ZAKARIA TALALI 11-06-2025

Traces des Projet

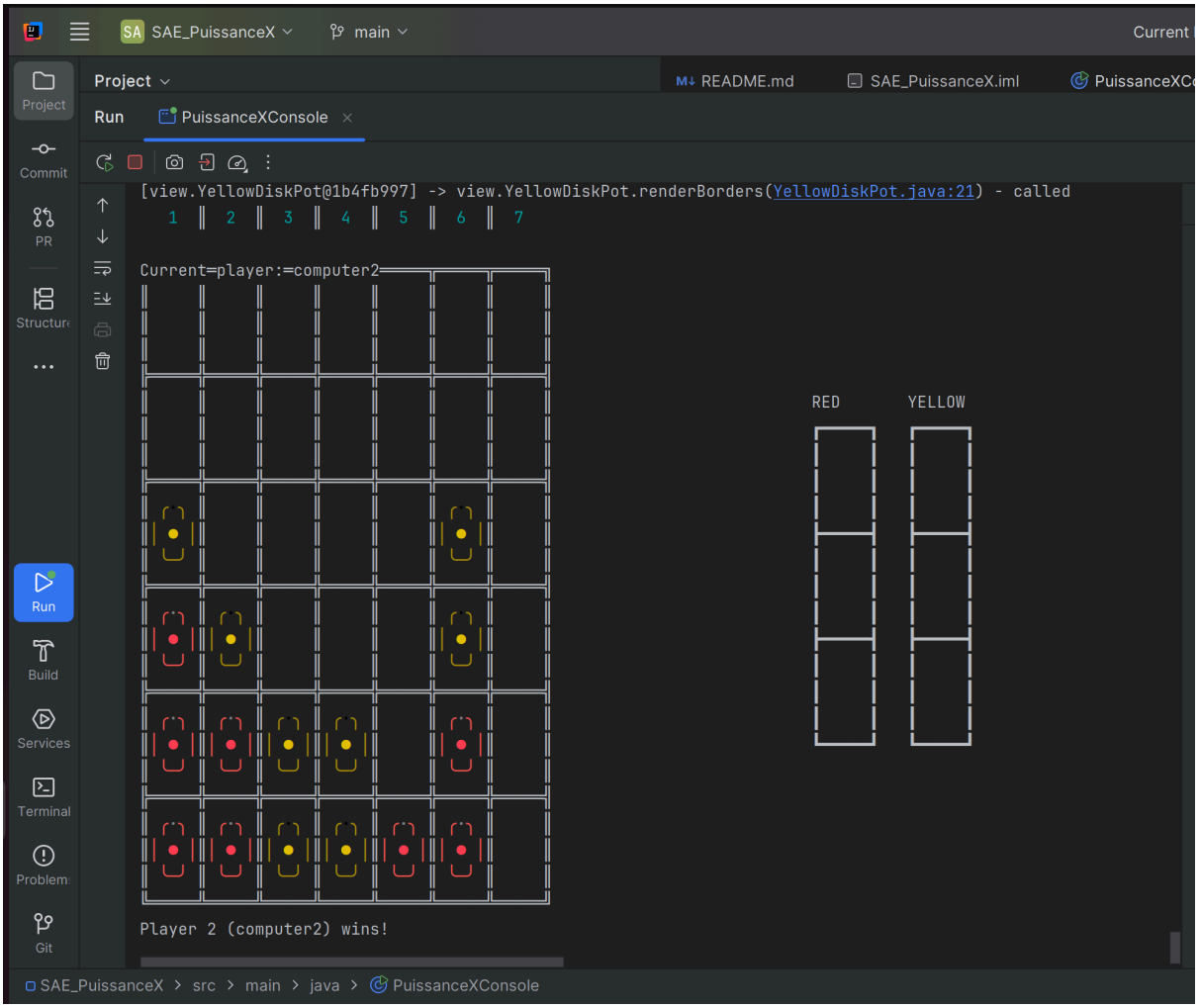

Figure 1 – Interface console de PuissanceX pendant une partie IA vs IA

La figure 1 présente une exécution typique de **PuissanceX**, une version avancée du jeu Puissance 4 développée en Java, en mode **console textuel**. La partie se joue entre deux IA, avec un affichage ASCII du plateau, des jetons placés, et de l'état de la partie. Le joueur 2 (ordinateur) gagne la partie, ce qui est indiqué en bas.

