

Étude et expérimentation d'un framework pour la réalisation collaborative d'une application web

Projet du cours « Réalisation de programme »

Sylvain Labopin

19 Juin, 2024

Table des matières

1

Introduction

- Étude d'un framework
- Expérimentation du framework

2

Comprendre l'application via son diagramme des classes

3

Conclusion

- Modélisation, persistance et centralisation CDI
- Gestion des comptes utilisateur
- Implémentation d'un moteur de décision min-max
- Implémentation d'un utilisateur virtuel intelligent

Étude d'un framework :

→ Spécification Jakarta EE

Étude d'un framework :

- Spécification Jakarta EE
- Serveur d'application Tomcat

Étude d'un framework :

- Spécification Jakarta EE
- Serveur d'application Tomcat
- Logiciel de compréhension et de construction Maven

Étude d'un framework :

- Spécification Jakarta EE
- Serveur d'application Tomcat
- Logiciel de compréhension et de construction Maven
- Technologie AJAX

Étude d'un framework :

- Spécification Jakarta EE
- Serveur d'application Tomcat
- Logiciel de compréhension et de construction Maven
- Technologie AJAX
- Bibliothèque Javascript React

Étude d'un framework :

- Spécification Jakarta EE
- Serveur d'application Tomcat
- Logiciel de compréhension et de construction Maven
- Technologie AJAX
- Bibliothèque Javascript React
- Les services usuels d'Internet pour le déploiement sécurisé par certificat Let's Encrypt

Étude d'un framework :

- Spécification Jakarta EE
- Serveur d'application Tomcat
- Logiciel de compréhension et de construction Maven
- Technologie AJAX
- Bibliothèque Javascript React
- Les services usuels d'Internet pour le déploiement sécurisé par certificat Let's Encrypt
- API de AbuseIPDB

Étude d'un framework :

- Spécification Jakarta EE
- Serveur d'application Tomcat
- Logiciel de compréhension et de construction Maven
- Technologie AJAX
- Bibliothèque Javascript React
- Les services usuels d'Internet pour le déploiement sécurisé par certificat Let's Encrypt
- API de AbuselPDB
- Conception : diagrammes UML et de cas d'utilisation, architecture multi-tiers

Étude d'un framework :

- Spécification Jakarta EE
- Serveur d'application Tomcat
- Logiciel de compréhension et de construction Maven
- Technologie AJAX
- Bibliothèque Javascript React
- Les services usuels d'Internet pour le déploiement sécurisé par certificat Let's Encrypt
- API de AbuselPDB
- Conception : diagrammes UML et de cas d'utilisation, architecture multi-tiers

Expérimentation du framework :

- ➡ Développement d'une application web multi-joueurs pour le Puissance 4 proposant une gestion des comptes utilisateur usuelle

Expérimentation du framework :

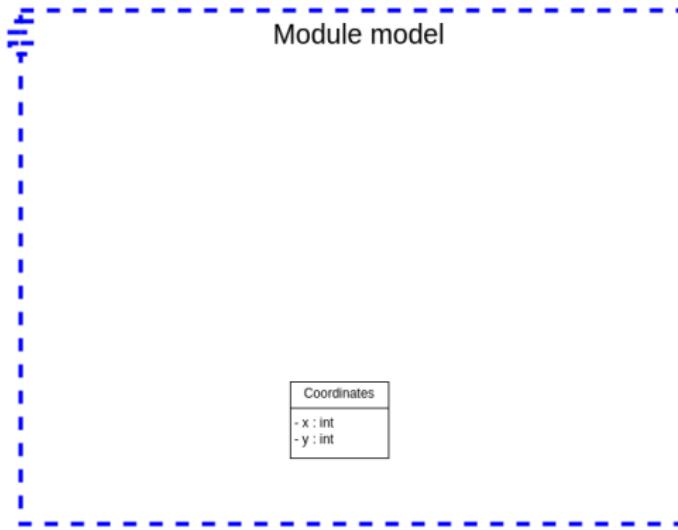
- ➔ Développement d'une application web multi-joueurs pour le Puissance 4 proposant une gestion des comptes utilisateur usuelle
- ➔ Développement d'un serveur amorçant un utilisateur virtuel de l'application web muni d'un moteur de décision min-max

Expérimentation du framework :

- ➔ Développement d'une application web multi-joueurs pour le Puissance 4 proposant une gestion des comptes utilisateur usuelle
- ➔ Développement d'un serveur amorçant un utilisateur virtuel de l'application web muni d'un moteur de décision min-max

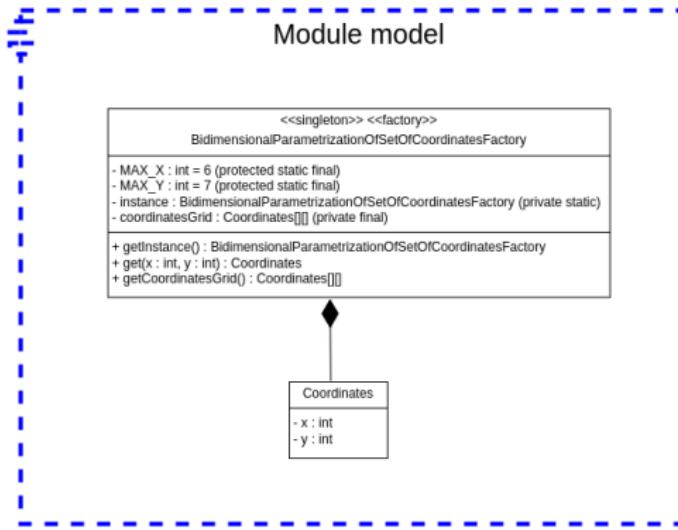
Modélisation, persistance et centralisation CDI

Coordinates modélise les cases de la grille de Puissance 4.



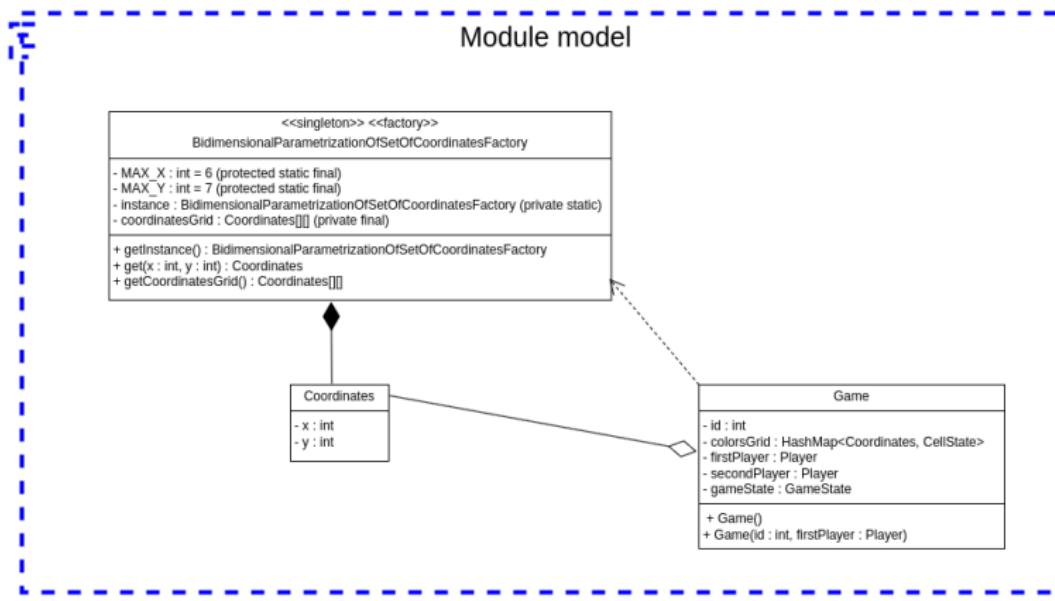
Modélisation, persistance et centralisation CDI

42 uniques instances via les patterns « Singleton » et « Factory »



Modélisation, persistance et centralisation CDI

Games modélise les parties de Puissance 4.



Modélisation, persistance et centralisation CDI

Coordinates → CellState pour l'état de la grille lors d'une partie.

Game

...

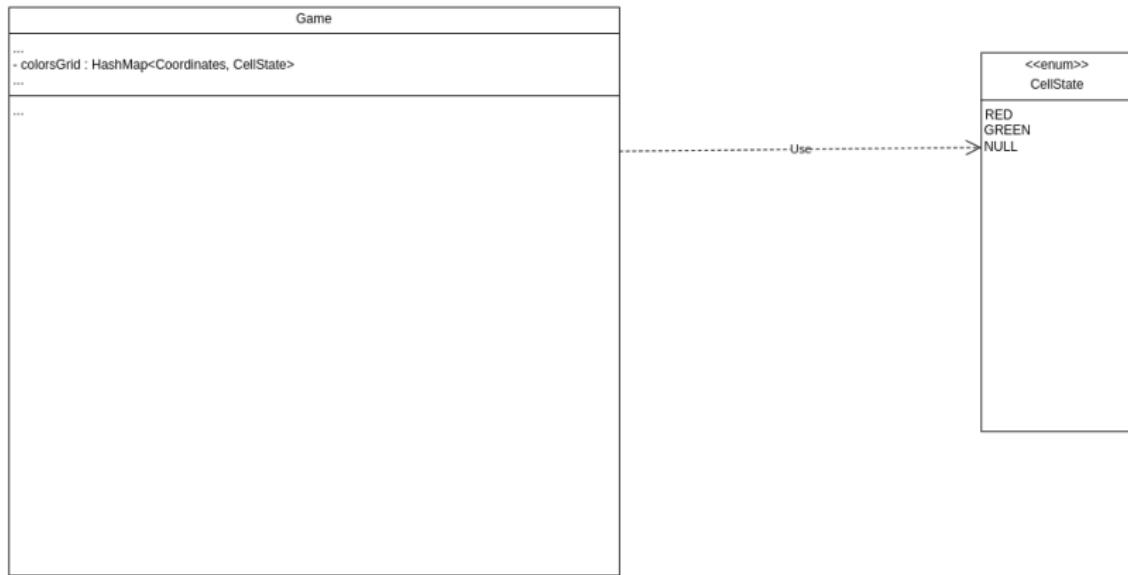
- colorsGrid : HashMap<Coordinates, CellState>

...

...

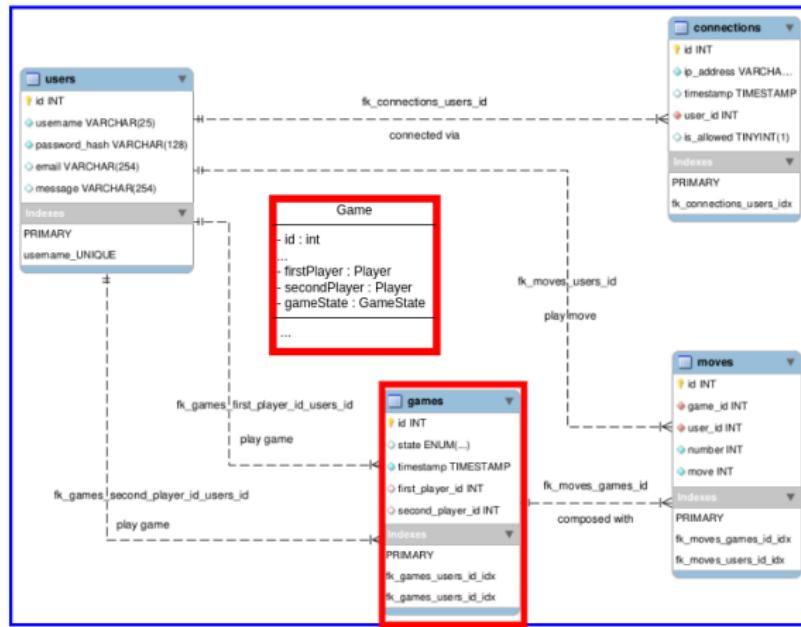
Modélisation, persistance et centralisation CDI

CellState modélise l'état d'une case : rouge, vert ou vide.



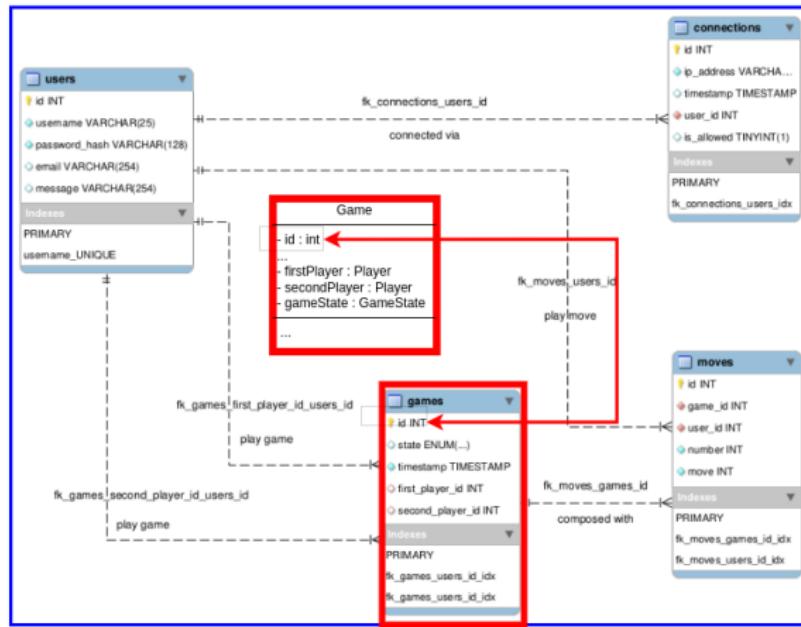
Modélisation, persistance et centralisation CDI

instance de Game ↔ enregistrement de la table games



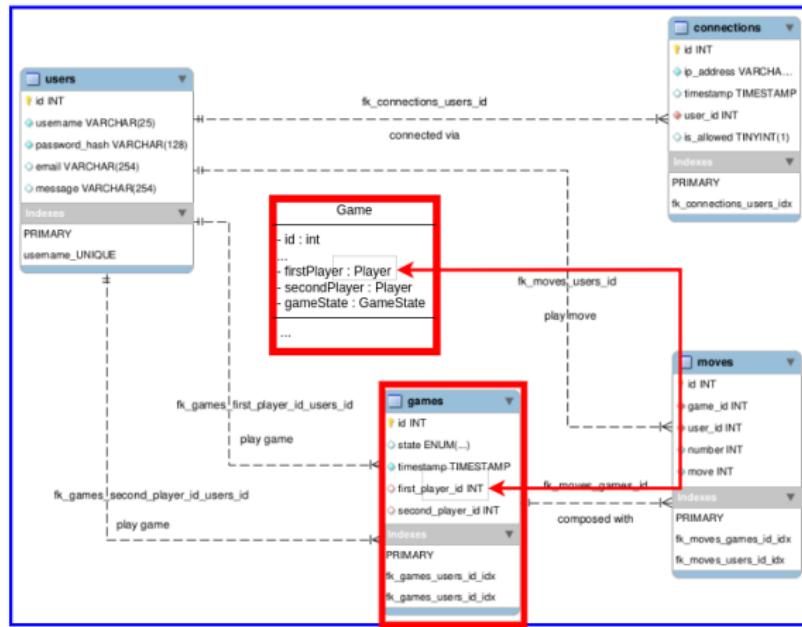
Modélisation, persistance et centralisation CDI

attribut **id** ↔ clé primaire



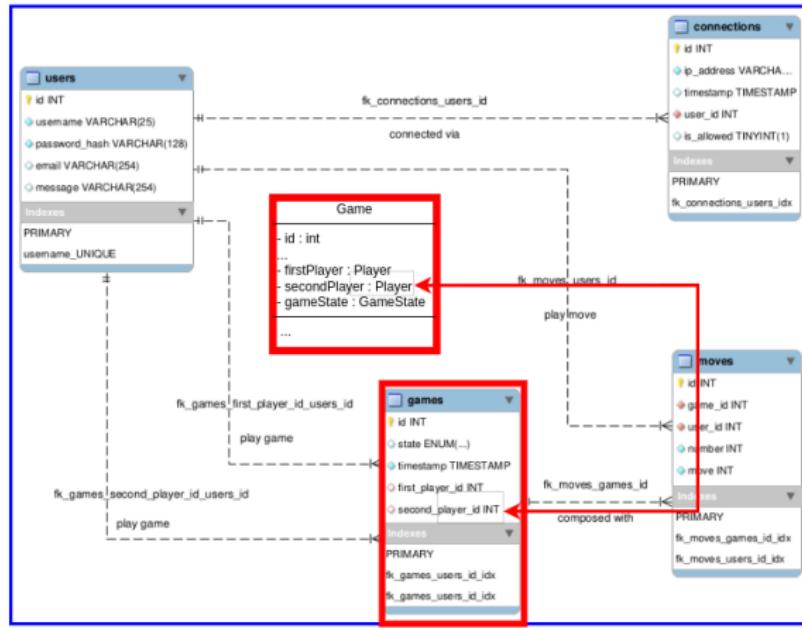
Modélisation, persistance et centralisation CDI

attribut **firstPlayer** ↔ identifiant de l'instance de User pour joueur 1



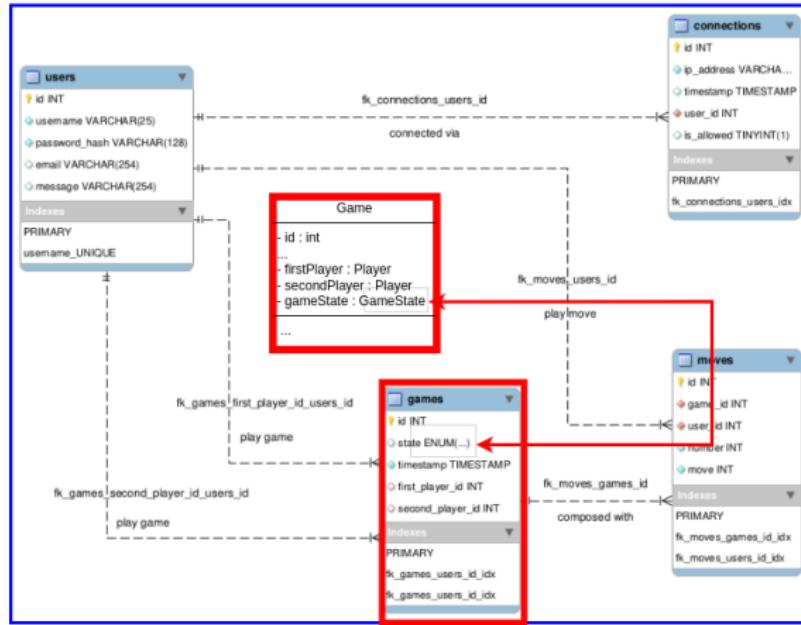
Modélisation, persistance et centralisation CDI

