Documentation du dossier `system` (serveur, la logique de jeu)

Arborescence

```
system
├── CollisionSystem.cpp
├── CollisionSystem.hpp
├── EnemySystem.cpp
├── EnemySystem.hpp
├── GameServer.cpp
├── GameServer.hpp
├── HealthSystem.cpp
├── HealthSystem.hpp
├── MovementSystem.cpp
├── MovementSystem.hpp
├── ProjectileSystem.cpp
├── ProjectileSystem.hpp
├── SpawnSystem.cpp
└── SpawnSystem.hpp
```

Vue pour les développeurs utilisateurs

But général

Le dossier `system` contient tous les systèmes du serveur de jeu. Chaque système est responsable d'une logique précise, par exemple :

- Gestion des collisions
- Gestion des ennemis
- Gestion des déplacements
- Gestion des projectiles
- Gestion des points de vie / santé
- Gestion des spawns d'entités

Ces systèmes travaillent avec **le moteur principal** (`rtype_engine`) pour mettre à jour l'état du jeu à chaque frame ou tick du serveur.

Comment utiliser

1. Inclure `GameServer.hpp` dans notre fichier `main_server.cpp`.
2. Créer une instance du `GameServer`.
3. Appeler la méthode principale de boucle de jeu (ex. `run()` ou `updateLoop()`).
4. Le serveur s'occupe de mettre à jour tous les systèmes (`CollisionSystem`, `MovementSystem`, etc.).

En pratique, le développeur n’a **pas besoin de toucher aux autres fichiers** pour lancer le serveur. Tout est encapsulé dans `GameServer`.

Vue technique (explication fichier par fichier)

Chaque fichier `.cpp/.hpp` contient un **système spécifique** du moteur ECS (Entity-Component-System).

2.1 CollisionSystem

- **CollisionSystem.hpp**
    - Déclare la classe `CollisionSystem`.
    - Méthodes principales :
        - `update()`: Vérifie toutes les collisions entre entités (joueurs, ennemis, projectiles, obstacles).
    - Utilise les composants de position et de hitbox des entités.
- **CollisionSystem.cpp**
    - Implémente la logique de détection des collisions.
    - Met à jour les états des entités touchées (ex : réduit les PV si un projectile touche un joueur ou un ennemi).

2.2 EnemySystem

- **EnemySystem.hpp**
    - Déclare la classe `EnemySystem`.
    - Responsable de la gestion des ennemis (IA, déplacements, comportements).
    - Méthodes :
        - `spawnEnemy()`
        - `updateEnemies()`
- **EnemySystem.cpp**
    - Implémente l’IA de base (suivi du joueur, déplacements automatiques, attaques).
    - Communique avec `MovementSystem` pour appliquer les déplacements.

2.3 GameServer

- **GameServer.hpp**
    - Classe principale du serveur.
    - Contient tous les systèmes comme membres :

```
CollisionSystem collisionSystem;
EnemySystem enemySystem;
```

```
HealthSystem healthSystem;
MovementSystem movementSystem;
ProjectileSystem projectileSystem;
SpawnSystem spawnSystem;
```

    - Méthode `run()` : boucle principale du serveur qui met à jour tous les systèmes à chaque tick.

- **GameServer.cpp**
    - Implémente `run()` et initialise tous les systèmes.
    - Reçoit les entrées réseau (via `Networkmanager`) et met à jour l'état du jeu.
    - Envoie les mises à jour aux clients connectés.

2.4 HealthSystem

- **HealthSystem.hpp**
    - Déclare la classe `HealthSystem`.
    - Gère la santé des entités.
    - Méthodes :
        - `updateHealth()`
        - `applyDamage(Entity, int)`
- **HealthSystem.cpp**
    - Implémente la logique de dégâts.
    - Vérifie si des entités sont mortes et notifie le `GameServer`.

2.5 MovementSystem

- **MovementSystem.hpp**
    - Classe `MovementSystem`.
    - Gère les déplacements des entités.
    - Méthodes :
        - `moveEntity(Entity, Vec2 direction)`
        - `updatePositions()`
- **MovementSystem.cpp**
    - Calcule les nouvelles positions en fonction de la vitesse et direction des entités.
    - Prend en compte les collisions via `CollisionSystem`.

2.6 ProjectileSystem

- **ProjectileSystem.hpp**

    - Classe `ProjectileSystem`.
    - Responsable des projectiles (tir du joueur et ennemis).
- **ProjectileSystem.cpp**
    - Gère le mouvement des projectiles.
    - Vérifie collisions avec entités via `CollisionSystem`.
    - Supprime les projectiles après collision ou sortie de l'écran.

2.7 SpawnSystem

- **SpawnSystem.hpp**
    - Classe `SpawnSystem`.
    - Gère l'apparition des entités (joueurs, ennemis, power-ups).
- **SpawnSystem.cpp**
    - Implémente les règles d'apparition et le timing des spawns.
    - Communique avec `GameServer` pour ajouter de nouvelles entités dans le moteur ECS.

Comment tout se connecte

```
GameServer
  ├─ CollisionSystem
  ├─ EnemySystem
  ├─ HealthSystem
  ├─ MovementSystem
  ├─ ProjectileSystem
  └─ SpawnSystem
```

- `GameServer` = boucle principale + gestion des systèmes
- Chaque système s'occupe d'un aspect précis du jeu
- Les systèmes peuvent interagir entre eux via le `GameServer` ou directement via les entités du moteur ECS (`rtype_engine`)

Conseils pour un développeur

- **Pour utiliser** : inclure seulement `GameServer` et lancer la boucle de jeu.
- **Pour modifier ou étendre** : éditer le système correspondant (`*.cpp/.hpp`).
    - Exemple : ajouter un nouveau type d'ennemi → modifier `EnemySystem`.
- **Pour debug** : ajouter des logs dans `update()` de chaque système pour suivre l'état du serveur.