Niveau 1: Les bases du jeu

- **Déplacement des joueurs :**
  - Deux joueurs peuvent se déplacer sur une grille 2D dans les quatre directions (haut, bas, gauche, droite)
  - Les mouvements sont limités par les bords de la grille
  - Les joueurs ne peuvent pas occuper la même case, à l'exception de la plaque de fin
- **Point de départ et d'arrivée :**
  - Chaque niveau a deux points de départ (un par joueur)
  - Une plaque de pression dorée marque la fin du niveau
  - Le niveau est terminé quand les deux joueurs sont simultanément sur la plaque dorée

Niveau 2: Obstacles et environnement

- **Gestion des murs :**
  - Des murs bloquent le passage des joueurs
  - Les murs sont placés selon les données du fichier JSON
- **Chargement de niveau :**
  - Implémentation de la lecture des fichiers JSON
  - Le fichier contient les positions des murs, points de départ et plaque dorée

Niveau 3: Mécaniques de portes

- **Plaques de pression :**
  - Des plaques de pression colorées peuvent être activées par les joueurs
  - Une plaque reste activée tant qu'un joueur se tient dessus
- **Portes :**
  - Les portes sont liées à des plaques de pression de même couleur
  - Une porte s'ouvre uniquement quand sa plaque est activée
  - Une porte se referme dès que sa plaque est libérée
- **Chargement des portes&plaques:**
  - Les portes et les plaques de pression sont placés selon les données du fichier JSON

Niveau 4: Des portes et des plaques multiples

- **Faire en sorte qu'il puisse y avoir plusieurs portes et plusieurs plaques de la même couleurs, tous reliés ensembles**

Niveau 5: Multijoueur en ligne

- **Architecture client-serveur :**
  - **Mise en place d'une API RESTful simple :**
    - Utiliser Express.js pour créer un serveur Node.js qui gère les requêtes.
    - Installer les dépendances nécessaires :

```
npm install express mongoose cors body-parser
```

- Créer le serveur et lui permettre de gérer les CORS (Cross-Origin Resource Sharing) pour que les clients puissent communiquer avec le serveur, notamment si le frontend et le backend sont hébergés sur des serveurs différents.

Endpoints API

1. **Créer une partie**
   - **Endpoint :** `POST /game/create`
     - **Description :** Crée une nouvelle partie et renvoie un identifiant unique pour celle-ci.
     - **Request Body :**

```
{
  "level": "0"
}
```

     - **Response :**

```
{
  "gameId": "12345",
  "level" : "0",
  "status": "waiting",
  "players": []
}
```

     - **Utilisation :** Lorsqu'un joueur veut commencer un nouveau jeu, il envoie une requête vers cet endpoint, qui crée une nouvelle partie dans le serveur.
2. **Rejoindre une partie**
   - **Endpoint :** `POST /game/join`
     - **Description :** Permet à un joueur de rejoindre une partie existante en utilisant un identifiant de partie.
     - **Request Body :**

```
{
  "gameId": "12345",
  "playerName": "Player1"
}
```

     - **Response :**

```
{
  "gameId": "12345",
  "players": ["Player1", "Player2"],
```

```
  "status": "active"
}
```

- **Utilisation :** Lorsque le deuxième joueur souhaite rejoindre une partie, il envoie cette requête avec l'identifiant du jeu. Le serveur met à jour l'état de la partie avec le nouveau joueur.

3. **Obtenir l'état du jeu**

- **Endpoint :** `GET /game/{id}`
  - **Description :** Récupère l'état actuel d'une partie spécifique.
  - **Response :**

```
{
  "gameId": "12345",
  "players": [
    {"position": {"x": 1, "y": 2}},
    {"position": {"x": 3, "y": 4}}
  ],
  "status": "active"
}
```

  - **Utilisation :** Chaque client peut appeler cet endpoint pour obtenir l'état actuel de la partie, ce qui lui permet de mettre à jour l'affichage à l'écran.

4. **Mettre à jour la position d'un joueur**

- **Endpoint :** `POST /game/{id}/move`
  - **Description :** Met à jour la position d'un joueur dans le jeu.
  - **Request Body :**

```
{
  "playerName": "Player1",
  "newPosition": {"x": 2, "y": 3}
}
```

  - **Response :**

```
{
  "message": "Position mise à jour",
  "players": [...]
}
```

  - **Utilisation :** Lorsqu'un joueur se déplace dans le jeu, il envoie une demande pour mettre à jour sa position. Le serveur valide le mouvement (par exemple, vérifie s'il y a des obstacles)

et met à jour l'état de la partie.

5. **Gestion de la déconnexion d'un joueur**
    - **Endpoint :** `POST /game/{id}/disconnect`
        - **Description :** Permet de gérer la déconnexion d'un joueur.
        - **Request Body :**

```
{
  "playerName": "Player1"
}
```

        - **Response :**

```
{
  "message": "Déconnexion réussie",
  "players": [...] // Mise à jour de la liste des joueurs
}
```

- **Utilisation :** Si un joueur se déconnecte, il peut envoyer cette requête pour informer le serveur. Cela permet de mettre à jour la liste des joueurs et de gérer l'état de la partie (par exemple, la mettre en pause ou continuer sans le joueur).

Synchronisation avec WebSockets

Pour une expérience de jeu fluide entre les joueurs, vous devrez également mettre en place un serveur WebSocket. Voici des points clés :

- **Initialiser le WebSocket :** Établissez une connexion WebSocket pour chaque client lors de la création ou de la connexion à une partie.
- **Écouter les mises à jour :** Utilisez les WebSocket pour diffuser les mises à jour d'état (mouvements, positions, événements comme les activations/de-désactivations des plaques de pression) à tous les clients connectés.
- **Gérer les déconnexions :** Modifiez l'état du jeu lorsque la connexion WebSocket d'un joueur est fermée, ce qui permet de gérer les scénarios de déconnexion de manière plus dynamique.

Fonctionnalités Optionnelles :

- **Niveaux additionnels :**
    - Faites parler vos skills en Level Design !
- **Éléments de gameplay additionnels :**
    - Téléporteurs
    - Plaques à minuteur (restent activées quelques secondes)
    - Portes à double activation (nécessitent deux plaques)

- **Éditeur de niveaux :**
    - Interface graphique pour créer/modifier des niveaux
    - Placement drag & drop des éléments
    - Export au format JSON compatible
    - Validation des niveaux créés
- **Intégration du Sokoban :**
    - Ajouter des trous et des rochers comme éléments de gameplay avec la logique du Sokoban.
    - Attention, cela risque de demander une modification de votre API pour la gestion de l'emplacement des rochers.
- **Système de compte :**
    - Création de compte joueur
    - Sauvegarde de la progression
    - Statistiques (temps par niveau, nombre d'essais...)
- **Chat in-game :**
    - Communication entre les joueurs
    - Émotes ou messages prédéfinis
    - Option pour désactiver le chat
- **Et plus encore...**
    - Amusez-vous !