attribut **secondPlayer** ↔ identifiant de l'instance de User joueur 2



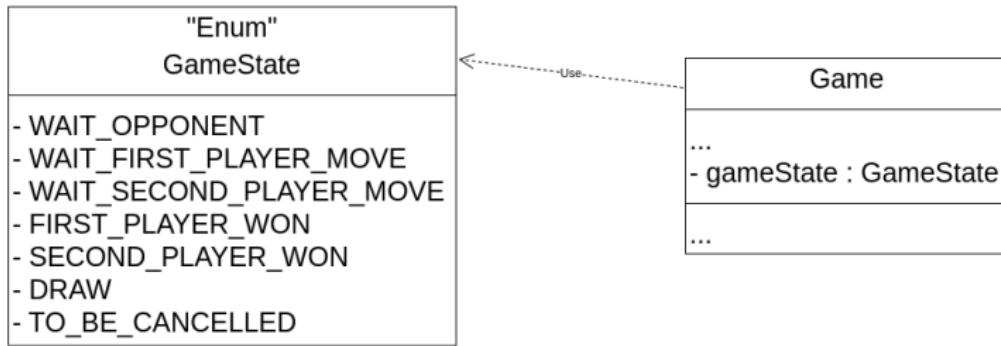
Modélisation, persistance et centralisation CDI

attribut gameState ↔ valeur du champ state



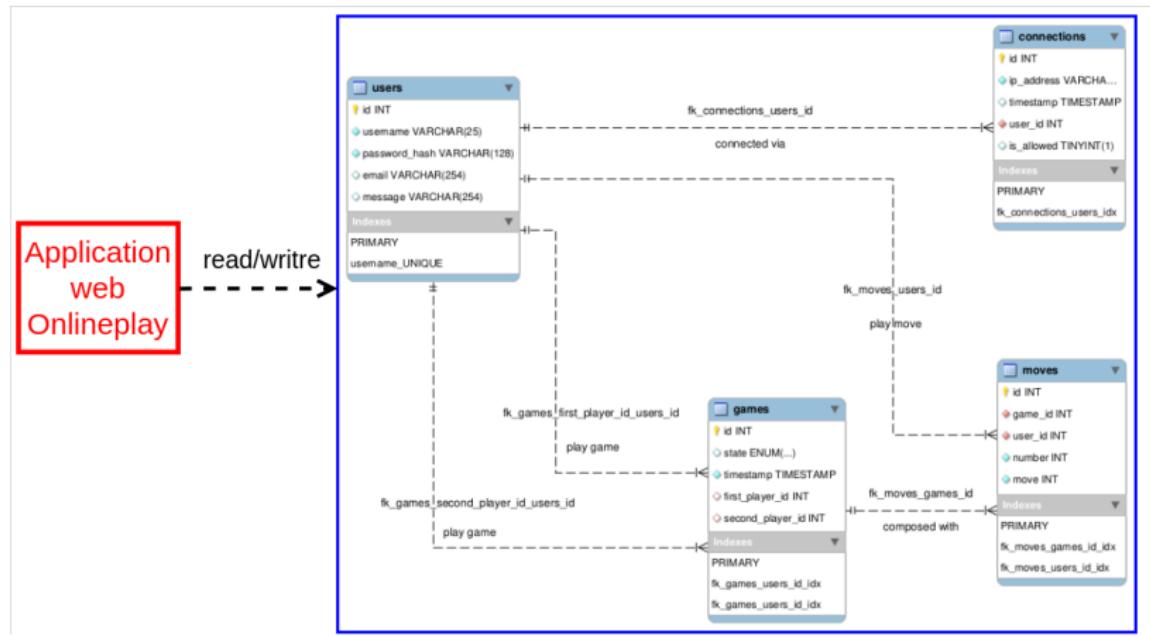
Modélisation, persistance et centralisation CDI

GameState stocke les états d'une partie.



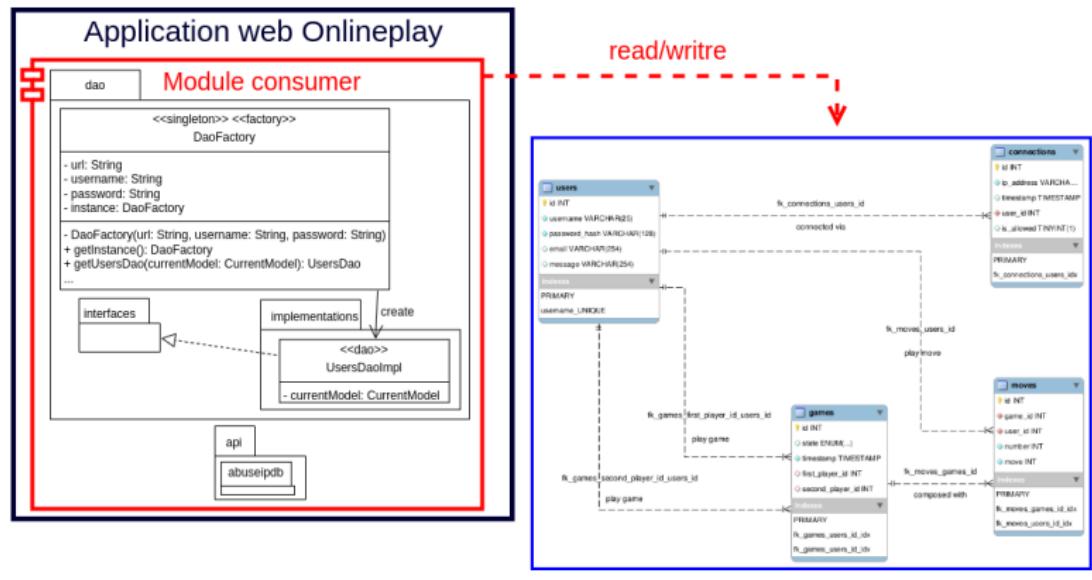
Modélisation, persistance et centralisation CDI

L'application web administre la BDD.



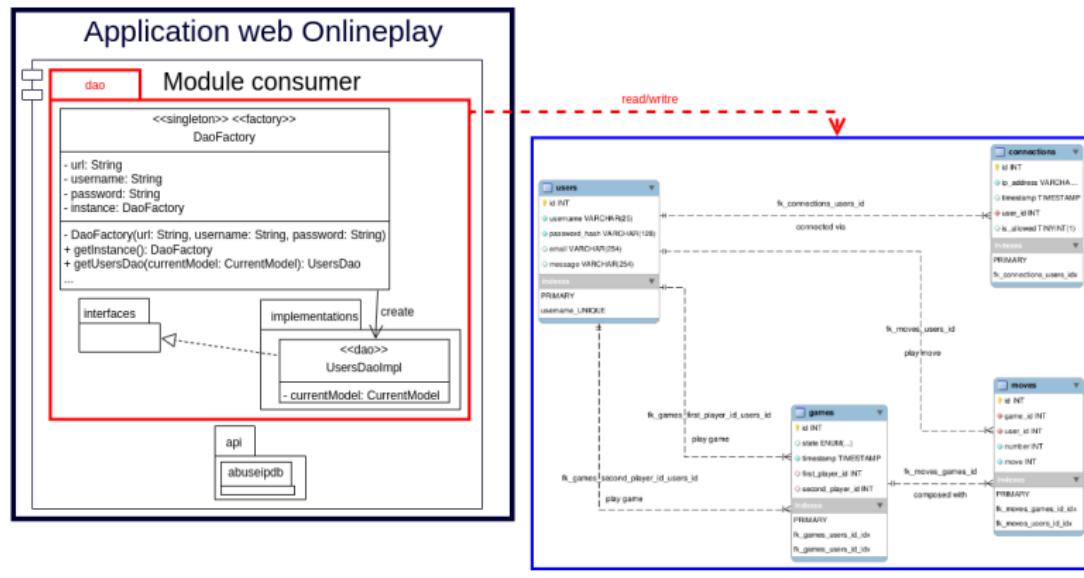
Modélisation, persistance et centralisation CDI

Le module consumer gère la communication avec les services tiers.



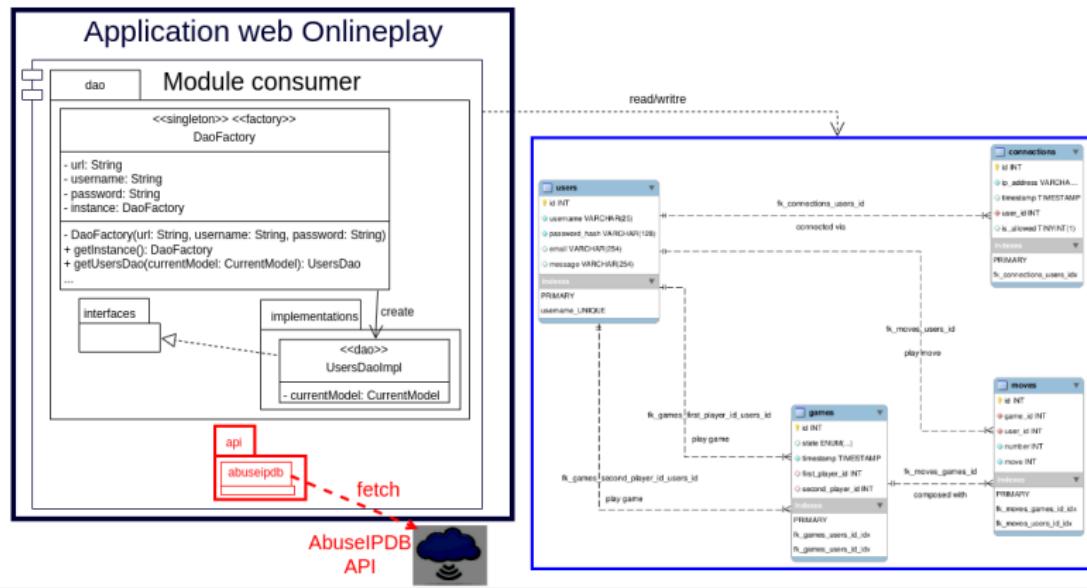
Modélisation, persistance et centralisation CDI

Le sous-package dao administre la BDD.



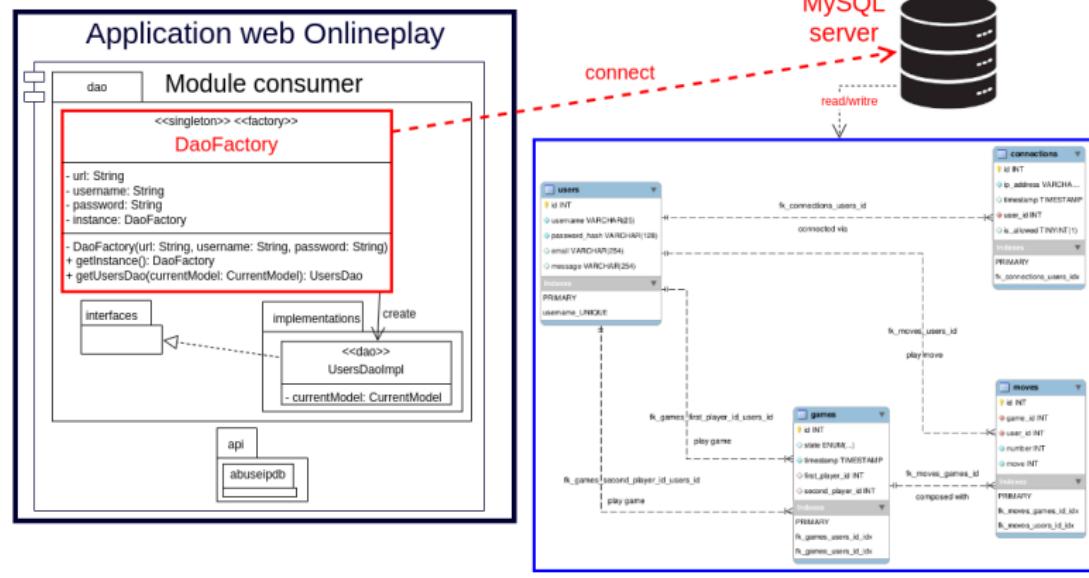
Modélisation, persistance et centralisation CDI

Le sous-package api gère la communication avec les API tierces.



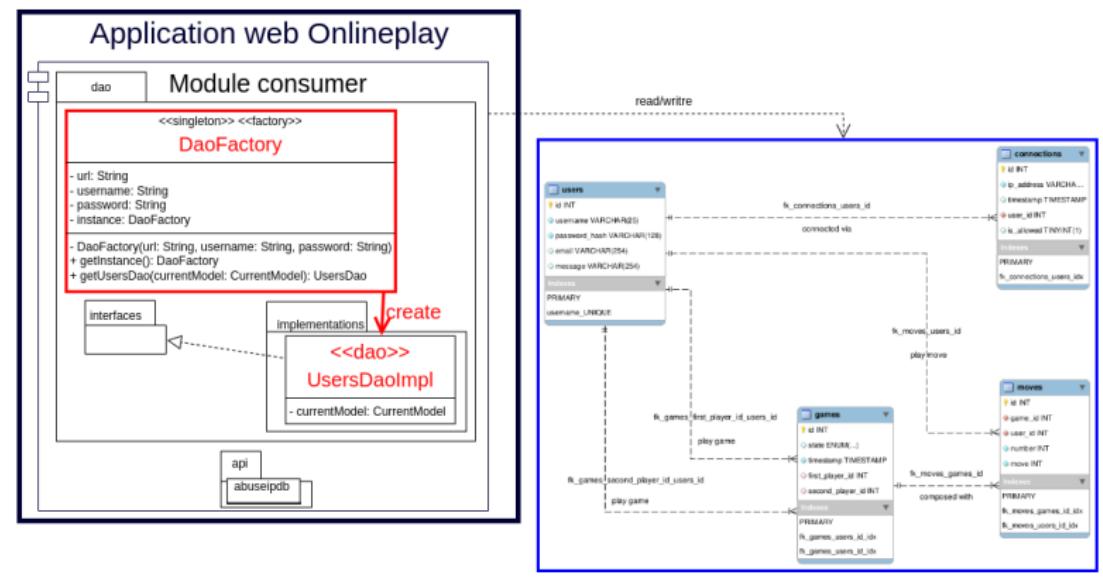
Modélisation, persistance et centralisation CDI

DaoFactory gère la connexion au serveur MySQL...



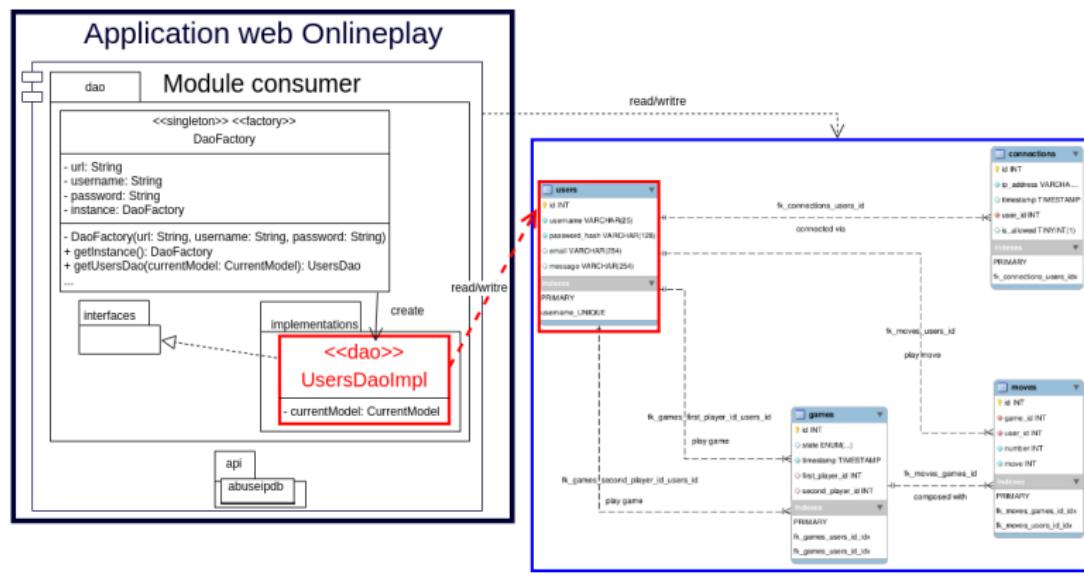
Modélisation, persistance et centralisation CDI

...et DaoFactory crée des DAO comme UserDaoImpl.



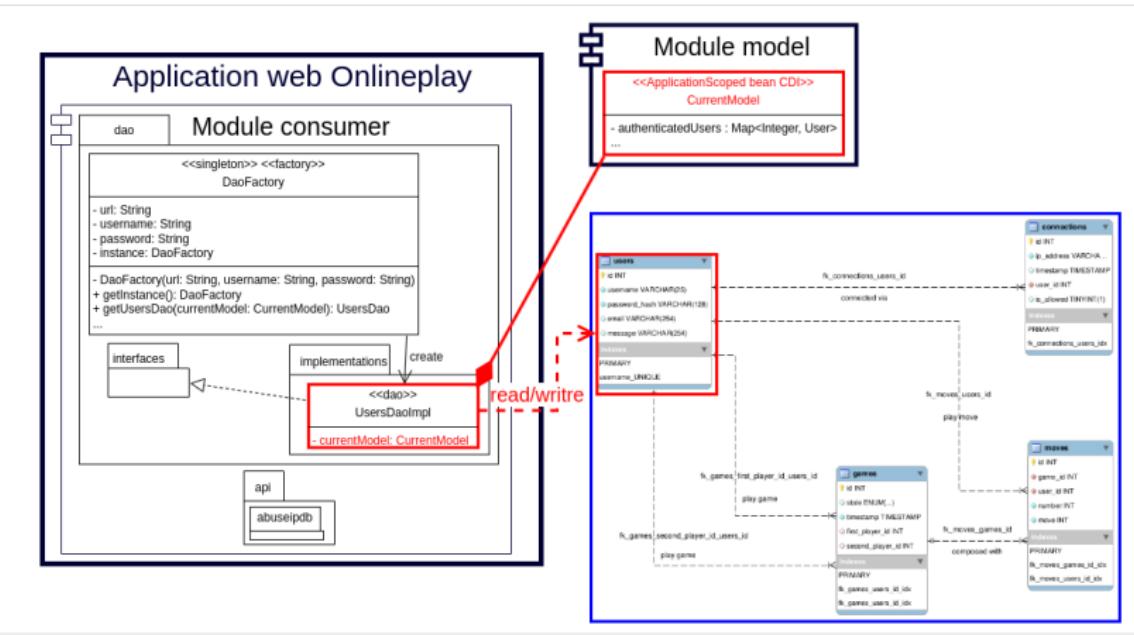
Modélisation, persistance et centralisation CDI

Le DAO UserDaolmpl administre la table users.



