EPITECH Lyon.

28 janvier 2025

Projet R-Type

C6 : Veille légale et besoins des PSH

1 - Accessibilité numérique

Veille sur les réglementations relatives à l'accessibilité numérique.

- **Directives européennes** (*2016/2102*)
- **R.G.A.A** (*Référentiel Général d'Amélioration de l'Accessibilité*)
- **Recommandations du W.C.A.G** (*Web Content Accessibility Guidelines*).

A. Perceptible

- Texte alternatif.
- Contraste.
- Sous-titres.

B. Utilisable

- Navigation clavier.
- Focus visible.
- Temps ajustable.

C. Compréhensible

- Langue définie.
- Labels clairs.
- Erreurs explicites.

C6 : Veille légale et besoins des PSH

2 - Etude comparative de technologie d’accessibilité

Technologie	Critères d'accessibilité	Avantages	Inconvénients	Conclusion
GLFW	- Applications graphiques avancées. - Peu d’intégration pour l’accessibilité utilisateur.	- Léger et performant. - Prise en charge des événements matériels.	- Documentation moins complète d’accessibilité. - Peu adapté aux interfaces textuelles.	Inadapté pour répondre aux besoins des PSH.
SFML (Simple and Fast Multimedia Library)	- Gestion simplifiée des événements.	- Documentation claire. - Peu gourmand en ressources.	- Pas de support natif des technologie. - Légèrement plus lourd que SDL.	Inadapté pour répondre aux besoins des PSH.
SDL (Simple DirectMedia Layer)	- Interaction avec des contrôleurs alternatifs. - Support du texte ajustable en taille et en couleur.	- Compatibilité avec divers systèmes d'exploitation. - Large communauté de développeurs. - Facile à intégrer.	- Complexe à configurer pour des interfaces graphiques basiques. - Légèrement plus lourd que SFML.	**Choix retenu :** Répond aux besoins d’accessibilité essentiels.

C6 : Veille légale et besoins des PSH

3 - Librairies et technologie d’accessibilité

Nous pouvons étendre les fonctionnalités de la **SDL** en intégrant des **bibliothèques tierces** ou en ajoutant **manuellement** des fonctionnalités d’accessibilité.

Exemples :

- **Text-to-Speech (TTS) avec eSpeak NG.**

- **Support des entrées alternatives avec AT-SPI2 (Assistive Technology Service Provider Interface).**

- **Ajout de filtres de couleurs pour les personnes atteintes de daltonisme.**

C7 : Révision des protocoles existants et standards de sécurité

1 - Sécurité et Révision des Protocoles

Objectifs :

- **Assurer la sécurité des données et des communications réseau.**
- **Aligner les technologies utilisées sur les standards de sécurité actuels.**
- **Réviser les normes international de cybersécurité, telles que**

  ISO/IEC 27001 : Gestion de la sécurité de l'information.
  ISO/IEC 29147 : Gestion des vulnérabilités.

- **Adoption des recommandation de OWASP (Open Worldwide Application Security Project)**

  Injection : Prévenir avec des requêtes préparées.
  Données sensibles : Utiliser des algorithmes de chiffrement modernes.

C7 : Révision des protocoles existants et standards de sécurité

2 - Etude des protocoles, des vulnérabilités et solutions

1.

ENET

Technologie	Vulnérabilités identifiées	Solutions	Sources et références
ENet (bibliothèque réseau UDP)	- Absence de chiffrement natif. - Risque d'injection de données ou altération des paquets non authentifiés.	- Validation stricte des paquets entrants pour éviter les injections. - Surveillance des connexions pour détecter les anomalies. - Integration de TLS.	- Documentation officielle ENet. - Articles de sécurité réseau (*OWASP Networking Guidelines*).

Autres vulnérabilités identifié en veille :

CVE-2006-1194 : Cette vulnérabilité, découverte en 2006, concerne une erreur de signe entier dans la fonction enet_protocol_handle_incoming_commands du fichier protocol.c.

