. Il a été créé en 2009 par *Ryan Dahl*. Il permet d'exécuter du JavaScript coté serveur pour effectuer de opérations. Il permet donc de se passer de serveurs web tels que Nginx ou Apache lors du déploiement de sites et d'applications web développés avec Node.js.

I.5 ReactJS

ReactJS est une bibliothèque JavaScript développé en 2013 par Facebook. Elle est considérée comme la vue dans le modèle *MVC*. Il vise principalement à faciliter la création d'application web monopage en créant des

composants qui dépendent de l'état général de la page. Elle est utilisé bon nombre d'applications connues comme *Netflix*, *Yahoo*, etc.

I.6 Quelques autres langages

Quelques autres langages ont aussi été utilisés pour réaliser l'application. Ce sont les langages HTML, CSS et JavaScript.

II Réalisations

II.1 Mise en place des données

Pour mettre en place les données de l'application, il a fallu utilisé l'outil *Protégé* pour créer une ontologie avec les concepts et les propriétés ci-dessous.

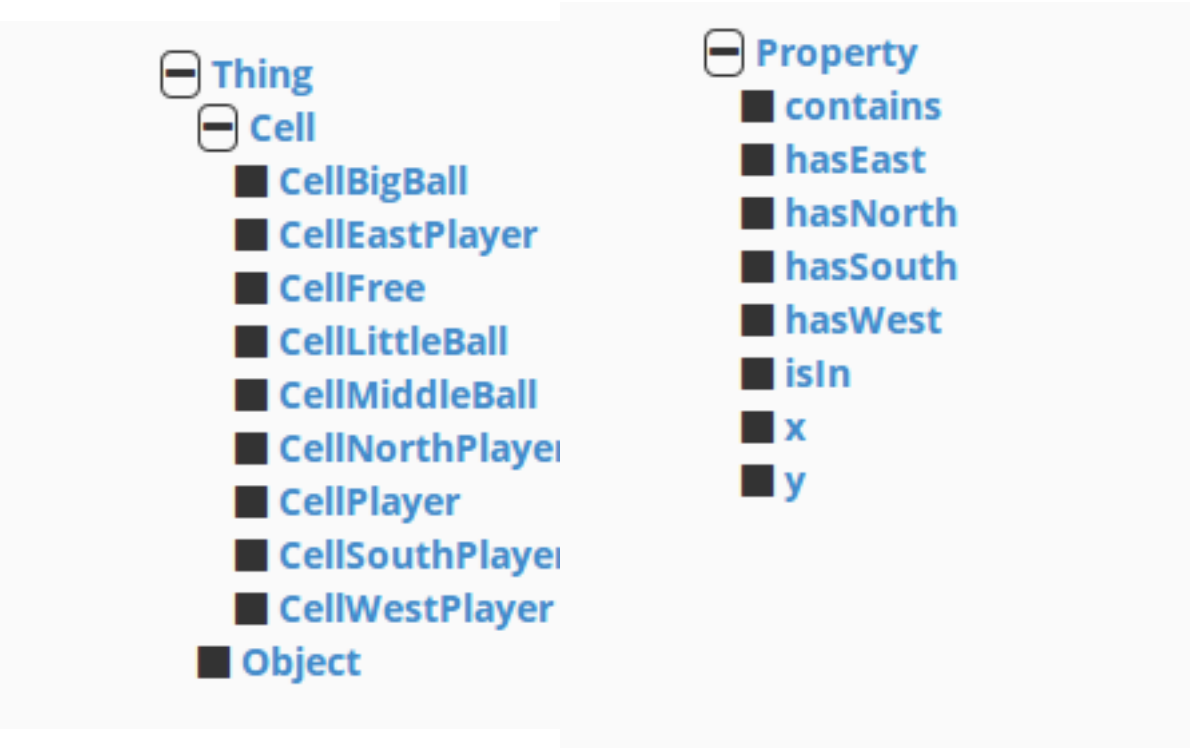


FIGURE 1 – Concepts et propriétés de l'ontologie

Puis, la classe **Object** est doté des individus :
— *bigBall* qui représente la grande boule de neige
— *middleBall* qui représente la boule moyenne
— *littleBall* qui représente la petite boule de neige

Ensuite, afin de créer des instances de la classe **Cell**, il a fallut effectuer un certains nombre d'opérations.