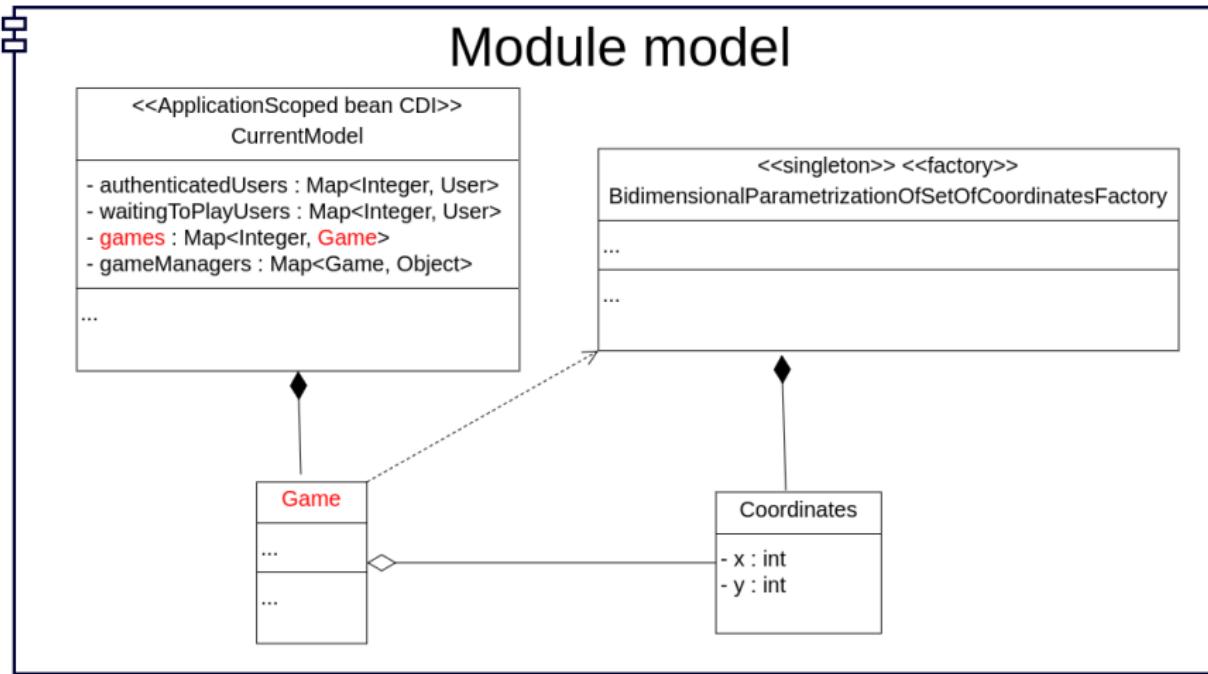
Modélisation, persistance et centralisation CDI

Ce DAO aggrège CurrentModel structurant les données centralisées.



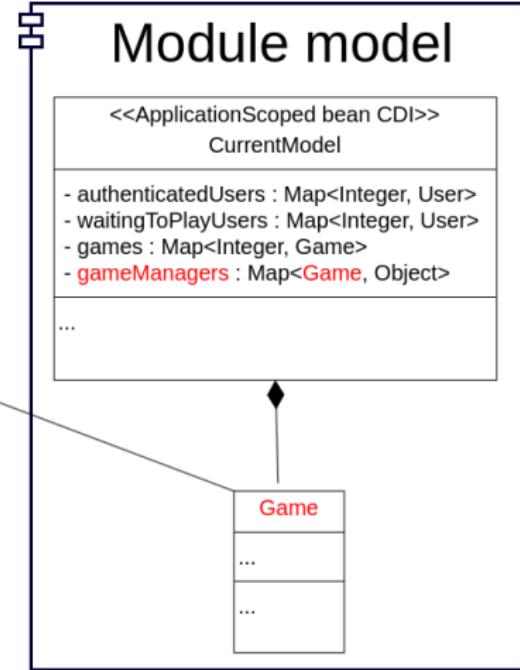
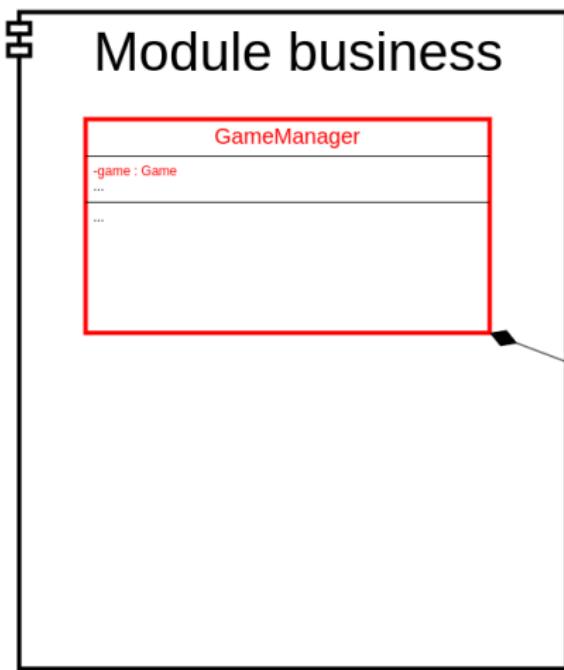
Modélisation, persistance et centralisation CDI

CurrentModel aggrège les parties suivant les clés primaires...



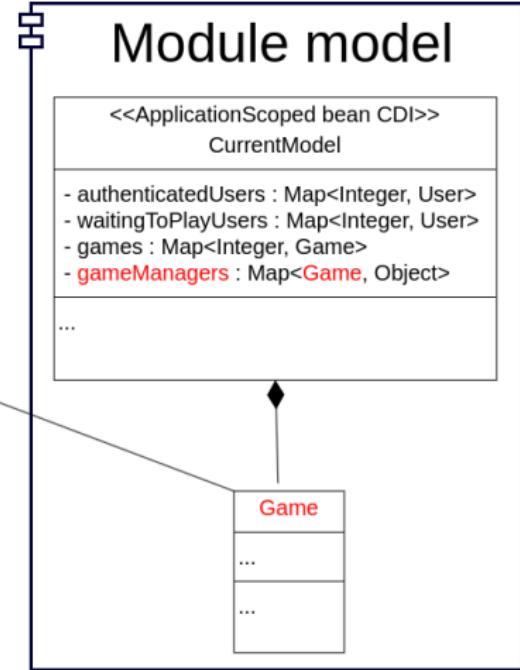
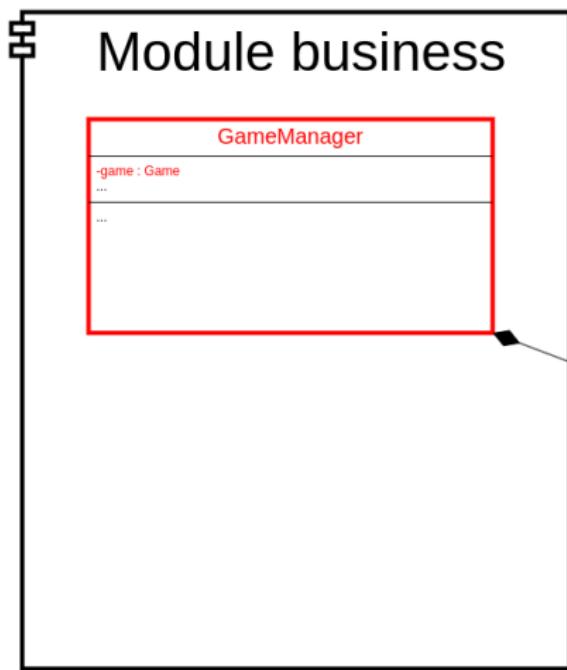
Modélisation, persistance et centralisation CDI

...et l'attribut gameManager leur associe des GameManager.



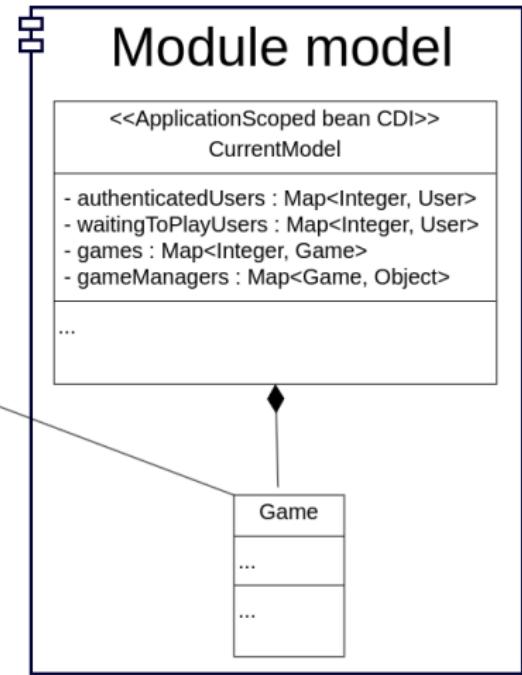
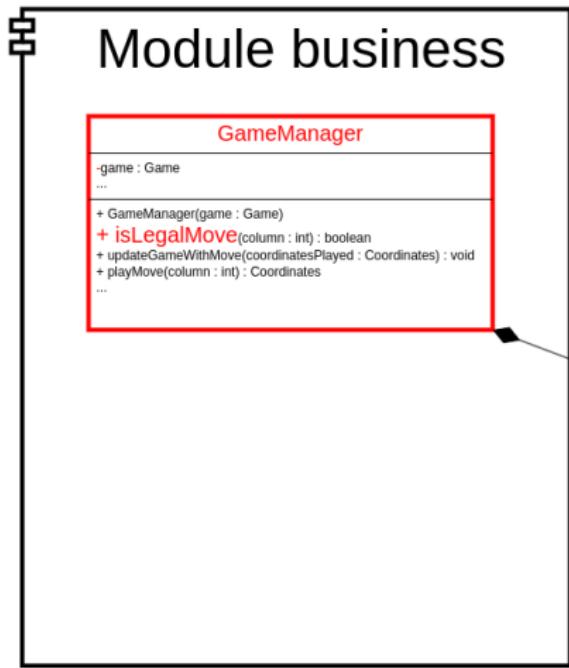
Modélisation, persistance et centralisation CDI

GameManager contrôle le déroulement d'une partie.



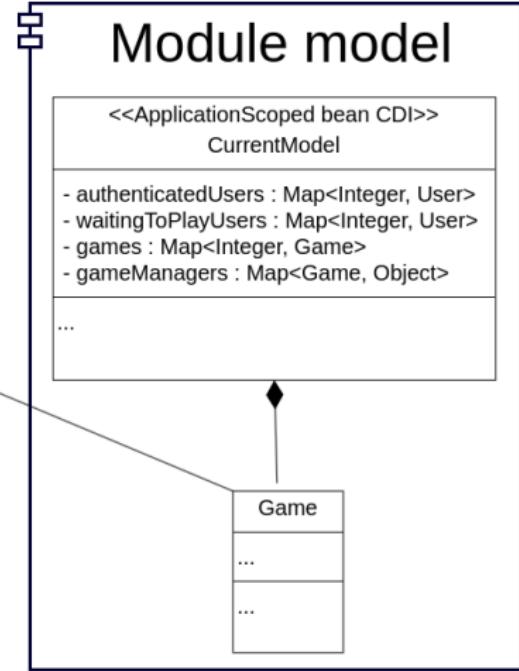
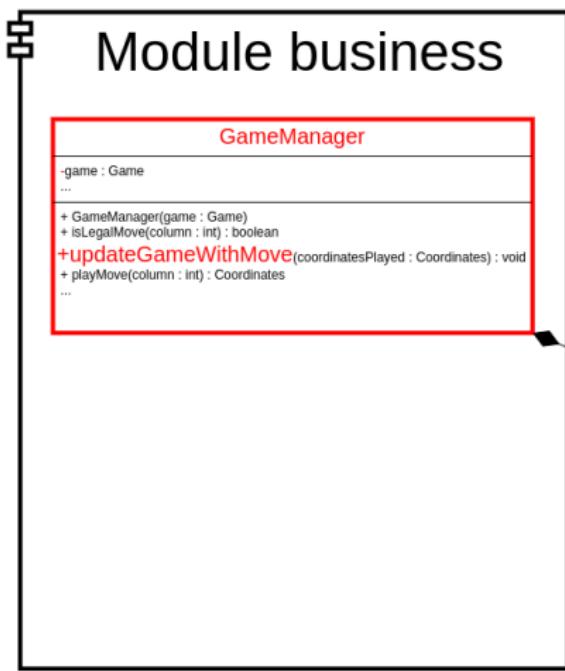
Modélisation, persistance et centralisation CDI

GameManager contrôle si un coup est légal.



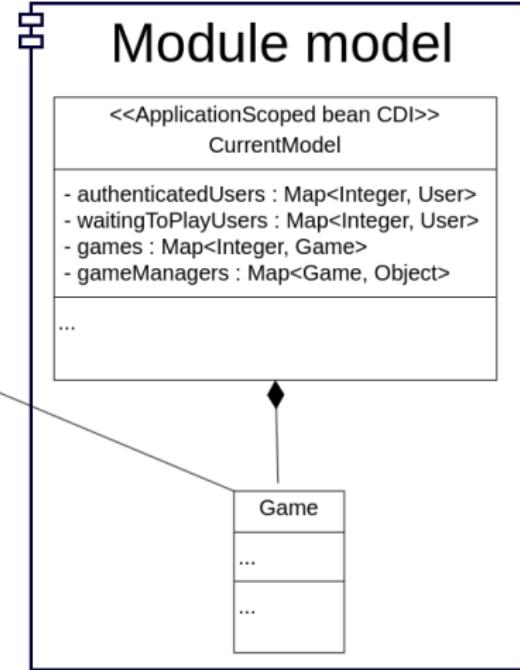
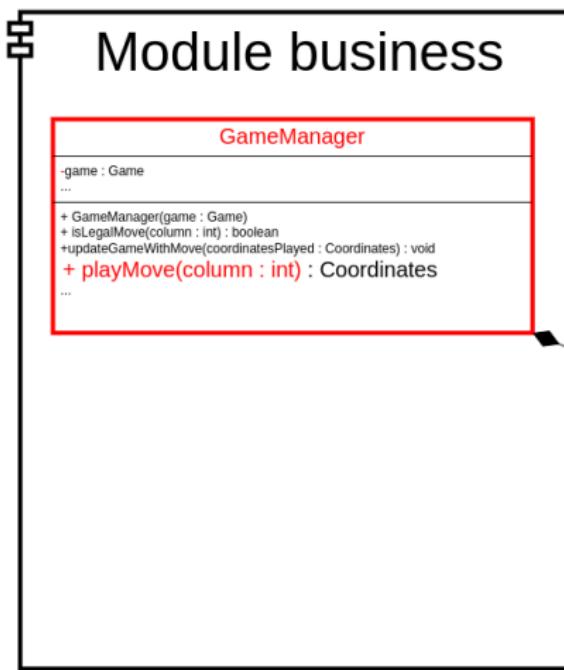
Modélisation, persistance et centralisation CDI

GameManager met à jour son attribut Game.



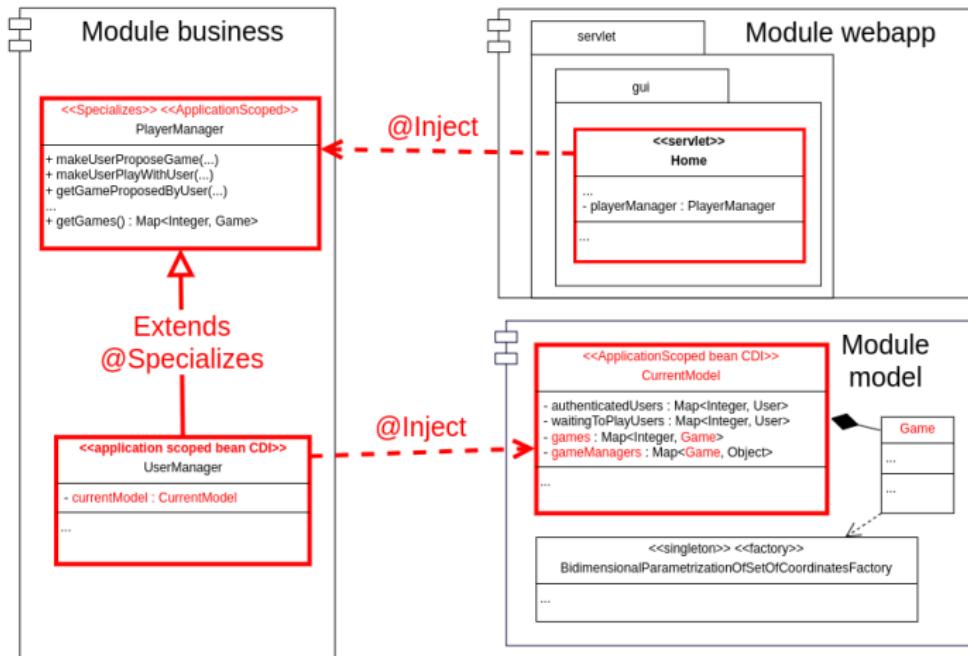
Modélisation, persistance et centralisation CDI

La méthode playMove combine les deux précédentes.



