Structure du Serveur de Jeu Multijoueur

1 Comment structurer la logique du jeu côté serveur ?

Le serveur devra gérer trois grandes responsabilités :

1. **Gérer les connexions des joueurs** (via Socket.io).
2. **Gérer la logique du jeu** (règles, état des parties, scores...).
3. **Envoyer les mises à jour aux clients** (synchronisation en temps réel).

1.1 Exemple de structuration du serveur

```
serveur/
        server.js           # Point d'entr e du serveur
        gameLogic.js        # Toute la logique du jeu ici
        roomsManager.js     # G re les parties et les joueurs
        package.json        # D pendances du projet
        database.js         # Connexion    la base de donn es (MongoDB, P
        models/
            player.js       # Mod le pour les joueurs
            game.js         # Mod le pour les parties
```

2 Express.js et Socket.io : Travailler ensemble

2.1 Express.js

Express.js est un **framework** qui simplifie la gestion des requêtes HTTP en Node.js. Il est utilisé pour créer des API RESTful ou des serveurs web.

Cas d'usage :

— Servir des fichiers HTML, CSS, JavaScript.
— Gérer des requêtes API (`GET`, `POST`, `PUT`, `DELETE`).
— Communiquer avec une base de données.

2.2 Socket.io

Socket.io est une bibliothèque permettant d'**envoyer et recevoir des messages en temps réel** entre le serveur et les clients via WebSockets.

Cas d'usage :

— Gestion du **multijoueur en temps réel** (ex : un jeu comme le tien).
— **Chat en direct**.
— **Mises à jour instantanées** (ex : notifications).

3 Combinaison Express.js et Socket.io

Dans un jeu multijoueur, on combine souvent Express et Socket.io :

— **Express.js** : pour gérer des routes API et interagir avec une base de données.
— **Socket.io** : pour la communication en temps réel entre les joueurs.