L'objectif de ce projet est double : **reproduire la logique du jeu** tout en **intégrant une IA évolutive** (Minimax, puis Deep Learning). Cette capture illustre le **résultat d'une simulation autonome**, démontrant l'intégration de l'algorithme Minimax.

Ce mode console m'a permis de tester rapidement la logique du jeu sans interface graphique. Il est le fruit d'une architecture **MVC** claire (séparation modèle / vue / contrôleur), facilitant les évolutions futures, comme une version JavaFX.

Même si le rendu console est simple, il est essentiel : il **vérifie le bon fonctionnement de la logique métier**, des entrées utilisateur et de l'IA. Il m'a notamment permis de repérer plusieurs erreurs dans la détection de victoire.

Ce projet m'a permis de renforcer mes compétences en **programmation objet**, en **architecture logicielle**, et en **conception de jeu avec IA**. À ce stade, je suis capable de créer des projets ludiques bien structurés, évolutifs et testés.

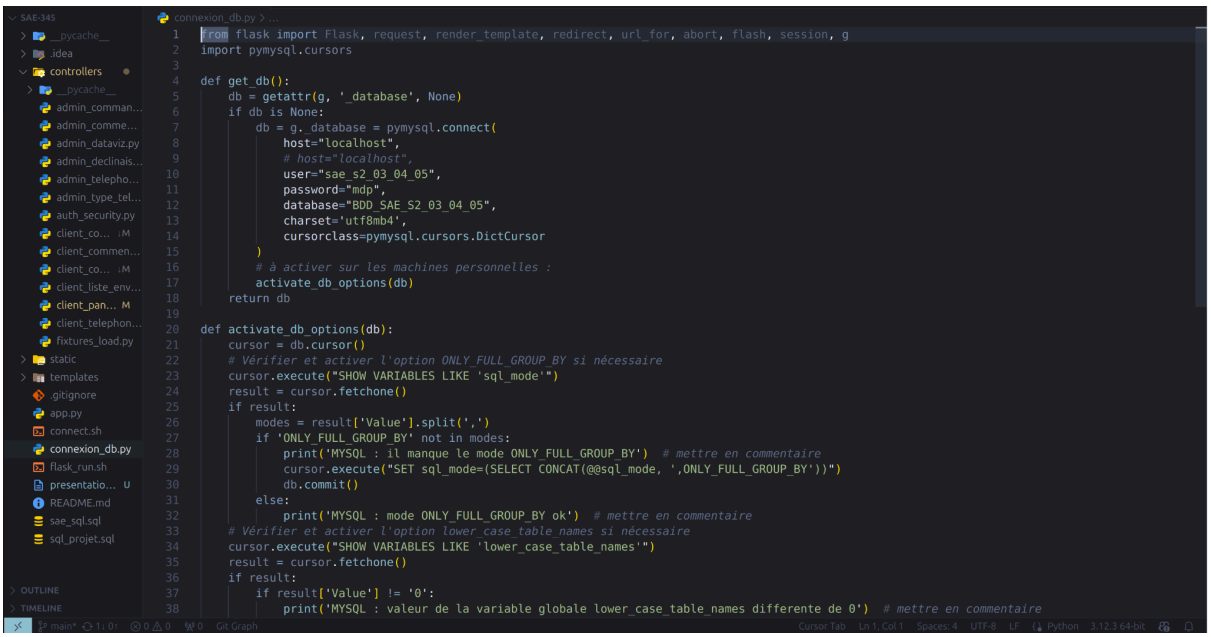

Figure 2– Fichier `connexion_db.py` : gestion de la connexion MySQL et des options SQL dans une application Flask

Dans la figure 2 , j'ai été amené à développer une couche de connexion entre l'application Flask et la base de données MySQL. Cette tâche a été centralisée dans le fichier `connexion_db.py`, qui assure la connexion sécurisée et dynamique à la base de données utilisée par l'application.