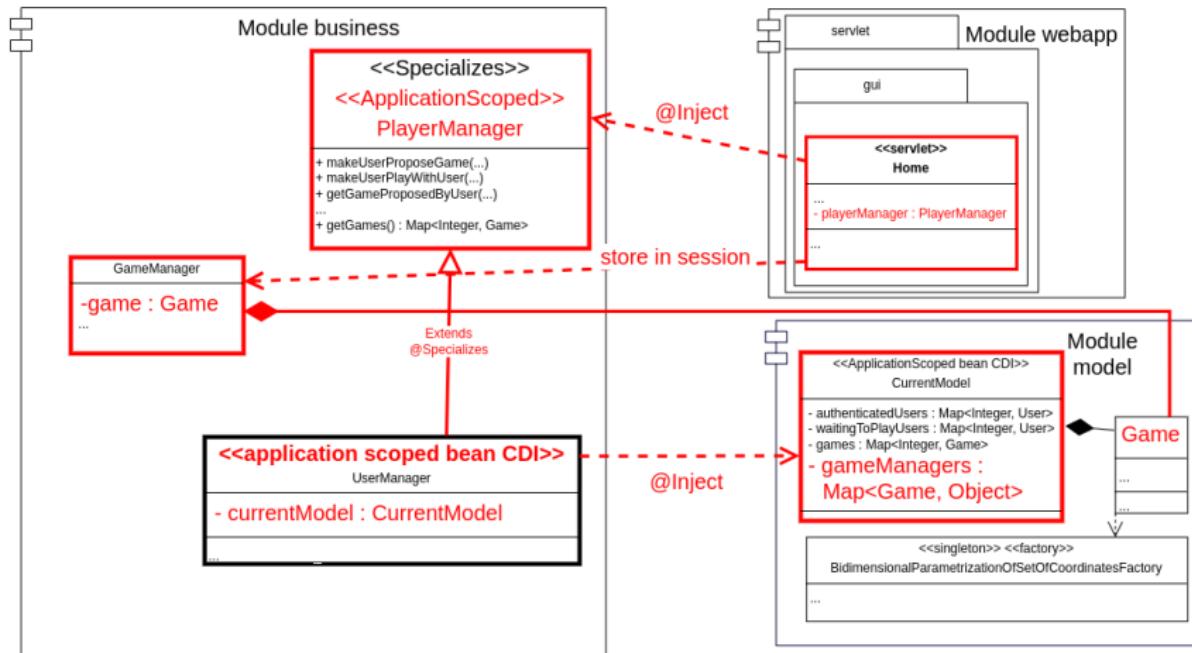
Modélisation, persistance et centralisation CDI

Le mécanisme CDI permet la récupération des données centralisées.



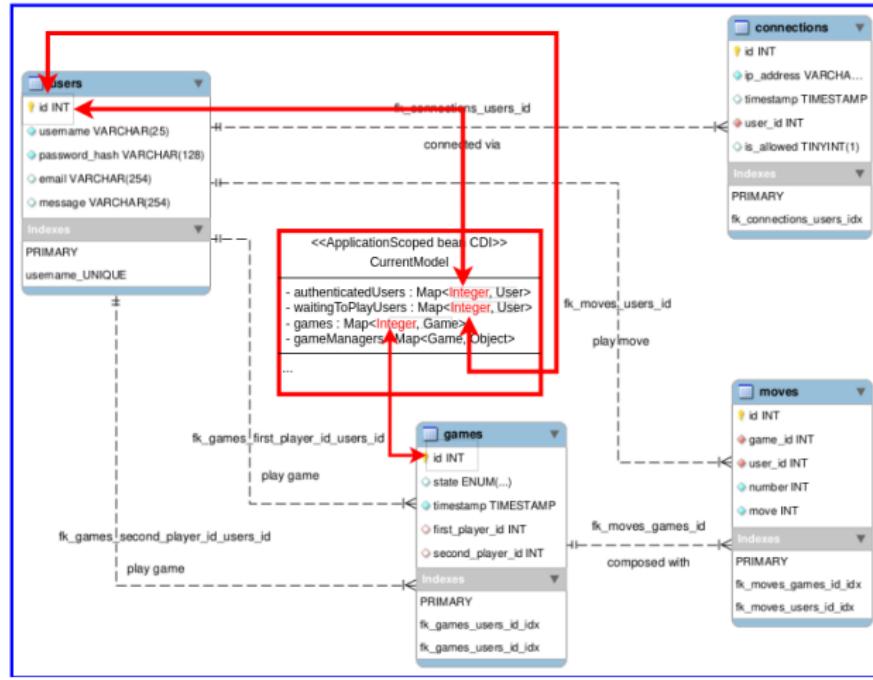
Modélisation, persistance et centralisation CDI

Récupération et stockage dans la session d'un GameManager.



Modélisation, persistance et centralisation CDI

clé d'un dictionnaire de CurrentModel ↔ clé primaire d'une table



Modélisation, persistance et centralisation CDI

clé d'un dictionnaire de CurrentModel ↔ clé primaire d'une table

UserDaoImpl
gère la table
users.

```
public class UserDaoImpl implements UserDao {  
    protected CurrentModel currentModel;  
    private final DaoFactory daoFactory;  
    //...  
    public User addUser(String username, String passwordHash, String email, String message) throws... {  
        User user = null;  
        //...  
        int generatedId = -1;  
        try {  
            connection = daoFactory.getConnection();  
            preparedStatement = connection.prepareStatement(  
                "INSERT INTO users (username, password_hash, email, message) VALUES (?, ?, ?, ?);",  
                Statement.RETURN_GENERATED_KEYS  
            );  
            //...  
            // fetch user id  
            int affectedRows = preparedStatement.executeUpdate();  
            if (affectedRows > 0) {  
                try (ResultSet generatedKeys = preparedStatement.getGeneratedKeys()) {  
                    if (generatedKeys.next()) {  
                        generatedId = generatedKeys.getInt(1);  
                        user.setId(generatedId);  
                    }  
                }  
                connection.commit();  
            } catch (SQLException e) {  
                //...  
            }  
            //...  
            return user;  
        }
```

Modélisation, persistance et centralisation CDI

clé d'un dictionnaire de CurrentModel ↔ clé primaire d'une table

Enregistrement
d'un utilisateur
dans la table
users

```
public class UsersDaoImpl implements UsersDao {  
    protected CurrentModel currentModel;  
    private final DaoFactory daoFactory;  
    //...  
    public User addUser(String username, String passwordHash, String email, String message) throws... {  
        User user = null;  
        //...  
        int generatedId = -1;  
        try {  
            connection = daoFactory.getConnection();  
            preparedStatement = connection.prepareStatement(  
                "INSERT INTO users (username, password_hash, email, message) VALUES (?, ?, ?, ?);",  
                Statement.RETURN_GENERATED_KEYS  
            );  
            //...  
            // fetch user id  
            int affectedRows = preparedStatement.executeUpdate();  
            if (affectedRows > 0) {  
                try (ResultSet generatedKeys = preparedStatement.getGeneratedKeys()) {  
                    if (generatedKeys.next()) {  
                        generatedId = generatedKeys.getInt(1);  
                        user.setId(generatedId);  
                    }  
                }  
                connection.commit();  
            } catch (SQLException e) {  
                //...  
            }  
            //...  
            return user;  
        }
```

Modélisation, persistance et centralisation CDI

clé d'un dictionnaire de CurrentModel ↔ clé primaire d'une table

Enregistrement
d'un utilisateur
dans la table
users



Récupération
de la clé
 primaire

```
public class UsersDaoImpl implements UsersDao {  
    protected CurrentModel currentModel;  
    private final DaoFactory daoFactory;  
    //...  
    public User addUser(String username, String passwordHash, String email, String message) throws... {  
        User user = null;  
        //...  
        int generatedId = -1;  
        try {  
            connection = daoFactory.getConnection();  
            preparedStatement = connection.prepareStatement(  
                "INSERT INTO users (username, password_hash, email, message) VALUES (?, ?, ?, ?);",  
                Statement.RETURN_GENERATED_KEYS  
            );  
            //...  
            // fetch user id  
            int affectedRows = preparedStatement.executeUpdate();  
            if (affectedRows > 0) {  
                try (ResultSet generatedKeys = preparedStatement.getGeneratedKeys()) {  
                    if (generatedKeys.next()) {  
                        generatedId = generatedKeys.getInt(1);  
                        user.setId(generatedId);  
                    }  
                }  
                connection.commit();  
            } catch (SQLException e) {  
                //...  
            }  
            //...  
        }  
        return user;  
    }
```

Modélisation, persistance et centralisation CDI

clé d'un dictionnaire de CurrentModel ↔ clé primaire d'une table

La méthode
`getUserById`
construit ou
récupère une
instance de
User

```
public class UsersDaoImpl implements UsersDao {  
    protected CurrentModel currentModel;  
    private final DaoFactory daoFactory;  
    ...  
    public User getUserById(int id) throws DaoException {  
        // try to find the User instance in the CurrentModel @ApplicationScoped bean ;  
        User user = ((Map<Integer, User>) currentModel.getAuthenticatedUsers()).get(id);  
        if (user != null) {  
            return user;  
        }  
  
        // try to construct the User instance with respect to information in the users table  
        String sql = "SELECT id, username, password_hash, email, message FROM users WHERE id = ?";  
        try (Connection connection = daoFactory.getConnection();  
             PreparedStatement statement = connection.prepareStatement(sql)) {  
  
            statement.setInt(1, id);  
            ResultSet resultSet = statement.executeQuery();  
  
            if (resultSet.next()) {  
                user = new User();  
                try {  
                    user.setId(resultSet.getInt("id"));  
                    user.setUsername(resultSet.getString("username"));  
                    user.setPasswordHash(resultSet.getString("password_hash"));  
                    user.setEmail(resultSet.getString("email"));  
                    user.setMessage(resultSet.getString("message"));  
                } catch (UserException e) {}  
                return user;  
            }  
            // ...  
        } catch (SQLException e) {  
            throw new DaoException("Error fetching user with ID " + id + ": " + e.getMessage());  
        }  
    }  
}
```

Modélisation, persistance et centralisation CDI

clé d'un dictionnaire de CurrentModel \leftrightarrow clé primaire d'une table

Recherche une instance associée à la clé primaire dans les données centralisée

```
public class UsersDaoImpl implements UsersDao {  
    protected CurrentModel currentModel;  
    private final DaoFactory daoFactory;  
    //...  
    public User getUserById(int id) throws DaoException {  
  
        // try to find the User instance in the CurrentModel @ApplicationScoped bean ;  
        User user = ((Map<Integer, User>) currentModel.getAuthenticatedUsers()).get(id);  
        if (user != null) {  
            return user;  
        }  
  
        // try to construct the User instance with respect to information in the users table  
        String sql = "SELECT id, username, password_hash, email, message FROM users WHERE id = ?";  
        try (Connection connection = daoFactory.getConnection();  
             PreparedStatement statement = connection.prepareStatement(sql)) {  
  
            statement.setInt(1, id);  
            ResultSet resultSet = statement.executeQuery();  
  
            if (resultSet.next()) {  
                user = new User();  
                try {  
                    user.setId(resultSet.getInt("id"));  
                    user.setUsername(resultSet.getString("username"));  
                    user.setPasswordHash(resultSet.getString("password_hash"));  
                    user.setEmail(resultSet.getString("email"));  
                    user.setMessage(resultSet.getString("message"));  
                } catch (UserException e) {}  
                return user;  
            }  
            // ...  
        } catch (SQLException e) {  
            throw new DaoException("Error fetching user with ID " + id + ": " + e.getMessage());  
        }  
    }  
}
```

Modélisation, persistance et centralisation CDI

clé d'un dictionnaire de CurrentModel ↔ clé primaire d'une table

Recherche une instance associée à la clé primaire dans les données centralisée

↓
Si aucune instance trouvée

Construction d'une instance de User

```
public class UsersDaoImpl implements UsersDao {  
    protected CurrentModel currentModel;  
    private final DaoFactory daoFactory;  
    ...  
    public User getUserById(int id) throws DaoException {  
  
        // try to find the User instance in the CurrentModel @ApplicationScoped bean ;  
        User user = ((Map<Integer, User>) currentModel.getAuthenticatedUsers()).get(id);  
        if (user != null) {  
            return user;  
        }  
  
        // try to construct the User instance with respect to information in the users table  
        String sql = "SELECT id, username, password_hash, email, message FROM users WHERE id = ?";  
        try (Connection connection = daoFactory.getConnection();  
             PreparedStatement statement = connection.prepareStatement(sql)) {  
  
            statement.setInt(1, id);  
            ResultSet resultSet = statement.executeQuery();  
  
            if (resultSet.next()) {  
                user = new User();  
                try {  
                    user.setId(resultSet.getInt("id"));  
                    user.setUsername(resultSet.getString("username"));  
                    user.setPasswordHash(resultSet.getString("password_hash"));  
                    user.setEmail(resultSet.getString("email"));  
                    user.setMessage(resultSet.getString("message"));  
                } catch (UserException e) {}  
                return user;  
            }  
            // ...  
        } catch (SQLException e) {  
            throw new DaoException("Error fetching user with ID " + id + ": " + e.getMessage());  
        }  
    }  
}
```

Gestion des comptes utilisateur

Ajouts d'attributs UserConnection et User dans un HttpSession

Stockage de données utilisateur dans la session

The screenshot shows a code editor with several tabs open. The main tab displays Java code for managing sessions. A red box highlights a section of code where two attributes are being stored in an HttpSession:

```
session.setAttribute("userConnection", userConnection);
session.setAttribute("user", user);
```

Below this, another red box highlights a note about setting the UserConnection attribute:

```
session.setAttribute("userConnection", userConnection). It is essential for authentication.jsp 34
set in the session with session.setAttribute("user", user). If home.jsp 29
home.jsp 34
```

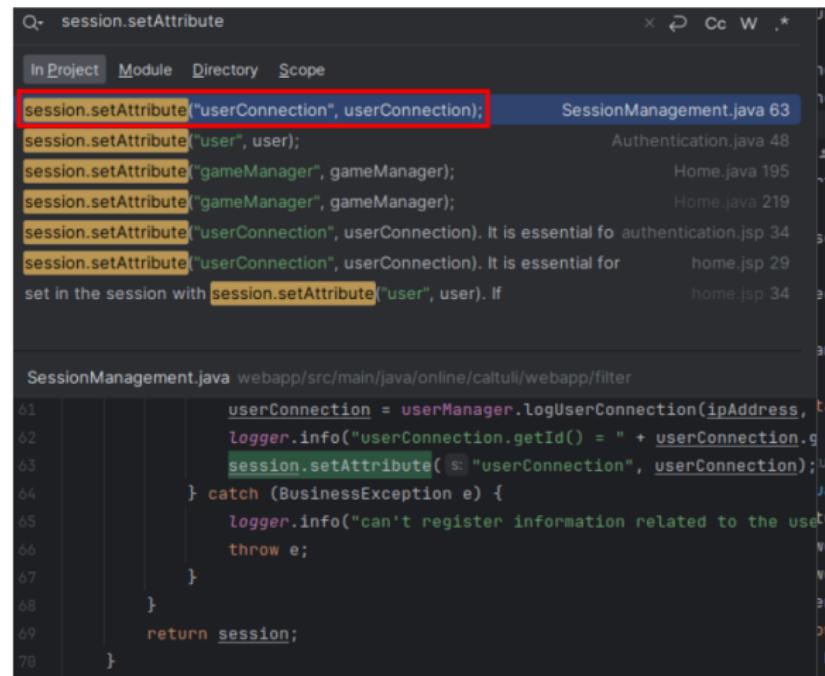
At the bottom, the code for SessionManagement.java is shown:

```
SessionManagement.java webapp/src/main/java/onlinedeal/webapp/filter
61     userConnection = userManager.logUserConnection(ipAddress,
62             logger.info("userConnection.getId() = " + userConnection.getId());
63             session.setAttribute("userConnection", userConnection);
64         } catch (BusinessException e) {
65             logger.info("can't register information related to the user");
66             throw e;
67         }
68     }
69     return session;
70 }
```

Gestion des comptes utilisateur

Ajouts d'attributs UserConnection et User dans un HttpSession

UserConnection
représente une
connexion
authentifiée ou
non



The screenshot shows a code editor with several tabs open. The current tab is 'SessionManagement.java'. A red box highlights the line 'session.setAttribute("userConnection", userConnection);'. Another red box highlights the word 'userConnection' in the line 'userConnection = userManager.logUserConnection(ipAddress, ...)'.

```
session.setAttribute("userConnection", userConnection); // Line highlighted by a red box
session.setAttribute("user", user);
session.setAttribute("gameManager", gameManager);
session.setAttribute("gameManager", gameManager);
session.setAttribute("userConnection", userConnection). It is essential fo authentication.jsp 34
session.setAttribute("userConnection", userConnection). It is essential for home.jsp 29
set in the session with session.setAttribute("user", user). If home.jsp 34

SessionManagement.java webapp/src/main/java/online/caltuli/webapp/filter
61     userConnection = userManager.logUserConnection(ipAddress,
62         logger.info("userConnection.getId() = " + userConnection.get
63         session.setAttribute( s: "userConnection", userConnection);
64     } catch (BusinessException e) {
65         logger.info("can't register information related to the user
66         throw e;
67     }
68 }
69 return session;
70 }
```

Gestion des comptes utilisateur

Ajouts d'attributs UserConnection et User dans un HttpSession

User représente
un utilisateur
inscrit

The screenshot shows a code editor with several tabs open. The active tab is 'SessionManagement.java' located at 'webapp/src/main/java/online/caltuli/webapp/filter'. The code contains two calls to `session.setAttribute`: one for 'userConnection' and one for 'user'. Both lines are highlighted with a red box. A tooltip or comment above the second line reads: 'session.setAttribute("user", user); It is essential for authentication.jsp 34'. Another tooltip below it says: 'session.setAttribute("userConnection", userConnection). It is essential for home.jsp 29 set in the session with session.setAttribute("user", user). If'. Other tabs visible include 'Authentication.java' at line 48, 'Home.java' at line 195, and another 'Home.java' at line 219.

```
session.setAttribute("userConnection", userConnection); SessionManagement.java 63
session.setAttribute("user", user); Authentication.java 48
session.setAttribute("gameManager", gameManager); Home.java 195
session.setAttribute("gameManager", gameManager); Home.java 219
session.setAttribute("userConnection", userConnection). It is essential fo authentication.jsp 34
session.setAttribute("userConnection", userConnection). It is essential for home.jsp 29
set in the session with session.setAttribute("user", user). If home.jsp 34

SessionManagement.java webapp/src/main/java/online/caltuli/webapp/filter
61     userConnection = userManager.logUserConnection(ipAddress,
62     logger.info("userConnection.getId() = " + userConnection.get
63     session.setAttribute( s: "userConnection", userConnection);
64 } catch (BusinessException e) {
65     logger.info("can't register information related to the user
66     throw e;
67 }
68 }
69 return session;
70 }
```

Gestion des comptes utilisateur

Ajouts d'attributs UserConnection et User dans un HttpSession

Ces deux stockages sont initialisés dans les classes SessionManagement et Authentication

The screenshot shows a code editor with two tabs open: SessionManagement.java and Authentication.java.