— D'abord, on utilise le logiciel *LibreOffice Calc* pour constituer une grille (10*10) qui exporter au format *csv*.

— Ensuite, on fait appel au script fournit dans l'énoncé du devoir pour transformer les données *csv* en données *turtle*. On a, ci-dessous, un exemple de ce que l'on peut obtenir suite à cette opération.

```
:cell00 rdf:type owl:NamedIndividual ,
                 :Cell ;
        :hasEast :cell10 ;
        :hasNorth :cell01 ;
        :hasSouth :wall ;
        :hasWest :wall ;
        :x 0 ;
        :y 0 .
```

— Enfin, les lignes du fichier obtenu (fichier au format *.ttl*) seront ajoutées à l'ontologie établie précédemment avec *Protégé*.

Une fois que toutes ces étapes sont effectuées, on établit les différentes relations entre concepts et concepts,propriétés et propriétés et entre propriétés et concepts.

— la propriété *hasEast* est l'inverse de la propriété *hasWest*.

— la propriété *hasNorth* est l'inverse de la propriété *hasSouth*.

— *isIn* inverse de *contains*.

— *contains* a pour domaine *Cell* et pour range *Object*.

— *hasNorth* a *Cell* comme domaine et range.

— la classe *CellPlayer* équivaut à être une cellule et contenir le bonhomme.

— CellFree est équivalent à être une cellule sans objet.

— etc.

Afin de pouvoir accéder et manipuler ces données depuis l'application, on les stocke dans la base de données *Stardog*. Pour ce faire, il a fallu télécharger *Stardog* et configurer le système pour pouvoir y utilise cette technologie. Puis ajouter les données dans le triple-store avec l'aide des instructions suivantes.

1. Se placer dans le dossier *Stardog*
2. `$ ./bin/stardog-admin server start --web-console`

   `$ ./bin/stardog-admin db create -n SnowMan PATH/SnowMan.owl`

Après la base sera créée et on pourra la visualiser à partir du endpoint de *Stardog* (*http ://localhost :5820*).

III Mise en place du serveur Graphql

Cette étape consiste à mettre sur pieds un serveur *nodeJS*, qui grâce à *Graphql*, va interroger la base de données de *Stardog* pour fournir des réponses à des requêtes données. En effet, il a été nécessaire d'établir la connexion avec le triple-store en créant une instance de la classe connexion de la manière suivante.

```
const conn = new Connection({
  username: 'admin',
  password: 'admin',
  endpoint: 'http://localhost:5820',
});
```

Ensuite on accède à la base pour soit récupérer des données, soit les modifier. Pour effectuer ces deux actions, il a fallu mettre en place un certain nombre de résolveurs.

III.1 Récupération de données (Query)

Lors de la récupération de données dans le triple-store, le serveur s'intéresse à cinq éléments.

~~3.~~ *cells*, l'ensemble de toutes les cellules.
— *cellPlayer*, la cellule qui contiens le bonhomme de neige.
— *cellLittleBall*, la cellule de la petite boule de neige.
— *cellMiddleBall*, la cellule de boule de neige moyenne.
— *cellBigBall*, la cellule de la grande boule de neige.

