YUBEN

—Le projet–

Nous avons choisi ce projet suite à un changement consécutif de 3 idées en l'espace de 3 semaines. Nous avons bloqué sur celle-ci qui est un jeu qui permet de faire visiter le lycée et de communiquer pour les personnes présentes sur le même réseau comme par exemple celui de la classe de NSI.

Notre projet est entièrement codé en python objet pour la structure et les différents fonctionnements et certaines parties sont en SQL par rapport à la base de données.

Notre projet comporte donc une carte et un personnage qui se déplace dessus et un chat qui permet de communiquer entre les différents utilisateurs sur un même réseau

—La Base de donnée—

```
-- Affichage du schéma de la base de données

-- Table des utilisateurs
CREATE TABLE utilisateurs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50),
    prenom VARCHAR(50),
    pseudonyme VARCHAR(50),
    mot_de_passe VARCHAR(50)
);

-- Table du chat
CREATE TABLE chat (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_eleve INT,
    message TEXT,
    FOREIGN KEY (id_eleve) REFERENCES utilisateurs(id)
);
```

Voici le schéma sous forme de requête SQL de notre base de données faite par Ruben

—Une classe faite par chacuns—

Ruben:

```
"""
    Ce fichier contient la classe DatabaseManager qui est utilisée pour gérer les
opérations de base de données.
"""

import mysql.connector

class DatabaseManager:
    def __init__(self, host, user, password, database):
        """
            Constructeur de la classe DatabaseManager.
        """
        self.host = host
        self.user = user
        self.password = password
        self.database = database

    def get_connection(self):
        """
            Méthode pour obtenir une connexion à la base de données.
        """
        return mysql.connector.connect(host=self.host, user=self.user,
password=self.password, database=self.database)

    def check_user(self, username, password):
        """
            Méthode pour vérifier si un utilisateur existe dans la base de données.
        """
        conn = self.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM utilisateurs WHERE pseudonyme = %s AND
mot_de_passe = %s", (username, password))
        user = cursor.fetchone()
        cursor.close()
        conn.close()
        return user
```

```
    def create_user(self, nom, prenom, pseudonyme, mot_de_passe):
        """
            Méthode pour créer un nouvel utilisateur dans la base de données.
        """
        conn = self.get_connection()
        cursor = conn.cursor()
        try:
            cursor.execute(
                "INSERT INTO utilisateurs (nom, prenom, pseudonyme, mot_de_passe)
VALUES (%s, %s, %s, %s)",
                (nom, prenom, pseudonyme, mot_de_passe))
            conn.commit()
            return True
        except Exception as e:
            print(f"Failed to register user: {e}")
            return False
        finally:
            cursor.close()
            conn.close()

    def send_message(self, user_id, message):
        """
            Méthode pour envoyer un message dans le chat, dans la base de données.
        """
        conn = self.get_connection()
        cursor = conn.cursor()
        cursor.execute("INSERT INTO chat (id_eleve, message) VALUES (%s,%s)",
(user_id, message,))
        conn.commit()
        cursor.close()
        conn.close()

    def get_messages(self):
        """
            Méthode pour obtenir les messages du chat depuis la base de données.
        """
        conn = self.get_connection()
        cursor = conn.cursor()
        cursor.execute("""
                    SELECT utilisateurs.pseudonyme, chat.message
                    FROM chat
                    JOIN utilisateurs ON chat.id_eleve = utilisateurs.id
```

```
                ORDER BY chat.id DESC LIMIT 3
            """)
        messages = cursor.fetchall()
        cursor.close()
        conn.close()
        return messages
```

1. __init__(self, host, user, password, database): Le constructeur initialise les paramètres de connexion à la base de données, tels que l'hôte, l'utilisateur, le mot de passe et le nom de la base de données. Ces informations sont nécessaires pour établir une connexion réussie avec la base de données lors de la création d'une instance de la classe.

2. get_connection(self): Cette méthode établit une connexion à la base de données en utilisant les paramètres fournis lors de l'initialisation de la classe. Elle retourne l'objet de connexion qui peut être utilisé pour exécuter des requêtes SQL et interagir avec la base de données.