SessionManagement.java:

```
session.setAttribute("userConnection", userConnection);
session.setAttribute("user", user);
```

Authentication.java:

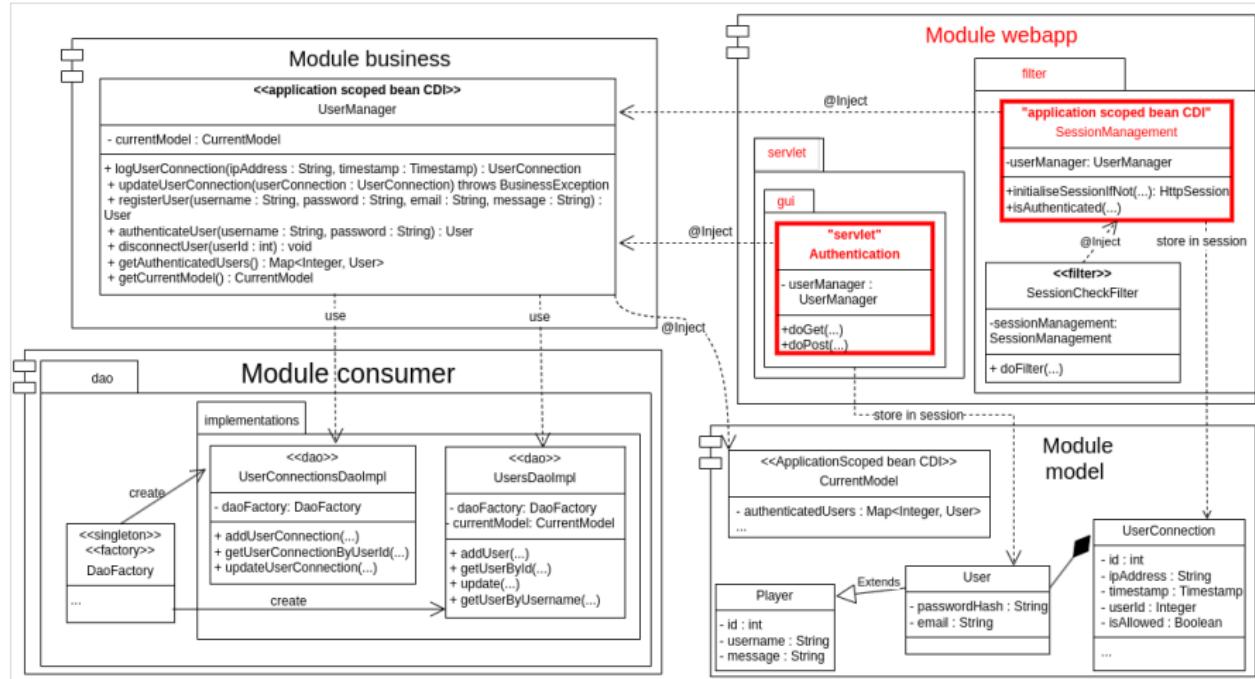
```
session.setAttribute("userConnection", userConnection); It is essential fo authentication.jsp 34
session.setAttribute("userConnection", userConnection). It is essential for home.jsp 29
set in the session with session.setAttribute("user", user). If home.jsp 34
```

SessionManagement.java (continued):

```
userConnection = userManager.logUserConnection(ipAddress,
logger.info("userConnection.getId() = " + userConnection.getId());
session.setAttribute("userConnection", userConnection);
} catch (BusinessException e) {
    logger.info("can't register information related to the user");
    throw e;
}
return session;
```

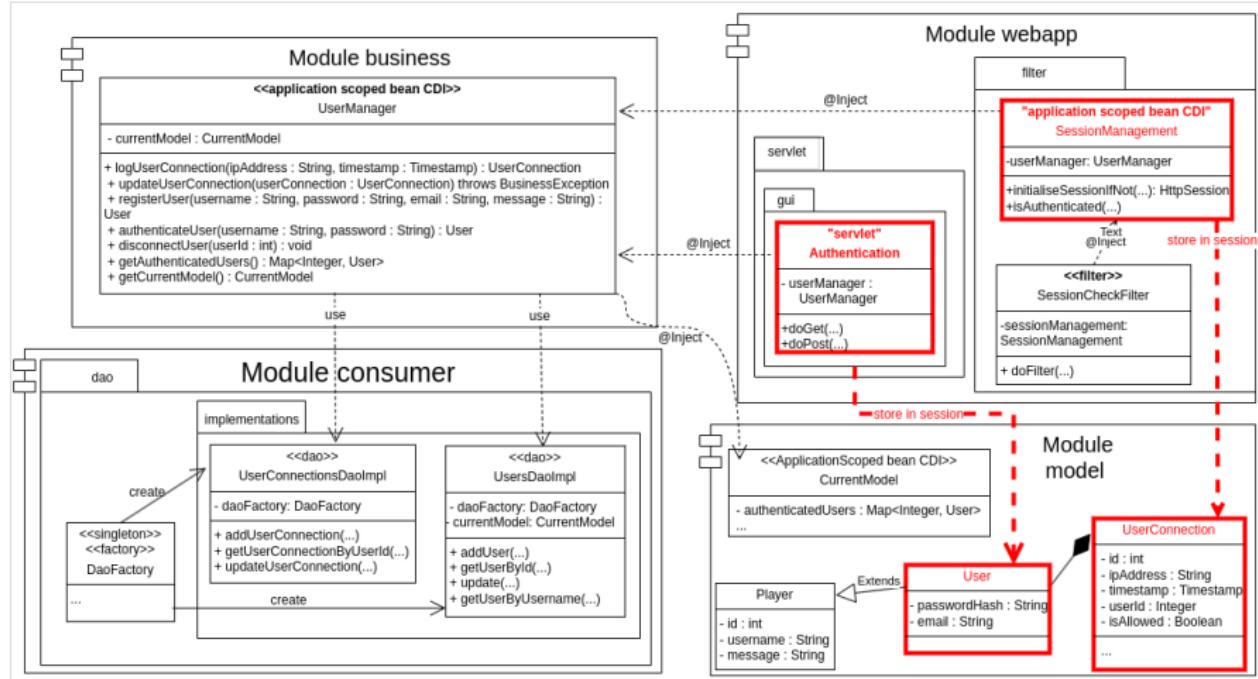
Gestion des comptes utilisateur

SessionManagement et Authentication stockent dans la session



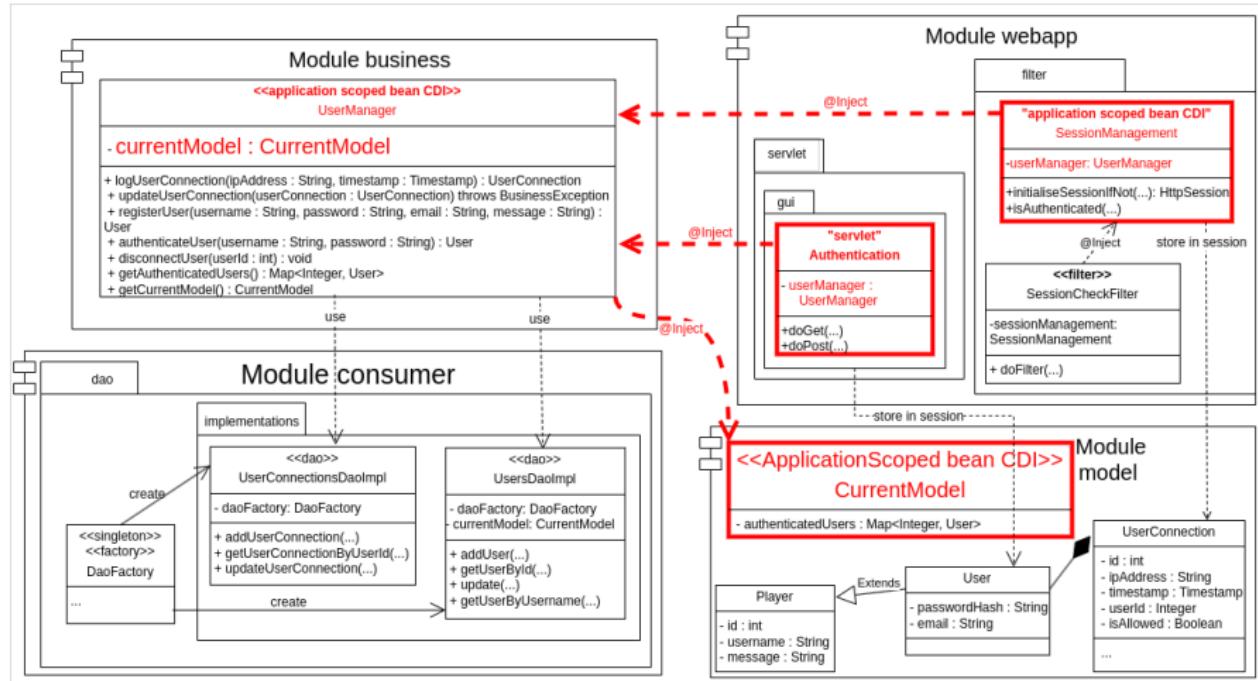
Gestion des comptes utilisateur

SessionManagement et Authentication stockent dans la session



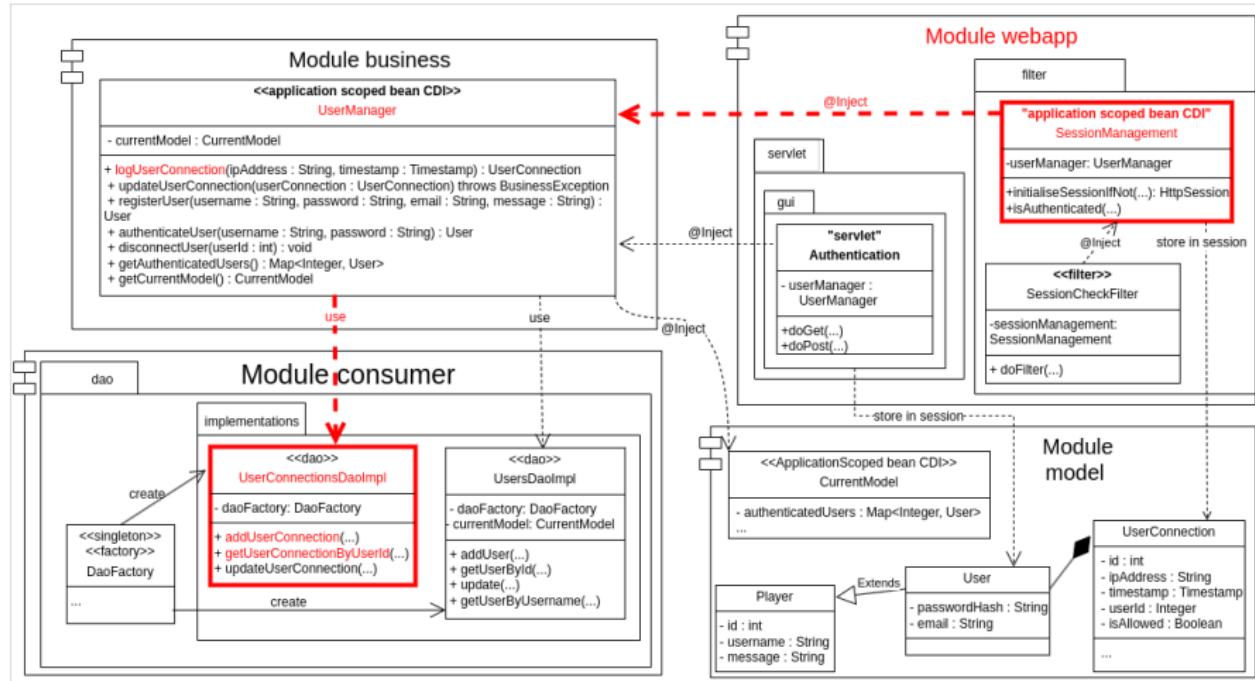
Gestion des comptes utilisateur

SessionManagement et Authentication contrôlent CurrentModel.



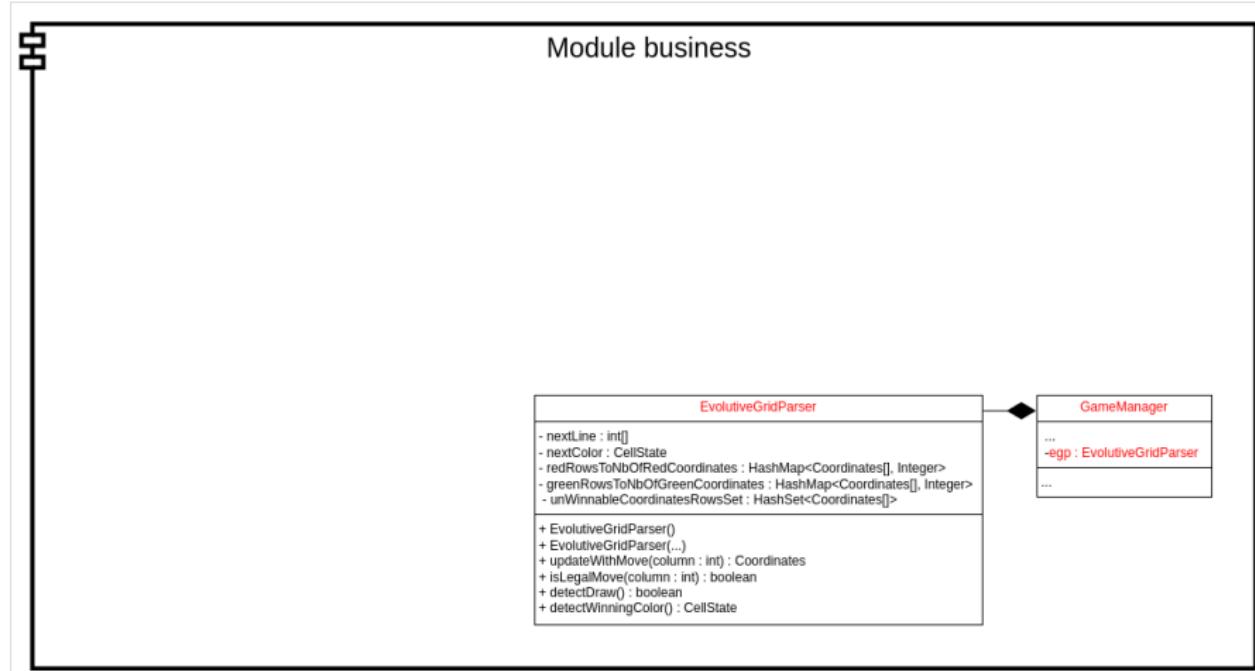
Gestion des comptes utilisateur

SessionManagement enregistre une connexion dans la BDD.



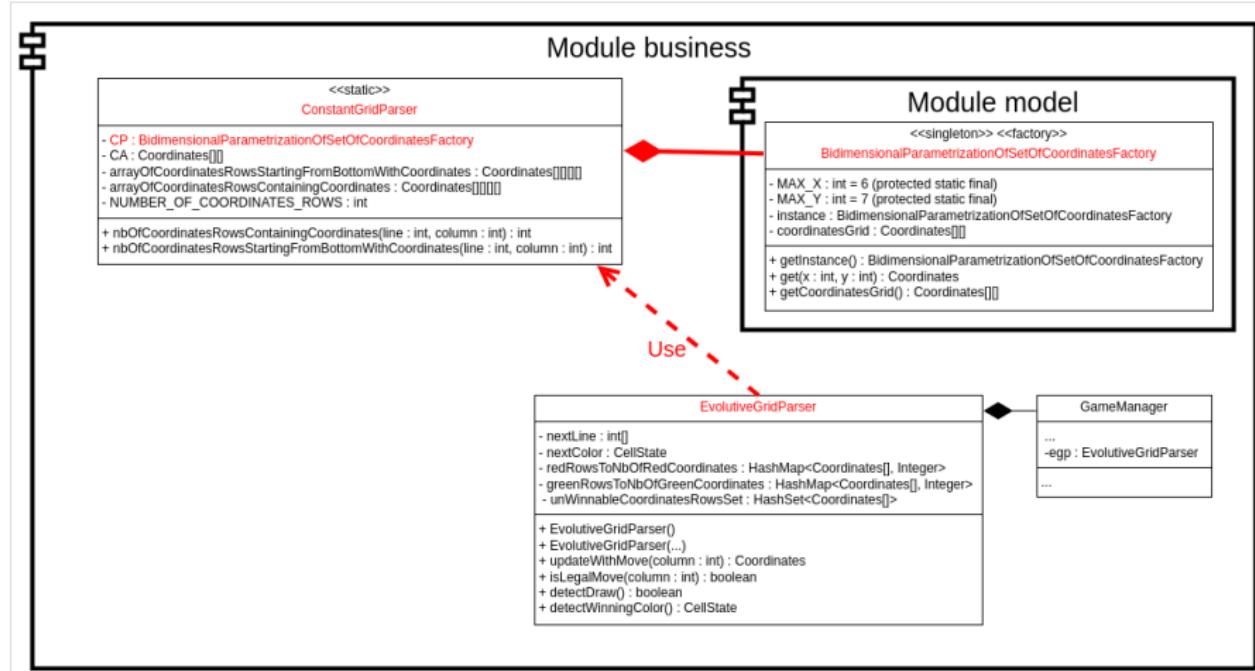
Implémentation d'un moteur de décision min-max

Le contrôleur de jeu agrège un parseur évolutif EvolutiveGridParser



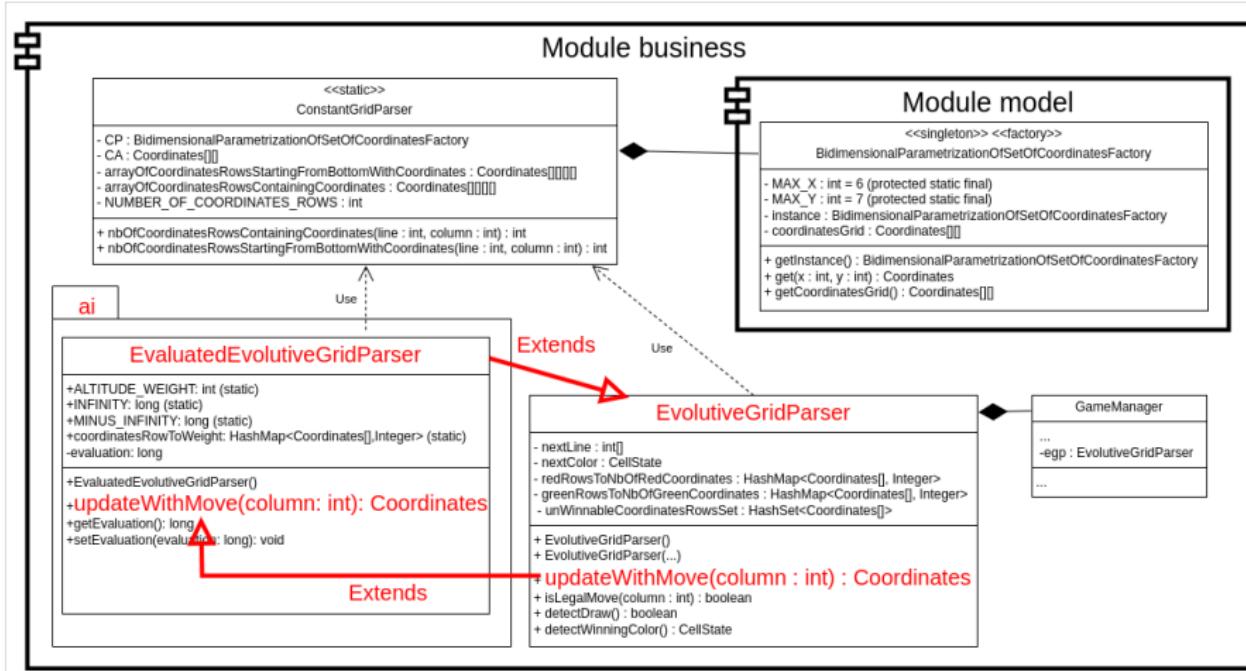
Implémentation d'un moteur de décision min-max