Chaque cellule est représentée par son *URI*. Par exemple, l'*URI* ci-dessous représente la cellule de coordonnées *x=0* et *y=0*.

```
http://www.semanticweb.org/21416864/ontologies/2018/9/devoir#cell00
```

Ainsi, une fois lancé, le serveur va exécuter les différentes requêtes nécessaires à l'obtention des valeurs des éléments cités ci-dessus. Puis ces valeurs seront retournées pour être plus tard utilisés lors de la visualisation des données. Le code ci-dessous est un exemple de requête que fait le serveur.

```
query.execute(connexion,'SnowMan','select␣?cell␣where␣{?cell␣a␣:Cell␣}
  }).then(({ body }) => {
    cells = rewriteList(body.results.bindings);
  });
```

Cette requête nous permet de récupérer toutes les cellules de la grille. On observe qu'elle a besoin d'une connexion qui lie le serveur au triple-store, du nom de la base de données à laquelle on s'intéresse, d'une requête *SPARQL*

qui spécifie le type de données qu'on souhaite obtenir et du format de sortie de la requête.
Puis dans la liste de résolveur, on renvoie le résultat obtenu par la requête.

```
cells(root, args, context){
    return cells;
}
```

III.2 Modification de données (Mutation)

Cette opération permet d'effectuer des modifications dans la base données lors que l'on veut, par exemple, modifier la position du bonhomme de neige ou la position d'une boule. On utilise ainsi la *mutation*. Le fichier *resolvers.js* comporte quatre type de mutations.

- *movePlayerNorth*, déplace le bonhomme de neige vers le nord (vers le haut).
- movePlayerSouth, dirige le joueur vers le sud (vers le bas).
- movePlayerWest, déplace le joueur vers la gauche.
- movePlayerEast, déplace le joueur vers la droite.

À chacune des mutations ci-dessus, on associe une requête de la forme suivante :

```
movePlayerNorth(root){
    query.execute(conn,'SnowMan',
                'DELETE {?positionObj :contains ?object}
                INSERT {?c :contains ?object}
                        WHERE {?positionObj :contains ?object.
                                ?positionObj :hasWest ?c.
                                ?positionObj :hasEast ?position.
                                ?position :contains :snowMan
                                FILTER (?c != :wall )
                                FILTER NOT EXISTS {?c :contains ?ob
                DELETE {?position :contains :snowMan}
                INSERT { ?cell :contains :snowMan.}
                WHERE { ?position :contains :snowMan.
                                ?position :hasWest ?cell
                                FILTER (?cell != :wall)
                        FILTER NOT EXISTS {?cell :contains ?object}
                }',
                'application/sparql-results+json',
                {});
    return cellPlayer;
```

```
}
```

En l'occurrence, cette requête permet de déplacer le joueur vers le nord (vers le haut). On remarque que dans un premier lieu, on vérifie si la cellule de destination ne contient pas un autre objet. Si celle-ci en contient alors on déplace l'objet vers la cellule d'après (dans le sens de déplacement du joueur) si elle n'est pas un mur et si elle contient un autre objet. Dans un second lieu, déplacer le joueur si la cellule de destination n'est pas un mur et si elle n'est pas occupée.

Pour permettre le bon fonctionnement des résolveurs, les schéma des requêtes en modifiant le fichier *modele.graphql*. On ajoute ainsi les lignes suivantes :

```
type snowManQueries{
  value: String
}
```

```
type MyQueryType{
  # returns a small json chunk
   cells(id: ID):[snowManQueries]
   cellPlayer(id: ID):[snowManQueries]
  cellLittleBall(id: ID):[snowManQueries]
  cellMiddleBall(id: ID):[snowManQueries]
   cellBigBall(id: ID):[snowManQueries]
}
```

```
type Mutation{
        movePlayerNorth(id: ID): snowManQueries
        movePlayerSouth(id: ID): snowManQueries
        movePlayerEast(id: ID): snowManQueries
        movePlayerWest(id: ID): snowManQueries
}
```

Puis onse place dans le répertoire du serveur pour lancer les commandes suivantes :

```
$ npm install
$ npm run dev
```

On pourra ainsi accéder au endpoint *Graphql* (*http ://localhost :3002/graphiql*) et y faire des requêtes *Graphql* sur les données de la base de données.