Dans ce fichier, j'ai mis en œuvre le module `pymysql` pour établir la connexion à MySQL. J'ai utilisé l'objet g fourni par Flask pour gérer une connexion persistante au sein d'une requête HTTP, évitant ainsi d'ouvrir plusieurs connexions simultanées. Cette approche repose sur le principe de singleton, une bonne pratique dans le développement web.

La fonction `get_db()` initialise la connexion si elle n'existe pas encore. Les paramètres de connexion (nom d'hôte, identifiants, nom de la base, encodage) sont définis en dur pour l'instant, mais cette structure est facilement extensible vers des variables d'environnement. Le curseur est configuré en `DictCursor` afin de faciliter la manipulation des résultats sous forme de dictionnaire plutôt que de tuples classiques.

Une fois la connexion établie, la fonction `activate_db_options()` est automatiquement appelée pour configurer deux options SQL essentielles. La première concerne le mode `ONLY_FULL_GROUP_BY`, souvent activé par défaut sur MySQL, qui peut provoquer des erreurs dans certaines requêtes utilisant GROUP BY. Le script vérifie sa présence dans la configuration SQL et le désactive si besoin via une requête SQL dynamique.

La seconde option gérée concerne la variable `lower_case_table_names`, qui influence la sensibilité à la casse des noms de tables. Cette configuration est importante pour assurer la compatibilité entre les environnements Linux et Windows.

Ce module m'a permis de mobiliser plusieurs compétences clés. Sur le plan des **savoirs**, j'ai consolidé mes connaissances sur le fonctionnement des connexions MySQL via PyMySQL,

sur les particularités de la configuration SQL, et sur l’usage de g dans Flask pour les objets partagés. Sur le plan des **savoir-faire**, j’ai pratiqué l’exécution de requêtes SQL dynamiques, la lecture conditionnelle de résultats SQL, et l’automatisation de l’environnement de base de données.

Enfin, cette page montre un souci d’extensibilité et de portabilité du code, puisque les options peuvent être adaptées à différents serveurs ou systèmes. Ce travail m’a permis de comprendre l'importance d’une configuration fine et adaptable dans un projet web professionnel.

Figure 3: Route GET `/admin/telephone/add` pour l'affichage du formulaire d'ajout de téléphone

La figure 3 presente la route GET - `@admin_telephone.route('/admin/telephone/add', methods=['GET'])` est utilisée pour préparer et afficher la page d'ajout d'un nouveau téléphone depuis l'interface d'administration. Cette route joue un rôle clé dans la séparation logique entre la récupération des données et l'affichage du formulaire. Lorsqu'un administrateur accède à cette URL, la fonction `add_telephone()` est exécutée. Elle commence par établir une connexion à la base de données via `get_db().cursor()`. Ensuite, une requête SQL est définie pour sélectionner l'identifiant et le libellé de tous les types de téléphones présents dans la table `type_telephone`, triés par ordre alphabétique de libellé (`ORDER BY libelle_type_telephone`). Cette information est essentielle pour alimenter dynamiquement la liste déroulante du formulaire HTML, permettant à l'administrateur de choisir le type de téléphone à ajouter.

La requête est exécutée via `mycursor.execute(sql)` et le résultat est stocké dans la variable `types_telephone` après appel de `fetchall()`. Cette liste d'objets est ensuite transmise à la vue HTML `add_telephone.html` à l'aide de la fonction `render_template()`. Le passage de la variable `types_telephone` au template permet à la vue d'afficher dynamiquement les options disponibles dans le formulaire. Cette approche montre une bonne pratique de développement web en Flask : préremplir le formulaire avec des données récupérées dynamiquement depuis la base pour éviter tout codage en dur.

Sur le plan des savoir-faire, cette portion de code illustre la maîtrise de la programmation orientée serveur avec Flask, l'utilisation de requêtes SQL simples mais efficaces, et l'interaction fluide entre la couche de données et la couche de présentation. Elle démontre également la capacité à structurer un contrôleur de manière claire et modulaire.