EvoluteGridParser utilise un parseur constant ConstantGridParser



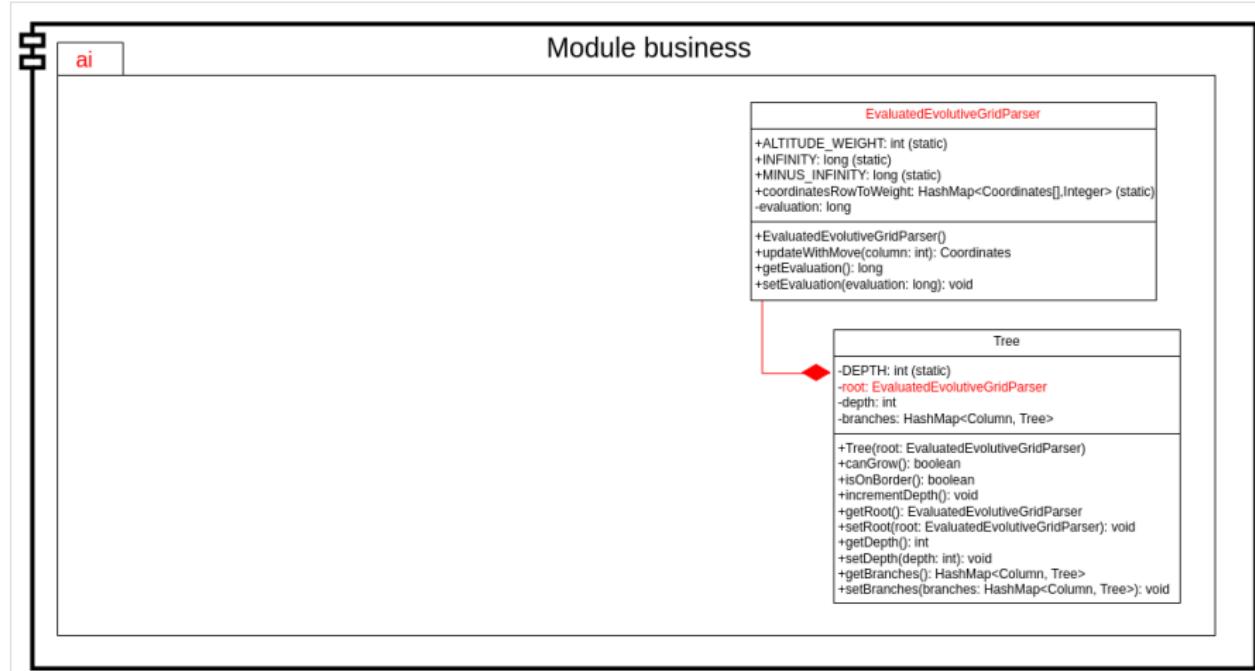
Implémentation d'un moteur de décision min-max

Extension du parseur évolutif avec surcharge pour évaluation



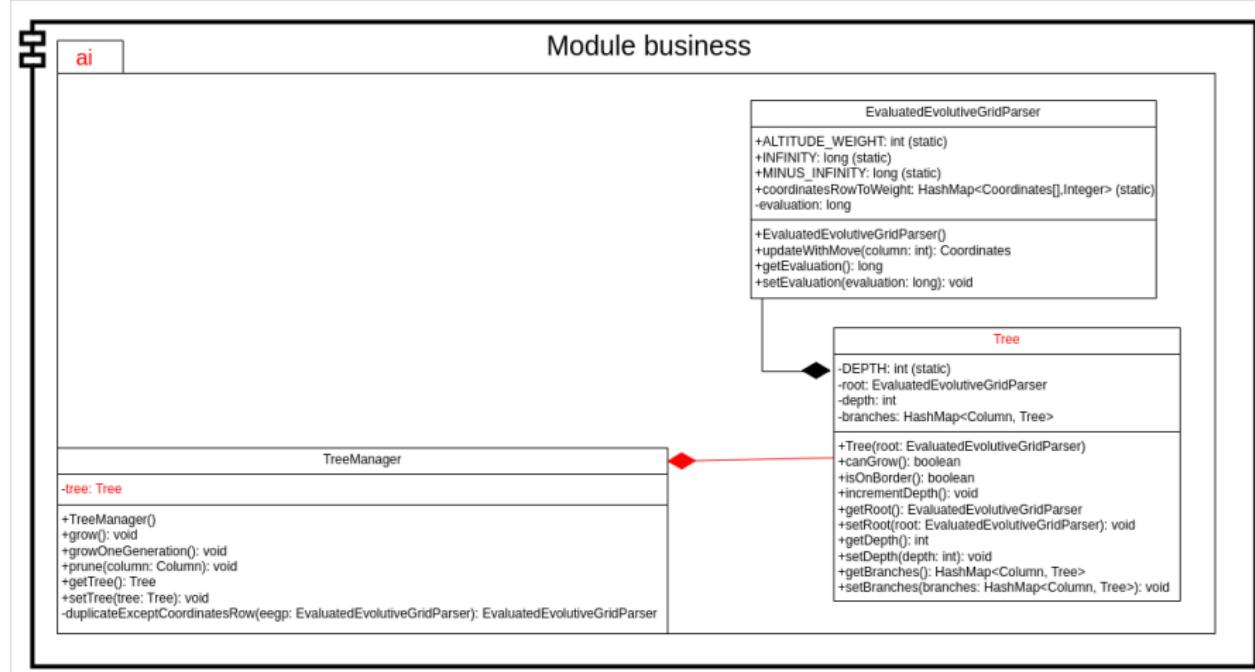
Implémentation d'un moteur de décision min-max

Les nœuds de l'arbre sont des parseurs évolutifs évalués



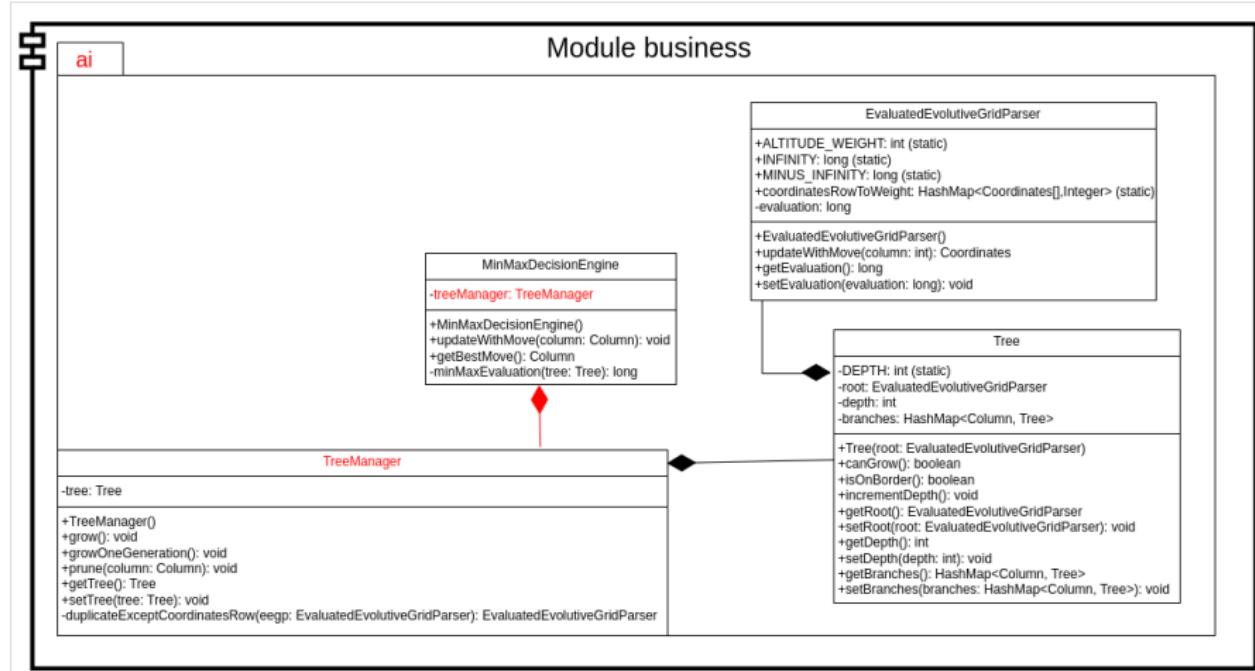
Implémentation d'un moteur de décision min-max

L'arbre s'agrège dans un gestionnaire pour croissance et élagage



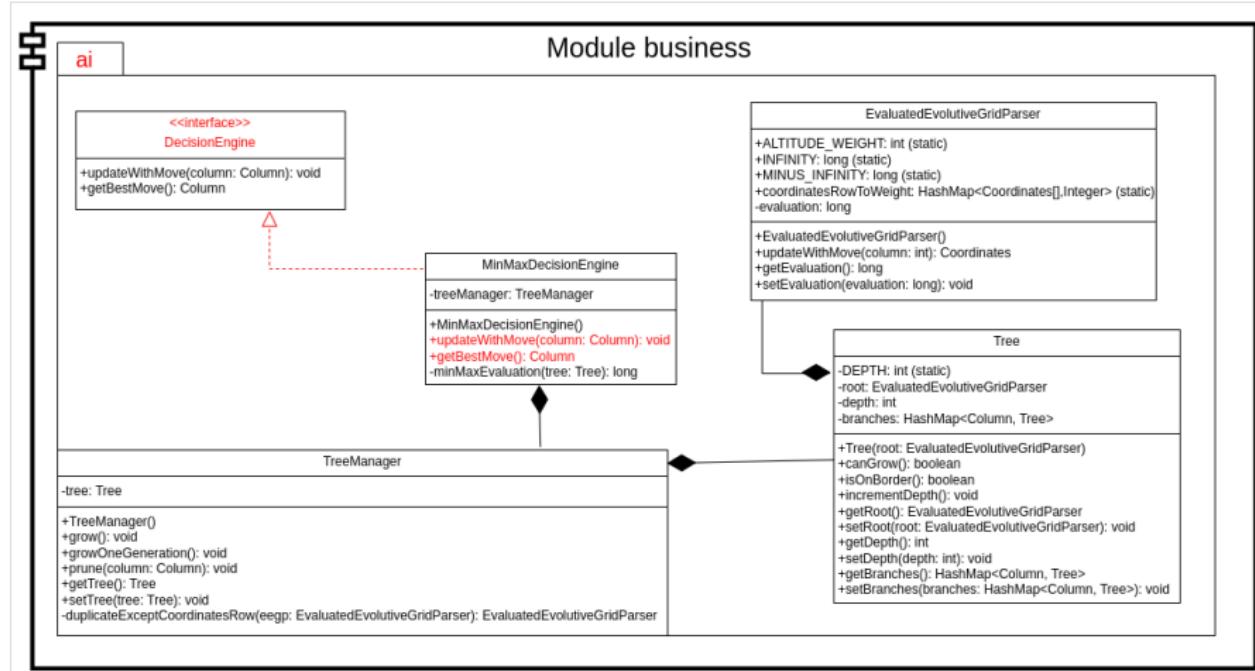
Implémentation d'un moteur de décision min-max

Le moteur de décision min-max agrège le gestionnaire d'arbre



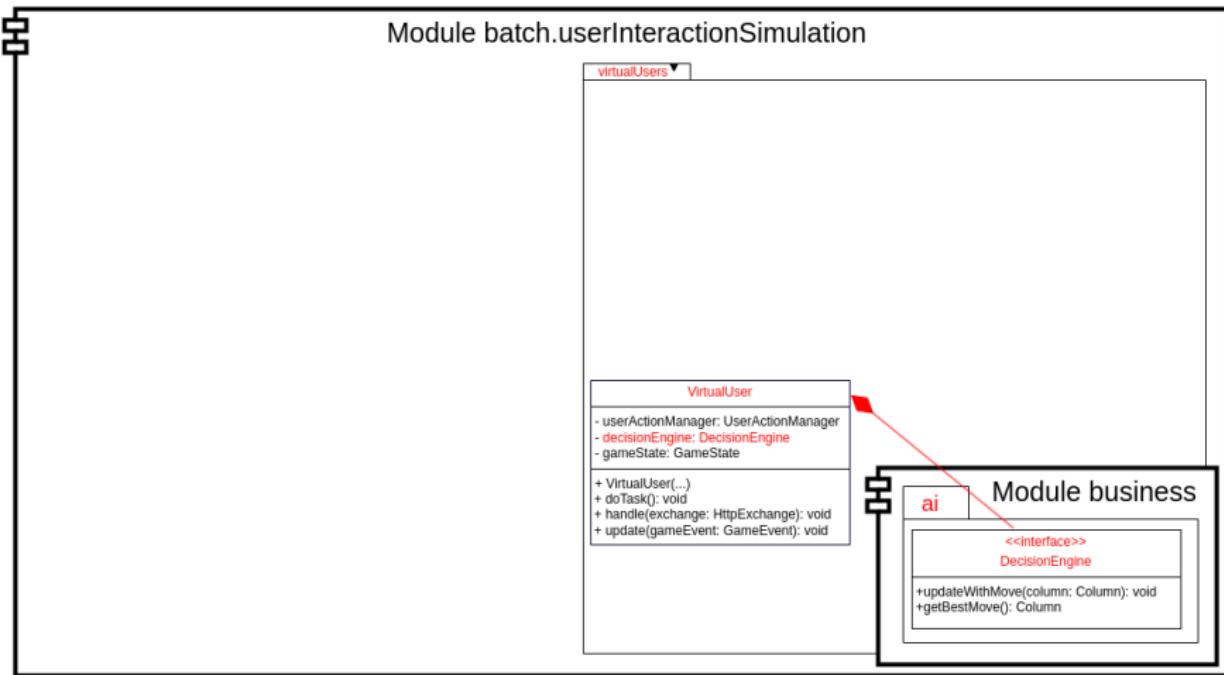
Implémentation d'un moteur de décision min-max

Le moteur implémente updateWithMove et getBestMove



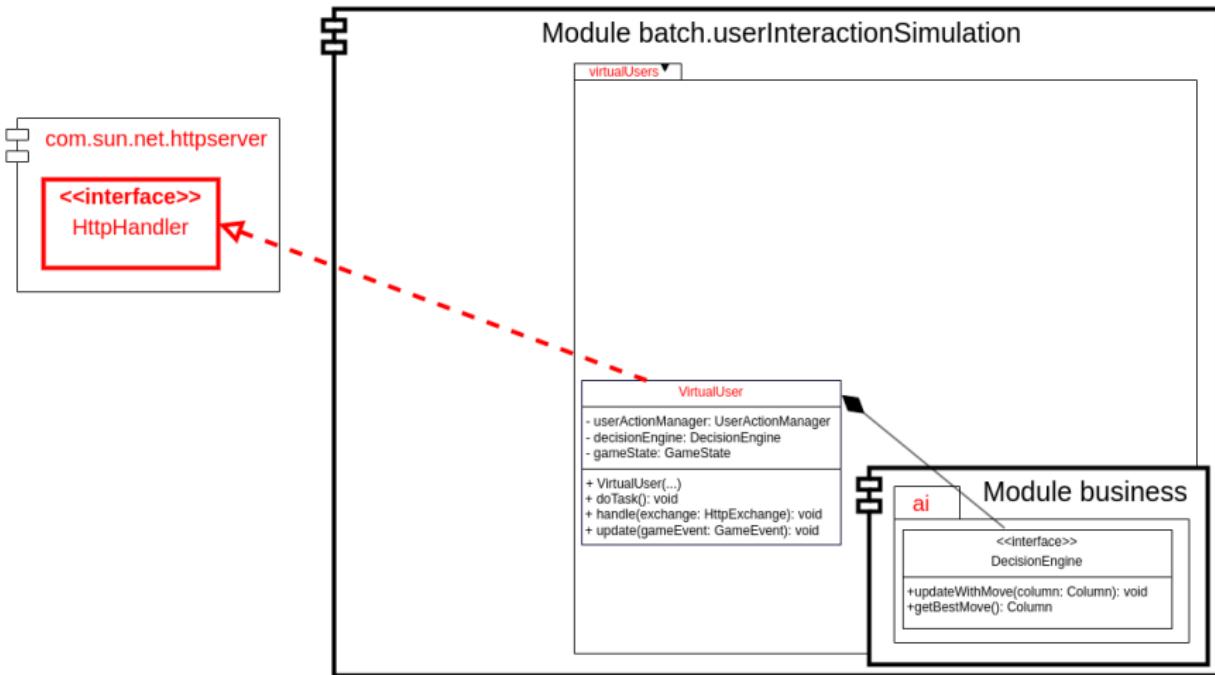
Implémentation d'un utilisateur virtuel intelligent