IV Mise en place du front-end de l'application

Afin de permettre à un utilisateur de pouvoir jouer au jeu du bonhomme de neige, il a fallu mettre en place une interface utilisateur. Pour ce faire, on utilise la technologie *ReactJS*. En effet, on commence d'abord par connecter la vue au serveur, qui permet d'accéder à la base de données du jeu, avec l'aide d'un *client apollo*.

```
const client = new ApolloClient({
  uri: "http://localhost:3002/graphql"
});
```

Puis on fait appel au composant *ApolloProvider* de *react-apollo*. On passe le client comme attribut au composant *ApolloProvider* afin de permettre aux autres éléments de l'application de bénéficier des services du client.

Différents composants ont été créer pour visualiser les données et pour pouvoir interagir avec elles. Il s'agit des composants *Grid*, *Cell* et *Controller*.

IV.1 Grid

Grid permet de représenter la grille. D'abord il récupères toutes cellules de la base en utilisant l'outil *gql* de *graphql-tag* et l'élément *Query* de *react-apollo*. En effet l'élément *Query* prend comme attribut un *query* qui a pour valeur le schéma de la requête qui permet de récupérer les différentes cellules de la grille.

```
const GRID_CELLS = gql`{
                    cells {
                            value
                    }
                    cellPlayer{
                            value
                    }
                    cellLittleBall{
                            value
                    }
                    cellMiddleBall{
                            value
                    }
                    cellBigBall{
                            value
                    }}
```

‘;

```
<Query query={GRID_CELLS}>
</Query>
```

Ensuite dans l'élément *Query*, on parcours les résultat de la requête concernant *cells* en demandant à chaque fois le composant *Cell* de représenter les données de cette cellule. L'élément Cell aura comme attributs *name* qui aura pour valeur l'*URI* de la cellule qu'on veut représenter et *noneFreeCells* qui est un objet JavaScript qui contient les l'*URI* des cellules occupées. Sachant que dans *noneFreeCells*, le nom de la celle est de la forme *Cell+ObjetOccupant* (EX : CellBigBall).

IV.2 Cell

Cell représente une cellule. Il vérifie si la cellule qu'on veut afficher est dans son *props noneFreeCells*. Si c'est le cas, il structure la cellule avec l'une des images ci-dessous.

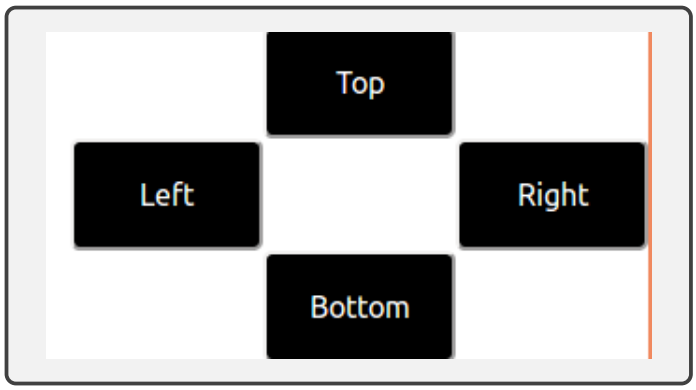
FIGURE 2 – Images représentant les différents objets

On note que L représente Little ball, M représente Middle ball, B représente Big ball et le bonhomme vert représente le joueur.

Si le cellule n'est pas occupée, on utilise l'image ci-contre.

