# TP Step 4: Prototype 2 du jeu

## Objectifs pédagogiques

- Comprendre la notion de challenge dans un jeu
- Introduction au level design

## Objectifs de l'étape

- Ajout du composant Bloc
- Ajout de l'entité Bloc
- Amélioration du système `movementSystem` pour prendre en compte la collision entre le joueur et les blocs

## Introduction

Nous souhaitons maintenant ajouter des entités dans le jeu pour empêcher le joueur d'atteindre son objectif. Concrètement, cela sera représenté par des blocs que le joueur devra déplacer en les poussant uniquement.

En poussant les blocs, le joueur devra libérer un chemin vers la sortie et ainsi terminer le niveau.

## Les étapes du travail

### 1. Point de départ

- Récupérez le projet de l'étape 4 soit par **git** ou par un fichier décompressé.
- Ouvrez le répertoire du projet avec **Visual Studio Code**.
- Lancez **Live Server** pour consulter la page `index.html` dans un navigateur.

### 2. Comprendre le chargement de la carte du jeu

Dans `game.js`, consultez la fonction `loadGameData()` :

```
loadGameData() {
    this.levelData = this.cache.json.get(`levelData${this.number}`);
    if (!this.levelData) {
        this.scene.start("Splash");
    }
}
```

Chaque niveau dans notre jeu est identifié par un numéro. Cette fonction va demander à récupérer un objet JavaScript (JSON) qui représente la description de la carte du niveau. Tous ces niveaux sont disponibles dans le répertoire `assets/levels`.

Voici un exemple d'un tel niveau en JSON :

```
{
    "stage": {
        "level": 0,
        "grid": {
            "width": 6,
            "height": 6
        },
        "exit": {
            "x": 5,
            "y": 2
        },
        "blocks": [
            {
                "id": 1,
                "position": { "x": 2, "y": 2 }
            },
            {
                "id": 2,
                "position": { "x": 3, "y": 1 }
            },
            {
                "id": 3,
                "position": { "x": 3, "y": 3 }
            },
            {
                "id": 4,
                "position": { "x": 4, "y": 2 }
            }
        ]
    },
    "player": {
        "position": { "x": 3, "y": 2 }
    }
}
```

Dans cette description, nous trouvons les informations suivantes : - La taille de la grille du jeu `stage.grid` - La liste des blocs dans `stage.blocks` - La position de la porte de sortie `stage.exit` - La position initiale du joueur `player.position`

### 3. Comprendre le modèle interne de l'état du jeu

Nous allons créer des variables qui représentent l'état de notre jeu. Ces variables sont créées par la fonction `gameStateConfig()` :

```
gameStateConfig() {
    this.gameState = {};
    this.gameState.grid = this.createGameState(this.levelData);
```

```
    this.solved = false;
    this.gameOver = false;
    this.finishing = false;
}
```

- L'attribut `solved` indique si le joueur a réussi le puzzle.
- L'attribut `gameOver` nous permet de savoir si la partie est terminée.
- L'attribut `finishing` nous permet de savoir si le jeu est terminé et que nous sommes en train de finaliser la scène avant de la quitter.
- L'attribut `gameState` définit une représentation interne du plateau de jeu sous forme de matrice. Chaque case de la matrice contient `undefined` si aucune entité n'y est présente. Si une entité est présente à cette position, la case contient son identifiant, ce qui permet de la retrouver dans le monde ECS.

### 4. Ajouter le composant Bloc

Ajoutez un composant `Bloc` qui sera un *tag* pour typer les entités représentant un obstacle dans le jeu.

Suggestion de code dans `Bloc.js` :

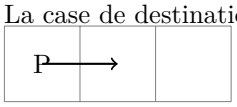
```
export const Bloc = defineComponent();
```

### 5. Implémentation du système de mouvement

Consultez le code dans `movementSystem.js`. La fonction `tryMove` essaie de faire bouger le joueur quand cela est possible.

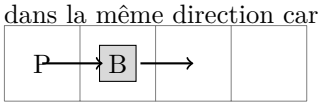
Le mouvement du joueur dans une direction est possible dans les cas suivants:

- La case de destination est libre




Case libre

- La case de destination est occupée, mais il est possible de pousser ce bloc dans la même direction car la case suivante est libre.




Pousser un bloc

Actuellement, la fonction ne traite que du cas de la direction `right`. Complétez ce code pour traiter les autres directions.

Vous pouvez consulter l'état des cases dans la variable d'état `scene.gameState.grid`. Vous devez mettre à jour cette variable en cas de mouvement des entités.

### 6. Test et validation

Testez maintenant votre application. Vérifiez que les règles du jeu sont bien respectées : le joueur doit pouvoir pousser les blocs si la place derrière eux est

libre, et il doit atteindre la sortie pour terminer le niveau.