Le moteur de décision s'agrège dans l'utilisateur virtuel



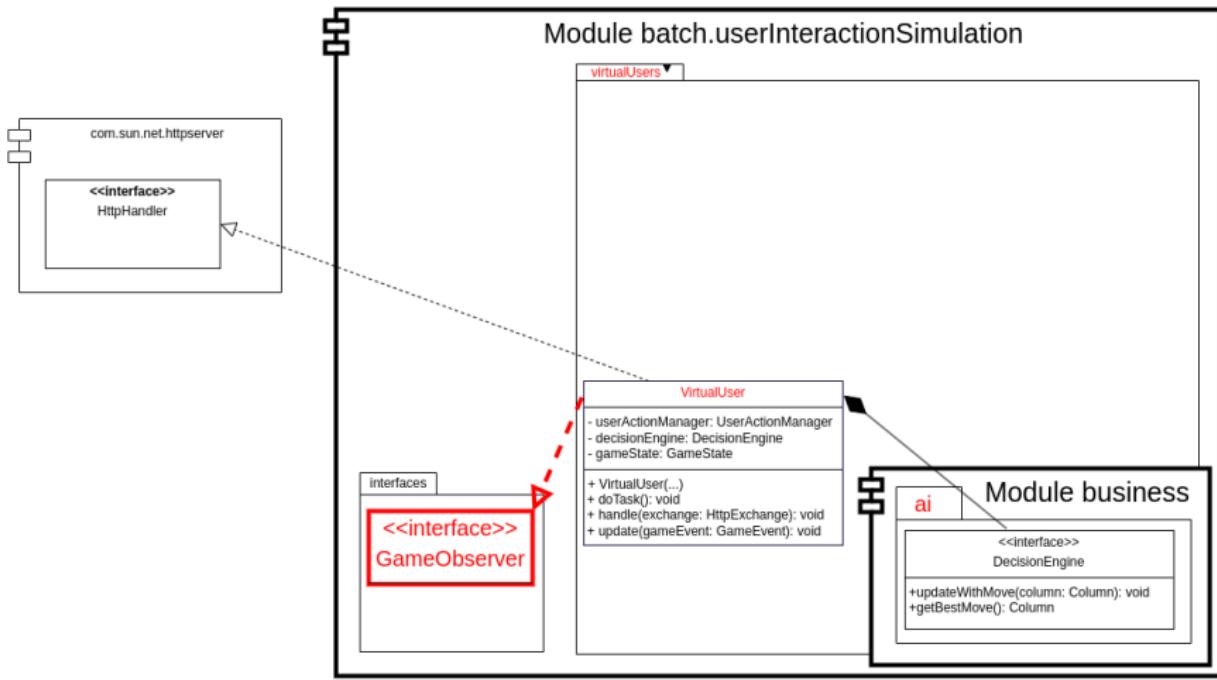
Implémentation d'un utilisateur virtuel intelligent

L'utilisateur virtuel est un gestionnaire de requête HTTP



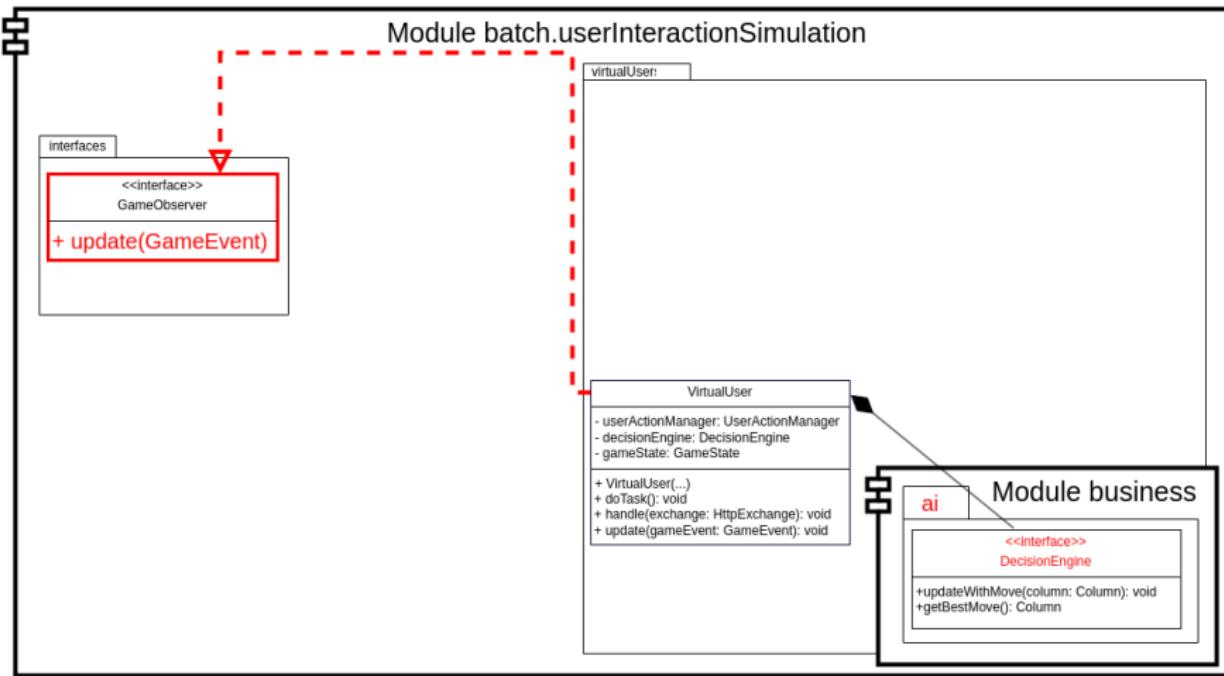
Implémentation d'un utilisateur virtuel intelligent

L'utilisateur virtuel est aussi un observateur de client Websocket



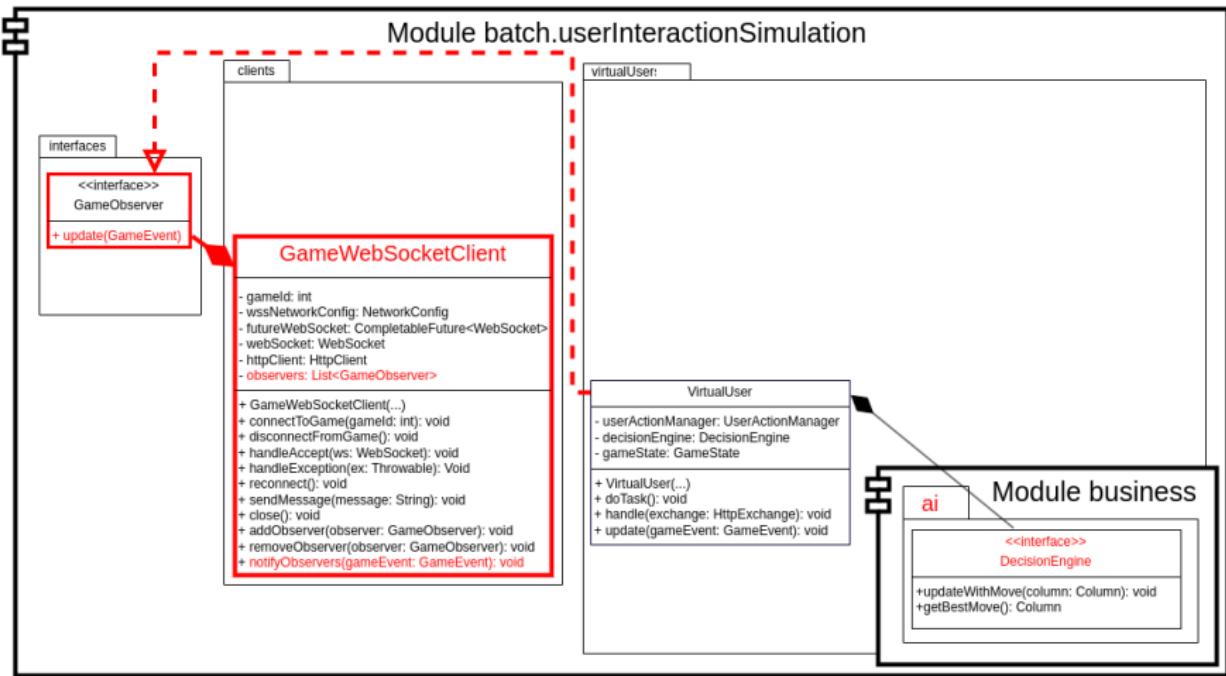
Implémentation d'un utilisateur virtuel intelligent

Un GameObserver implémente une méthode update appelée...



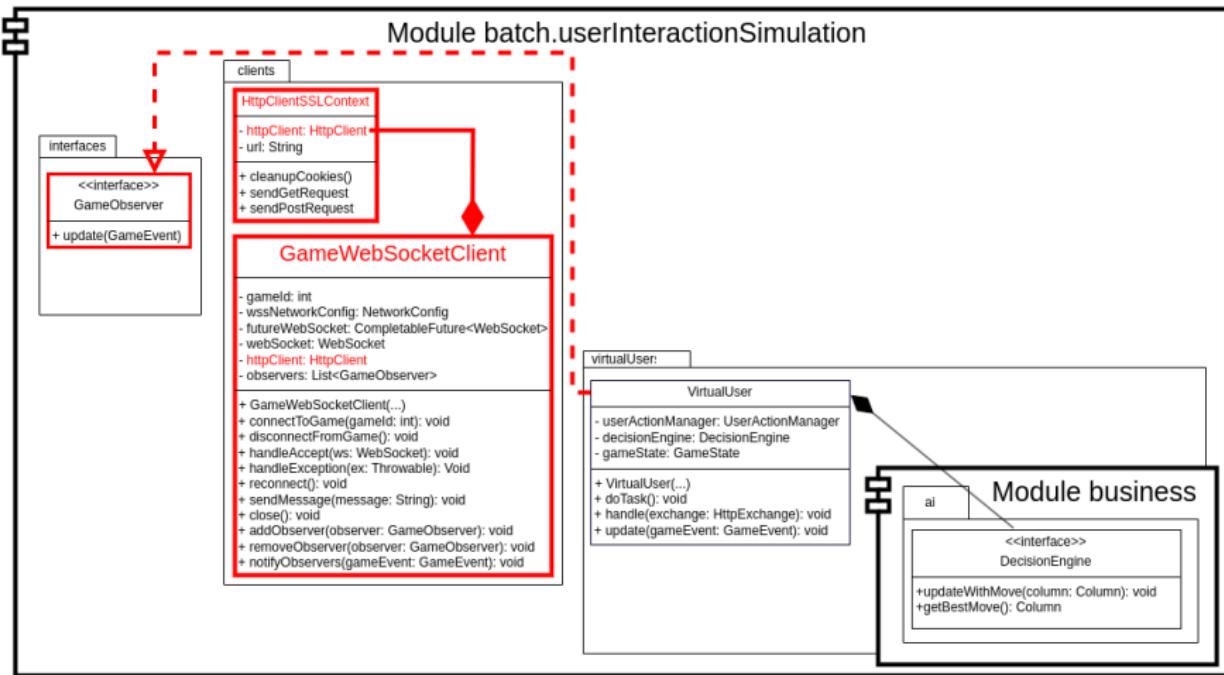
Implémentation d'un utilisateur virtuel intelligent

...par la méthode notifyObservers du client Websocket qui l'agrège



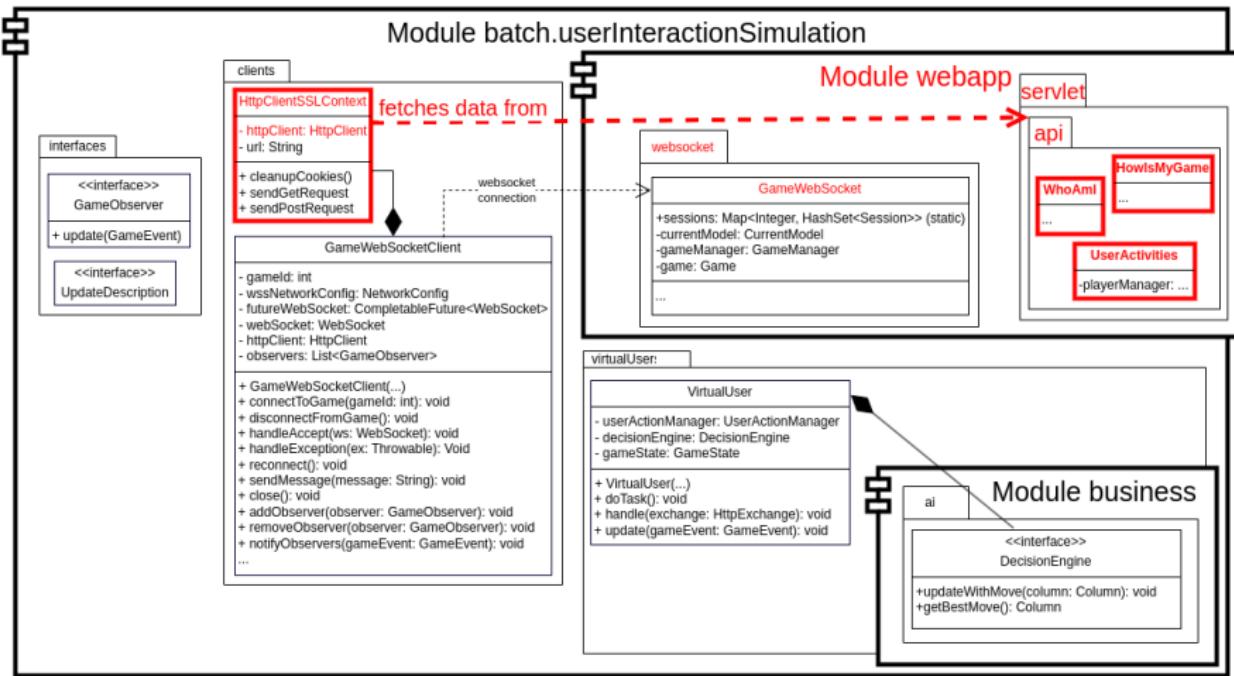
Implémentation d'un utilisateur virtuel intelligent

Le client Websocket agrège un client HTTP



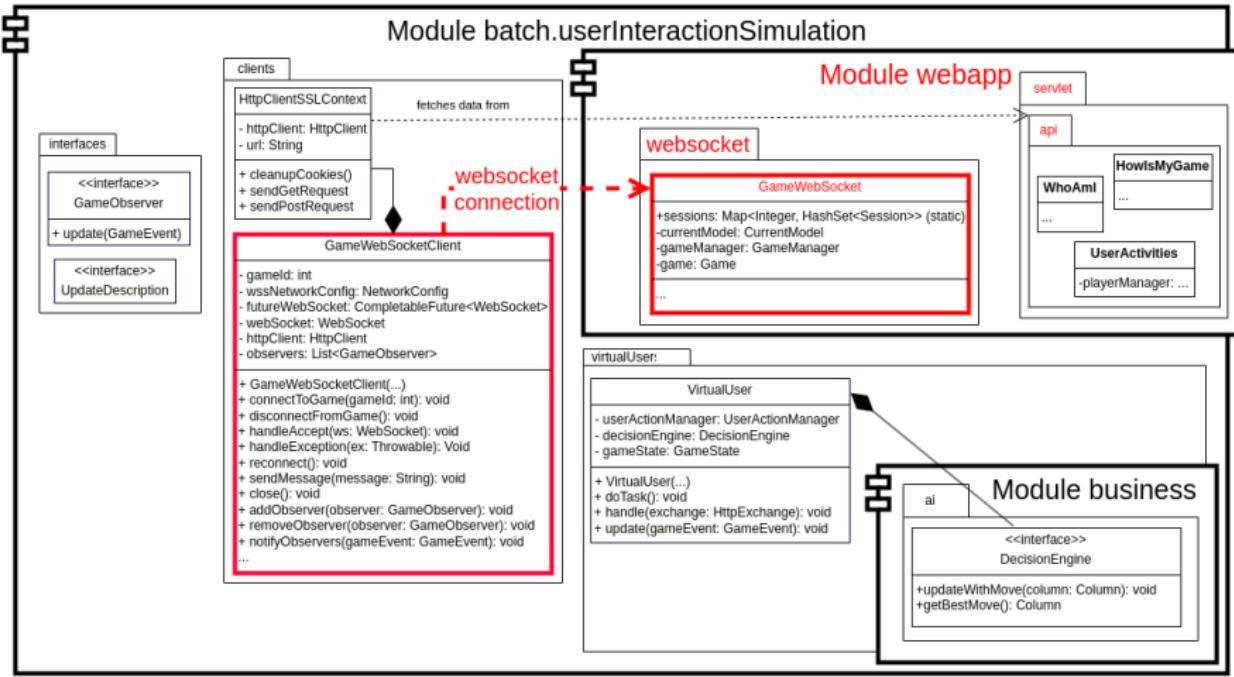
Implémentation d'un utilisateur virtuel intelligent

Le client HTTP récupère des données depuis l'API de l'application



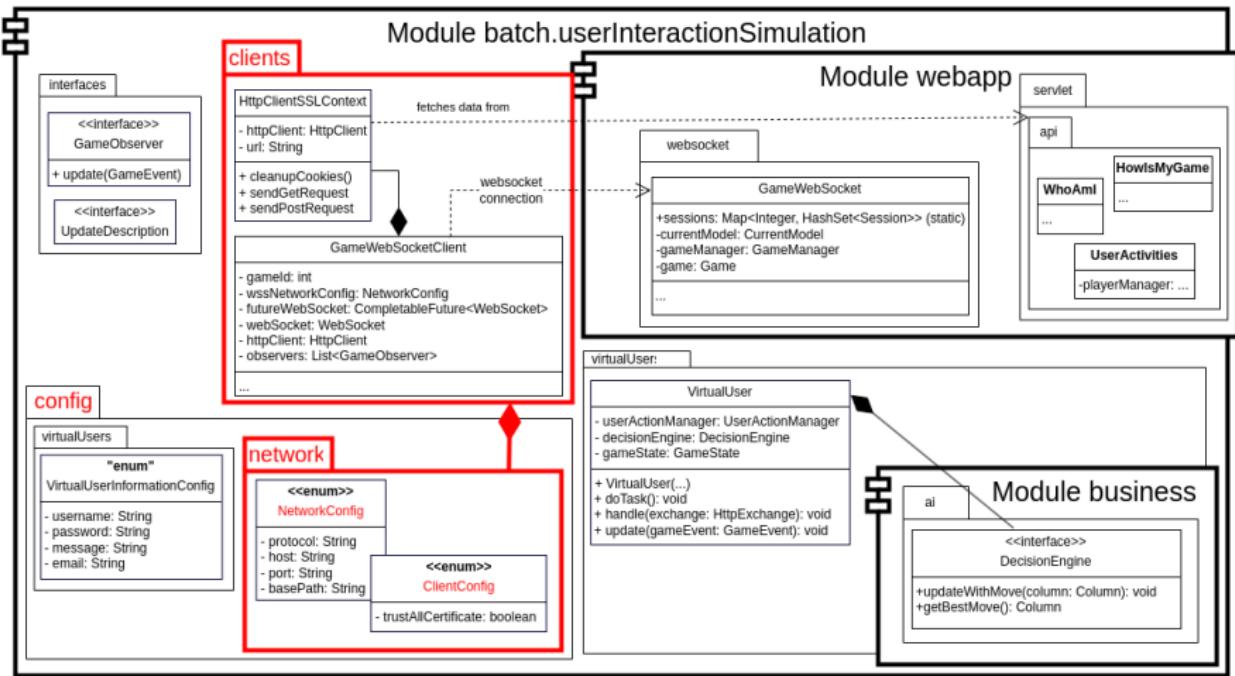
Implémentation d'un utilisateur virtuel intelligent