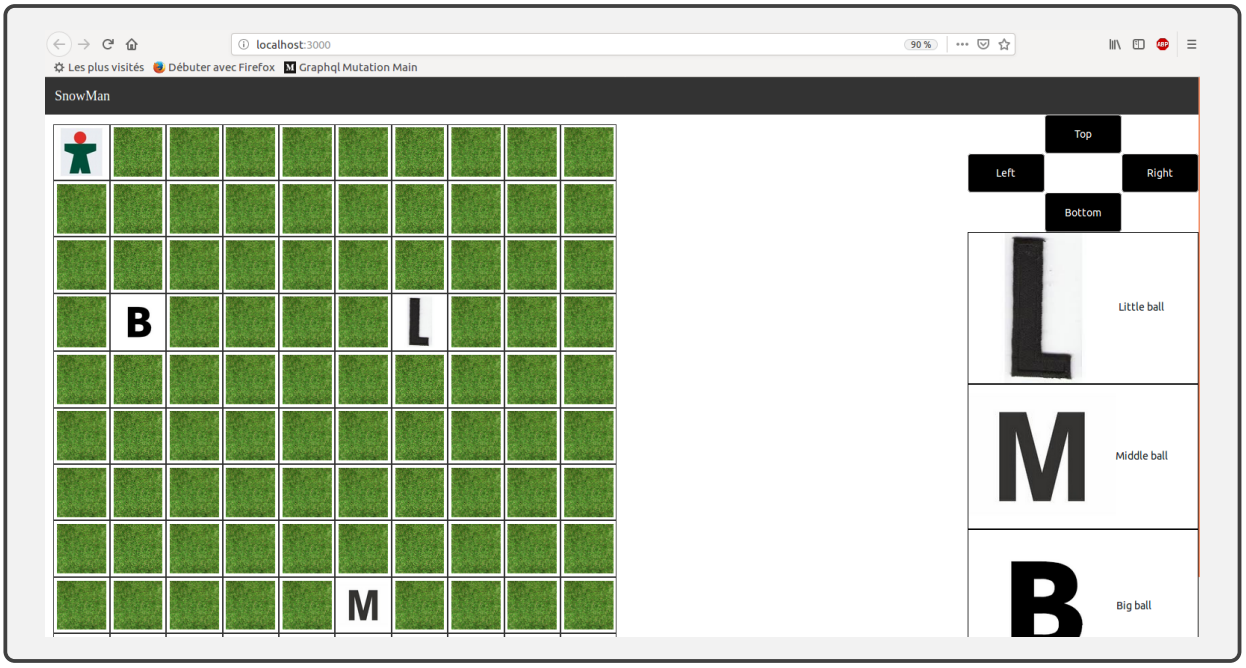
IV.3 Controller

Ce composant gère les déplacements du joueur. Il est constituer de quatre autres composants à savoir Left qui déplace vers la gauche, Right qui déplace vers la droite, Top et Bottom qui déplace le joueur vers, respectivement, le haut et le bas. Ces composants sont représentés à l'affichage par des boutons. Chaque composant est associé à une mutation qui modifie les données de la base en fonction du déplacement effectuer.



V Bilan

À l'état actuel de l'application, on peut lancer le jeu et avoir l'affichage de la grille et du contrôleur. On peut aussi déplacer le joueur à notre guise. Cependant, lors d'un déplacement du joueur, la grille ne se met pas à jour. Il est nécessaire de redémarrer le serveur pour apercevoir le déplacement du joueur ou des et/ou des autres object. Même l'utilisation de l'outil *refetchQueries* n'a pas pu résoudre ce problème.


localhost:3000
90 %
Les plus visités
Débuter avec Firefox
Graphql Mutation Main
SnowMan
B
L
M
Top
Left
Right
Bottom
Little ball
Middle ball
Big ball

VI Conclusion

Pour améliorer cette application, ça pourrait être utilise de pouvoir voir les modifications dans la grille lors d'un déplacement du joueur. Il pourrait être intéressant de pouvoir faire varier la taille de grille et le nombre d'objets.

En définitif, ce projet m'a permis de me familiariser avec les technologies *ReactJS*, *Graphql* et *Stardog* qui sont des technologies assez intéressantes du le monde du développement web.

Webographie

https://fr.wikipedia.org/
https://reactjs.org/
https://www.stardog.com/
https://www.apollographql.com/
https://blog.apollographql.com/react-graphql-tutorial-mutations-764d7ec23c15
https://www.youtube.com/watch?v=5evJqX5i1zE