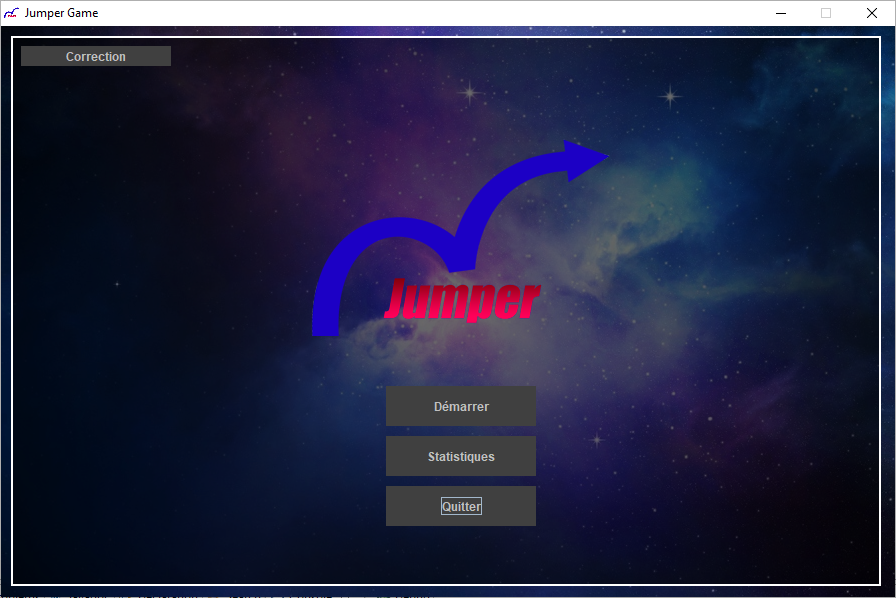
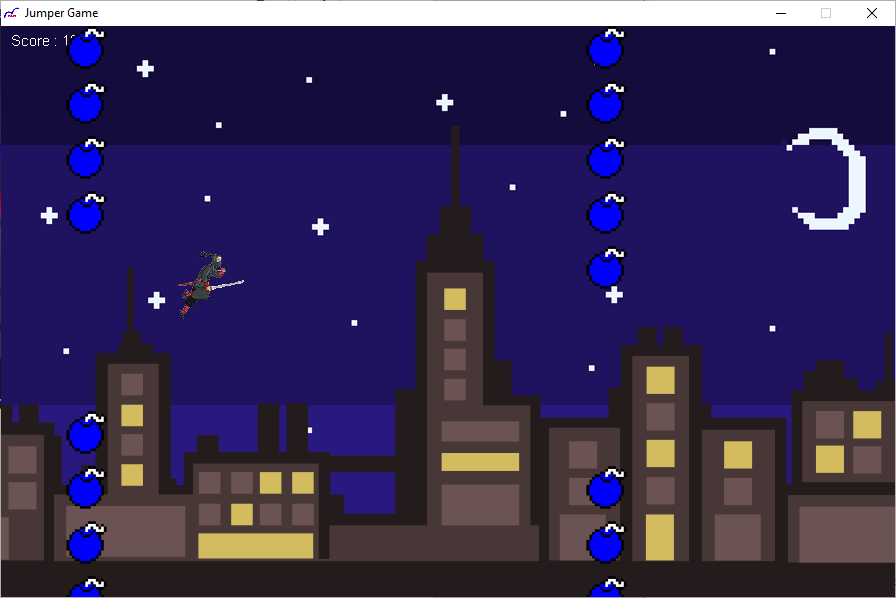
# Le jeu Jumper





Ce jeu consiste à contrôler un ninja qui doit éviter des obstacles. Les déplacements se font par les touches : A, S, W et D.

Lorsque le ninja touche une bombe, le jeu se termine. À ce moment, le nom du joueur et son score doivent être sauvegardés dans une base de données MySQL.

## Notes

* Pour réaliser ce projet, je vous recommande FORTEMENT de suivre les étapes en ordre.
* IntelliJ a été utilisé pour créer le programme, vous devriez donc l’utiliser pour compléter ce travail.
* Si vous avez des questions, n’hésitez pas à me demander de l’aide.
* Vous devriez conserver un fichier, jumper.sql, qui contiendra vos énoncés SQL.
* N’oubliez pas de tester chaque fonctionnalité développée afin de s’assurer qu’elle fonctionne correctement.
* **IMPORTANT** : Ne modifiez en aucun cas la classe « CorClient.java »!

## Exemples d’accès à MySQL dans le programme

### Exemple 1 : Lire une listes d’items

Pour ces deux **exemples**, prenez pour acquis que userId est défini ainsi que la classe Item.

List<Item> items = new ArrayList<Item>();

Connection connection = DBConnection.getConnection();

try {

PreparedStatement statement = connection.prepareStatement(

"SELECT \* FROM items WHERE id\_user = ? ");

statement.setInt(1, userId);

ResultSet result = statement.executeQuery();

while(result.next()) {

items.add(new Item(result.getInt("id"), result.getString("name")));

}

statement.close();

} catch (SQLException e) {

e.printStackTrace();

}

return items;

### Exemple 2 : Supprimer un item

boolean success = false;

Connection connection = DBConnection.getConnection();

PreparedStatement statement;

try {

statement = connection.prepareStatement("DELETE FROM items WHERE id = ?");

statement.setInt(1, id);

statement.execute();

statement.close();

success = true;

} catch (SQLException e) {

e.printStackTrace();

}

return success;

## Les tâches à réaliser

Ces tâches doivent être réalisées en ordre, autrement il sera difficile de tester que tout fonctionne.