Le client websocket se connecte au serveur dans le protocole WSS



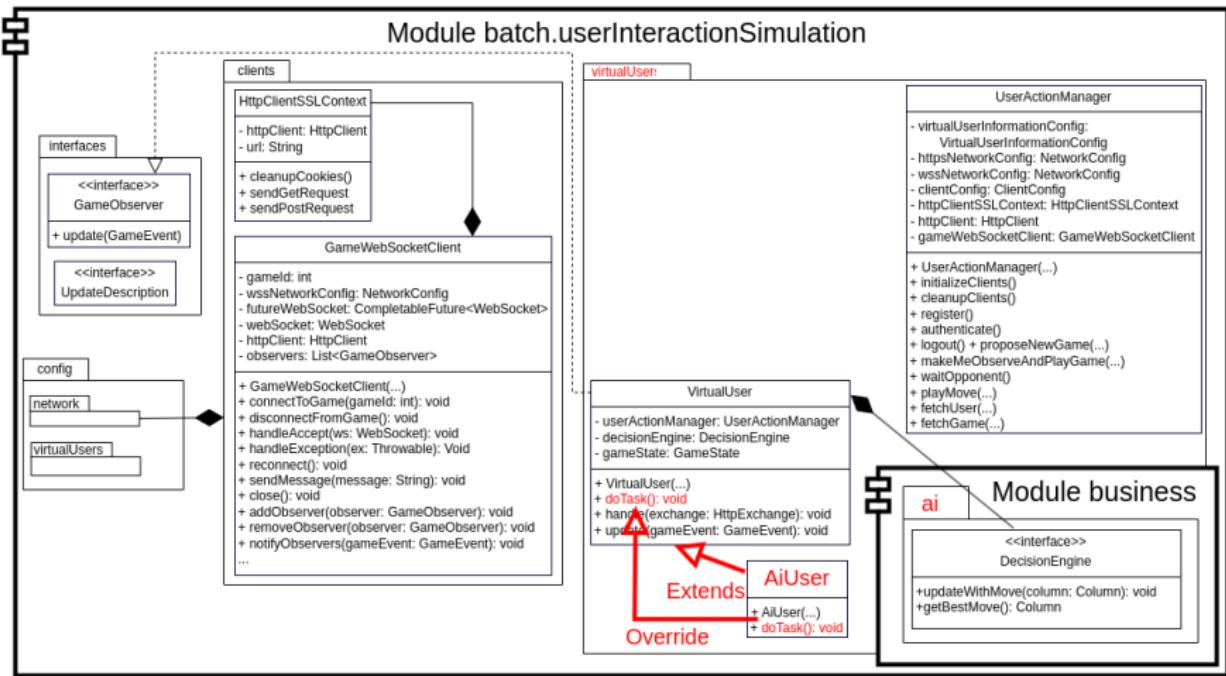
Implémentation d'un utilisateur virtuel intelligent

Le client agrège les configurations réseau du package config.network



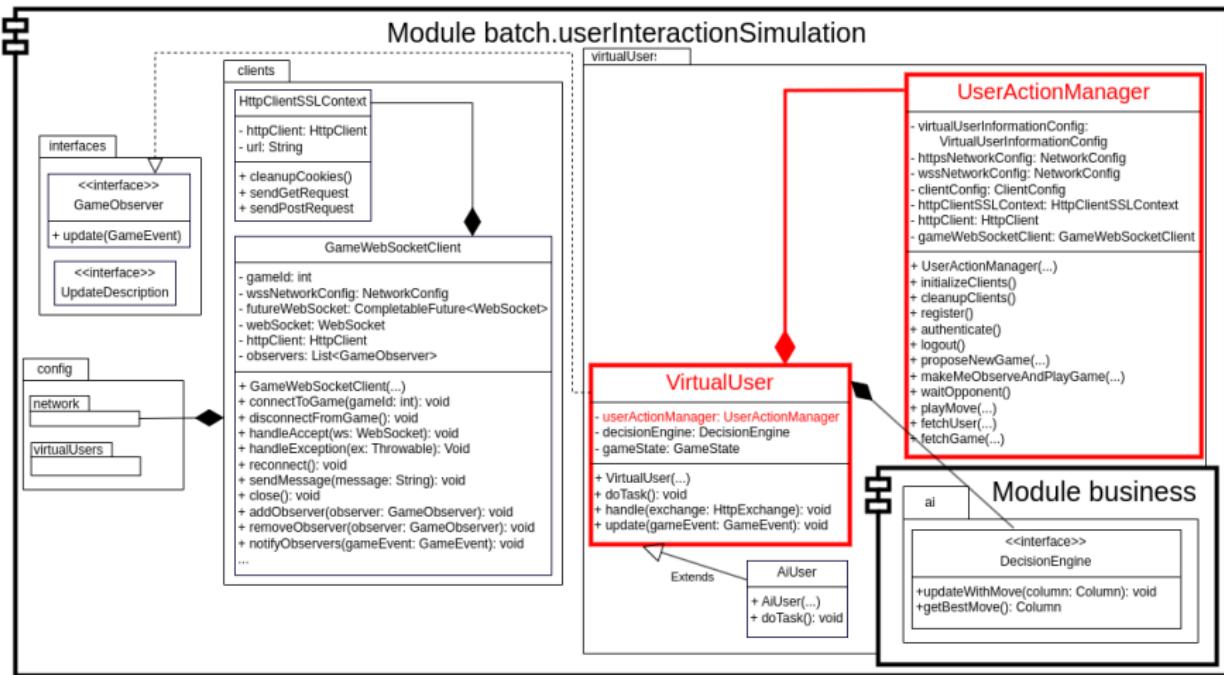
Implémentation d'un utilisateur virtuel intelligent

Extension de l'utilisateur virtuel avec surcharge pour les tâches



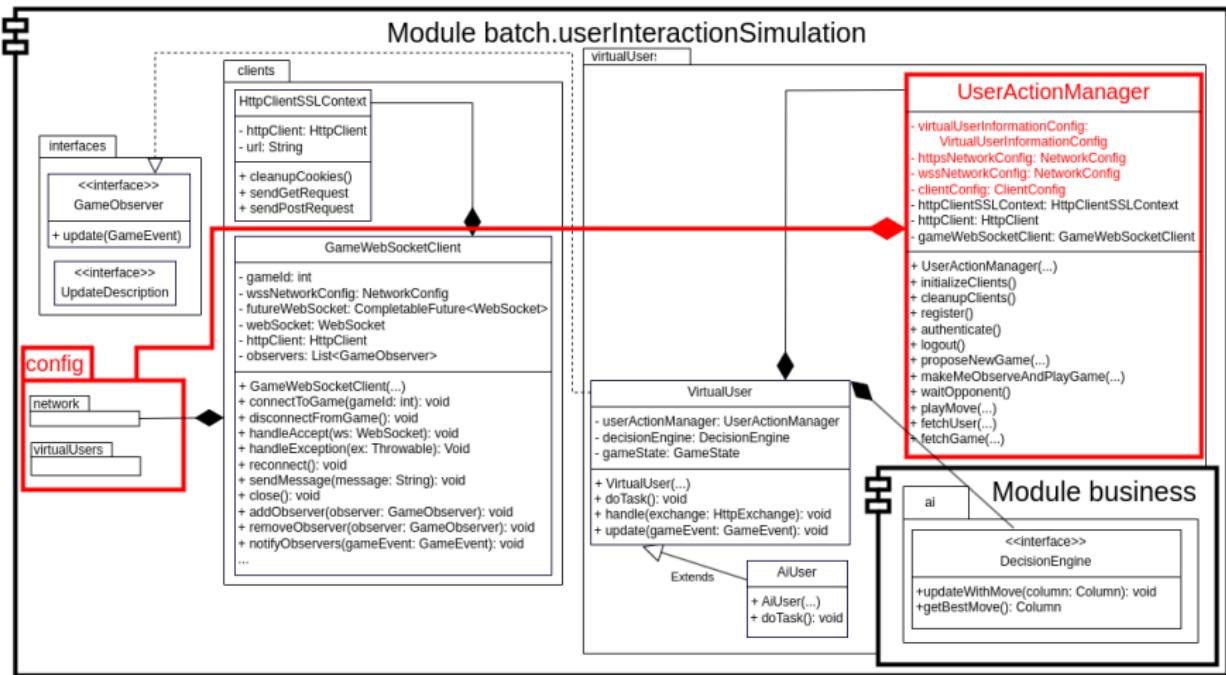
Implémentation d'un utilisateur virtuel intelligent

L'utilisateur virtuel agrège un gestionnaire de tâches



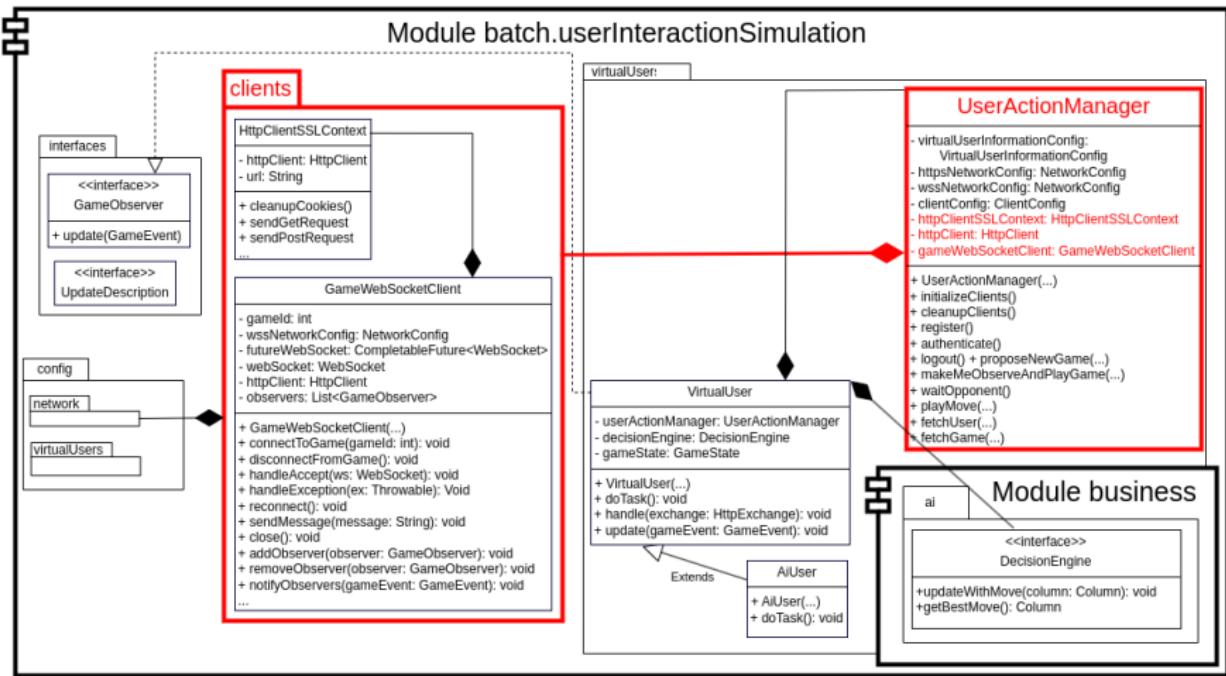
Implémentation d'un utilisateur virtuel intelligent

Le gestionnaire de tâches agrège les configurations



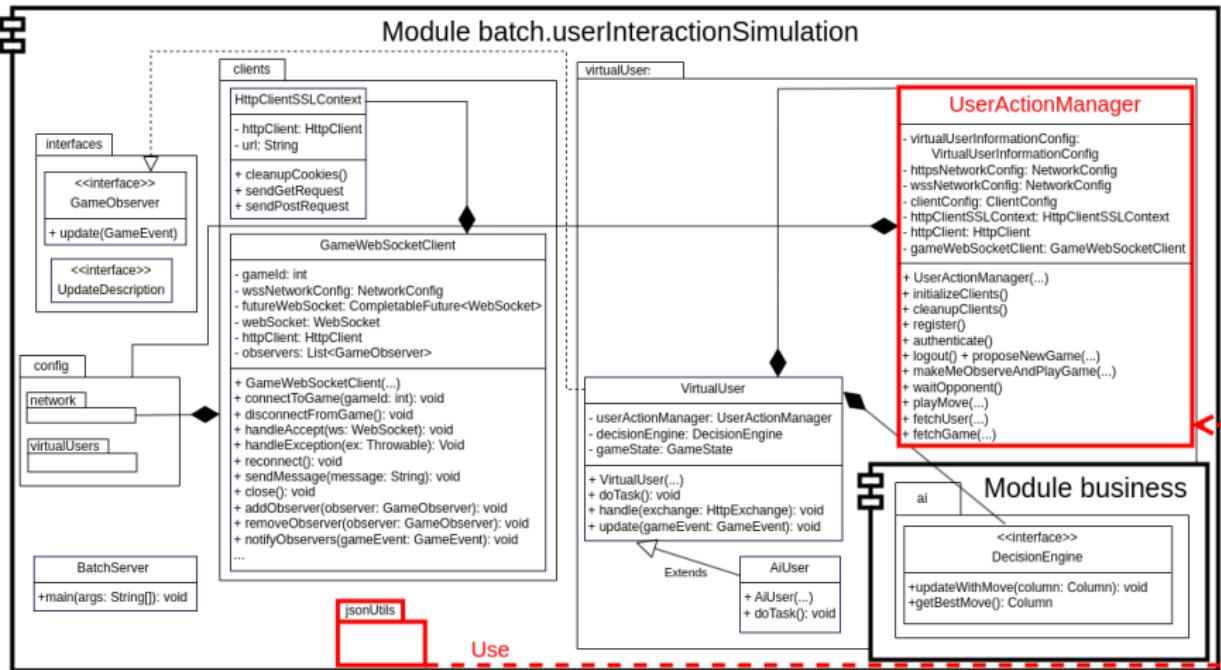
Implémentation d'un utilisateur virtuel intelligent

Le gestionnaire de tâches agrège les clients du package clients



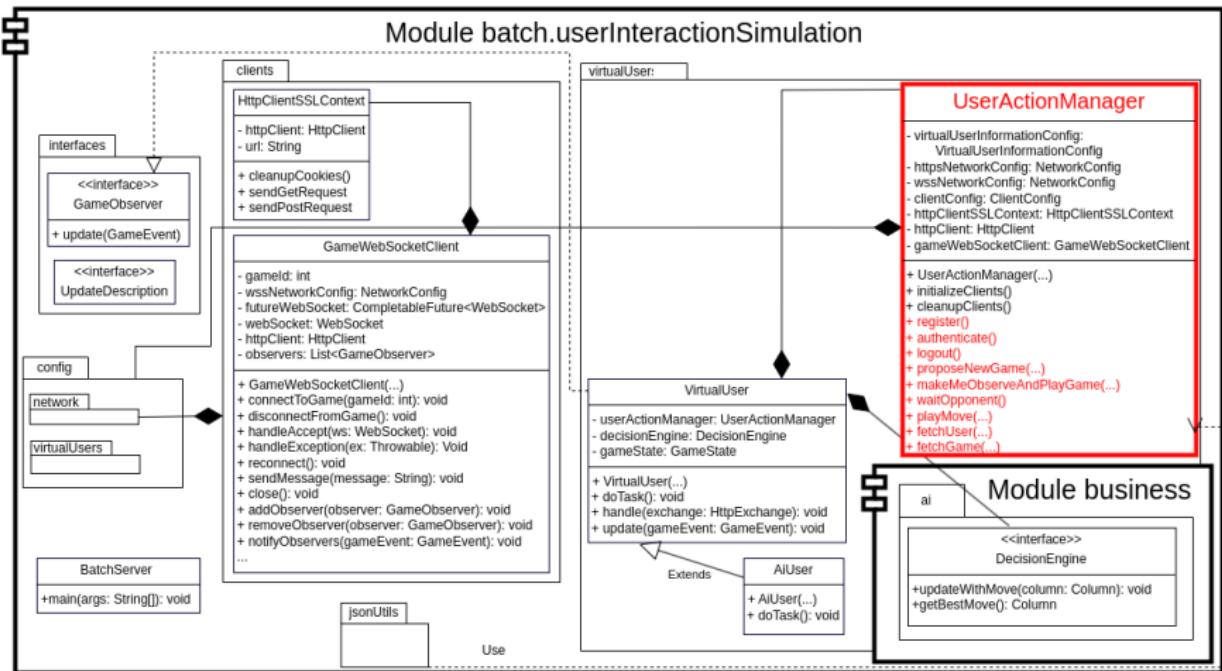
Implémentation d'un utilisateur virtuel intelligent

Le gestionnaire de tâches utilise sérialiseurs et dé-sérialiseurs



Implémentation d'un utilisateur virtuel intelligent

Le gestionnaire de tâches simule toutes les utilisations des services



Trois leçons à en tirer :

- Préciser davantage la conception avant de commencer l'implémentation.

Trois leçons à en tirer :

- ➡ Préciser davantage la conception avant de commencer l'implémentation.
- ➡ Utiliser TDD et tâcher d'obtenir une pyramide des tests et non un choux-fleur débordant de tests fonctionnels chronovores.

Trois leçons à en tirer :

- Préciser davantage la conception avant de commencer l'implémentation.
- Utiliser TDD et tâcher d'obtenir une pyramide des tests et non un choux-fleur débordant de tests fonctionnels chronovores.
- Favoriser l'abstraction et la modularité en dépendant d'interfaces plutôt que d'implementations concrètes pour faciliter la maintenance et l'évolutivité du code.

Trois leçons à en tirer :

- Préciser davantage la conception avant de commencer l'implémentation.
- Utiliser TDD et tâcher d'obtenir une pyramide des tests et non un choux-fleur débordant de tests fonctionnels chronovores.
- Favoriser l'abstraction et la modularité en dépendant d'interfaces plutôt que d'implementations concrètes pour faciliter la maintenance et l'évolutivité du code.

Merci de votre attention ! Questions ?