3. `check_user(self, username, password) : Cette méthode vérifie si un utilisateur avec le pseudonyme et le mot de passe fournis existe dans la base de données. Elle exécute une requête SELECT dans la table des utilisateurs en spécifiant le pseudonyme et le mot de passe, puis renvoie les informations de l'utilisateur si elles correspondent, sinon elle renvoie `None`.

4. create_user(self, nom, prenom, pseudonyme, mot_de_passe) : Cette méthode crée un nouvel utilisateur dans la base de données en insérant ses informations dans la table des utilisateurs. Elle prend en paramètre le nom, le prénom, le pseudonyme et le mot de passe du nouvel utilisateur, puis exécute une requête INSERT pour ajouter ces informations à la base de données. Si l'opération réussit, elle renvoie `True`, sinon elle renvoie `False`.

5. send_message(self, user_id, message) : Cette méthode envoie un message dans le chat en insérant les informations de l'utilisateur et le message lui-même dans la table du chat. Elle prend en paramètre l'ID de l'utilisateur qui envoie le message et

le contenu du message, puis exécute une requête INSERT pour ajouter ces informations à la base de données.

6. get_messages(self) : Cette méthode récupère les messages du chat depuis la base de données. Elle exécute une requête SELECT pour sélectionner les messages les plus récents de la table du chat, en les associant aux pseudonymes des utilisateurs correspondants à partir de la table des utilisateurs grâce à une jointure. Ensuite, elle renvoie les messages sous forme de liste pour qu'ils puissent être affichés dans l'interface utilisateur du jeu.

En résumé, la classe `DatabaseManager` fournit une interface entre le jeu et la base de données, facilitant ainsi la gestion des données utilisateur et des messages du chat. Elle encapsule les opérations SQL nécessaires dans des méthodes simples et abstraites, ce qui rend l'interaction avec la base de données plus facile et plus sécurisée pour les autres composants du jeu.

Yuna:

```
    def update(self):
        """
            Methode pour mettre à jour le jeu.
            Elle pertmet de gérer les déplacements du joueur.
        """
        keys = pygame.key.get_pressed()
        speed = 1
        if not self.chat_box.active:
            if keys[pygame.K_z]:
                self.player.rect.y -= speed
                self.change_direction('up')

            if keys[pygame.K_q]:
                self.player.rect.x -= speed
                self.change_direction('left')

            if keys[pygame.K_s]:
                self.player.rect.y += speed
                self.change_direction('down')

            if keys[pygame.K_d]:
```

```
                self.player.rect.x += speed
                self.change_direction('right')

        self.group.center(self.player.rect.center)
```

Cette méthode `update(self)` est appelée à chaque itération de la boucle de jeu pour mettre à jour l'état du jeu. Voici une explication ligne par ligne :

1. **`keys = pygame.key.get_pressed()`** : Cette ligne récupère l'état de toutes les touches du clavier à l'instant donné. Elle retourne un tableau où chaque élément correspond à une touche, et indique si elle est enfoncée (1) ou non (0).

2. **`speed = 1`** : Cette ligne définit la vitesse de déplacement du joueur. Ici, elle est fixée à 1 pixel par itération de la boucle de jeu.

3. **`if not self.chat_box.active:`** : Cette condition vérifie si la boîte de chat n'est pas active. Si le chat est actif, les touches de déplacement ne sont pas prises en compte.

4. **Déplacements du joueur** : Les quatre blocs `if` suivants vérifient si les touches de déplacement (Z, Q, S, D) sont enfoncées. Si c'est le cas et que le chat n'est pas actif, le joueur se déplace dans la direction correspondante en ajustant les coordonnées de son rectangle (`self.player.rect`). Par exemple, si la touche Z est enfoncée, la coordonnée y du rectangle du joueur est décrémentée de la valeur de `speed`, ce qui fait monter le joueur à l'écran. À chaque déplacement, la méthode `change_direction()` est appelée pour mettre à jour la direction du joueur.

5. **`self.group.center(self.player.rect.center)`** : Cette ligne recentre le groupe d'affichage (`self.group`) sur le centre du rectangle du joueur. Cela permet de suivre le joueur à l'écran lorsqu'il se déplace.

En résumé, cette méthode permet de gérer les déplacements du joueur en fonction des touches du clavier tout en vérifiant que le chat n'est pas actif. Elle

ajuste les coordonnées du joueur et recentre l'affichage sur lui pour le suivre à l'écran.