### Tâche #0 : S’assurer que le TimeZone de MySQL est défini

Dans MySQL Workbench, écrire ceci :

SET GLOBAL time\_zone = '-5:00';

### Tâche #1 : Création de la base de données MySQL (dans MySQL Workbench)

Cette base de données MySQL contiendra les tables de l’application. Pour cette question, il s’agit de créer la base de données (create database …), pas les tables.

#### Notes/Énoncés SQL:

create database db\_jumper character set utf8 collate utf8\_general\_ci;

### Tâche #2 : Avoir une connexion à la base de données valide (dans MySQL Workbench)

Afin d’avoir accès à cette nouvelle base de données, il est nécessaire de se créer un usager MySQL et d’y donner les droits nécessaires.

#### Notes/Énoncés SQL:

create user jumper\_user@'localhost' identified by 'AAAaaa111';

grant all on db\_jumper.\* to jumper\_user@'localhost';

### Tâche #3 : Spécifier les informations de connexion à la base de données MySQL

Pour que le programme puisse avoir accès à la base de données, il faut spécifier les informations de connexion nécessaires. Pour ce faire, allez dans la classe dao/DBConnexion.java et modifiez les informations qui se trouvent dans la méthode « getConnection() ».

Si, lorsque vous exécutez le programme Java, une erreur qui est affichée, c’est parce que les informations de connexion sont erronées.

#### Notes:

public class DBConnection {

private static Connection connection;

/\*\*

\* M�thode qui permet de retourner une connexion � la base de donn�es MySQL

\*

\* @return

\*/

public static Connection getConnection() {

if (connection == null) {

MysqlXADataSource dataSource = new MysqlXADataSource();

dataSource.setUser("jumper\_user");

dataSource.setPassword("AAAaaa111");

dataSource.setServerName("localhost");

dataSource.setDatabaseName("db\_jumper");

try {

connection = dataSource.getConnection();

} catch (SQLException e) {

e.printStackTrace();

}

}

return connection;

}

/\*\*

\* M�thode permettant de tester la connexion

\*

\* @return si la connexion est ouverte ou pas

\*/

public static boolean connectionCredentialsValid() {

Connection c = getConnection();

boolean valid = c != null;

releaseConnection();

return valid;

}

public static void releaseConnection() {

if (connection != null) {

try {

connection.close();

connection = null;

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

### Tâche #4 : Compléter la sauvegarde des parties

Afin de permettre la persistance des parties jouées, il s’agit de créer une table « game\_logs » et de **compléter** la méthode *GameLogDAO.save().*

Les informations qui doivent être sauvegardées sont :

* id : clé primaire de la table
* player\_name : chaîne de caractères de 40 caractères maximum
* score : entier (int)

Dans la méthode *GameDAO.save()*, un booléen doit être retourné. Si la sauvegarde dans la base de données a bien fonctionné, la méthode retourne « true ». S’il y a eu une erreur lors de la sauvegarde, alors « false » doit être retournée.

#### Notes/Énoncés SQL:

## dans WorkBench

use db\_jumper;

create table game\_logs (

id int not null auto\_increment,

playername varchar(40),

score int,

primary key pk\_game\_logs(id)

) engine=InnoDB;

## dans IntelliJ

public static boolean save(GameLog gameLog) {

boolean success = false;

Connection connection = DBConnection.getConnection();

PreparedStatement statement;

try {

statement = connection.prepareStatement("INSERT INTO game\_logs (playername, score) values(?, ?);");

statement.setString(1, gameLog.getPlayerName());

statement.setInt(2, gameLog.getScore());

statement.execute();

statement.close();

success = true;

} catch (SQLException e) {

e.printStackTrace();

}

### Tâche #5 : Terminer la section « Statistiques »

À partir du menu principal, il est possible de consulter les « high scores » du jeu.

Afin que cette fonctionnalité fonctionne, il s’agit de compléter la méthode *GameDAO.getHighScores()*. N’oubliez pas de bien vérifier le(s) paramètre(s) de la méthode.

#### Notes/Énoncés SQL:

public static List<GameLog> getHighScores(int limit) {

List<GameLog> scoreList = new ArrayList<GameLog>();

Connection connection = DBConnection.getConnection();

try {

PreparedStatement statement = connection.prepareStatement(

"SELECT \* FROM game\_logs limit ?");

statement.setInt(1, limit);

ResultSet result = statement.executeQuery();

while(result.next()) {

scoreList.add(new GameLog(result.getString("playername"), result.getInt("score")));

}

statement.close();

} catch (SQLException e) {

e.printStackTrace();

}

### Tâche #6 : Créer les index aux bons endroits

Imaginez que le jeu est tellement populaire (oh yeah!) que le nombre de parties sauvegardées dans la base de données est immense. Quel(s) index devriez-vous créer afin de maximiser la performance des requêtes?

Créez le ou les index nécessaire dans MySQL Workbench

#### Notes/Énoncés SQL:

create index idx\_game\_logs\_score on game\_logs(score);

### Tâche #7 : Faire un backup de la base de données

En utilisant mysqldump.exe, faites une copie de votre base de données.

#### Notes/Énoncés SQL:

C:\Program Files\MySQL\MySQL Server 8.0\bin>.\mysqldump.exe -uroot -p db\_jumper > jumper\_backup.sql

### Tâche #8 : Tester le module de correction

En haut à gauche du jeu se trouve le bouton « correction », qui sera utilisé pour aider à la correction des projets à venir. Essayer ce module et avisez-moi. Je pourrai vous « corriger » à distance.

#### Notes/Énoncés SQL:

100%