Aucune autre vulnérabilité spécifique à ENet n'a été répertoriée dans la base de données CVE récement.

C7 : Révision des protocoles existants et standards de sécurité

2 - Etude des protocoles, des vulnérabilités et solutions

2. 

Technologie	Vulnérabilités identifiées	Solutions	Sources et références
C++ et gestion mémoire	-Dépassements de tampon (**buffer overflow**) accès mémoire invalides. - Risques de crashs ou d'escalade de privilèges	- Bonnes pratiques de gestion mémoire (ex. : utilisation de **std::unique_ptr** et **std::vector**). - Analyse du code avec des outils de détection statique **Valgrind**	- OWASP Secure Coding Practices.

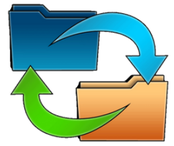
Autres vulnérabilités identifié en veille :

CVE-2023-43896 : Vulnérabilité de dépassement de tampon dans Macrium Reflect 8.1.7544 et versions antérieures permet aux attaquants d'escalader les privilèges ou d'exécuter du code arbitraire.

CVE-2025-21333 : Une vulnérabilité de dépassement de tampon basé sur le tas dans Microsoft Windows Hyper-V NT Kernel Integration VSP permet à un attaquant local d'obtenir des privilèges SYSTEM.

C7 : Révision des protocoles existants et standards de sécurité

2 - Etude des protocoles, des vulnérabilités et solutions

3. 

Technologie	Vulnérabilités identifiées	Solutions	Sources et références
Protocoles de communication	- Manque d'authentification des messages entre clients et serveur. - Risques d'usurpation d'identité ou envoi de données malveillantes pour perturber les clients ou le serveur.	- Ajout d'un système d'authentification des connexions avec des UUID sécurisés. - Chiffrement AES pour les données sensibles	- OWASP Top 10 Security Risks 2024. - Articles sur l'authentification sécurisée dans les jeux multijoueurs.

Autres vulnérabilités identifié en veille :

CVE-2025-23006 : SonicWall SMA1000 contiennent une vulnérabilité de désérialisation de données non fiables, permettant à un attaquant distant non authentifié d'exécuter des commandes système arbitraires.

C7 : Révision des protocoles existants et standards de sécurité

2 - Etude des protocoles, des vulnérabilités et solutions

4.

SDL

Technologie	Vulnérabilités identifiées	Solutions	Sources et références
SDL	- Sécurité réseau intégré faible pour la gestion des échanges de données.	- Utilisation d'ENet comme couche réseau sécurisée en complément.	- Documentation officielle SFML. - OpenClassroom

C8 : Solution technique créative sous contraintes

1 - Prototype d’ECS*

Volée	Inconvenients	Avantages	Conclusion
ECS de test	- agis comme un tableau pour les composants, ce qui limite ses utilisations (besoin de plus de fonctionnalitées)	- compréhension de base des systèmes et point clés	insatisfaisant car trop peu performant pour le temps investi. besoin de plus de complexité pour y voir un potentiel
ECS utilisé	- complexité grandissante par rapport aux différentes parties concernées (ajout d’ensembles tel qu’un menu ou des options...)	- autonome (ajout d’éléments simple, gestion automatique) - optimisé pour de meilleures performances (accès aux données simplifié)	grande modilarité et optimisation. pratique a utiliser a l’echelle du projet mais plus complexe a l’echelle de certain systèmes (renderer)
ECS cible	- même problematique de complexité que l’ECS utilisé	- plus rapide que l’ECS utilisé (meilleure gestion des données)	par manque d’organisation, les avantages de cet ECS ont été mis de côté ce qui a mener a sa complétion partielle

***ECS :** Entity Component System, model d’architecture logiciel pour la représentation des objets dans un jeu vidéo.

C8 : Solution technique créative sous contraintes

2 - Prototype de gestion des paquets avec ENET

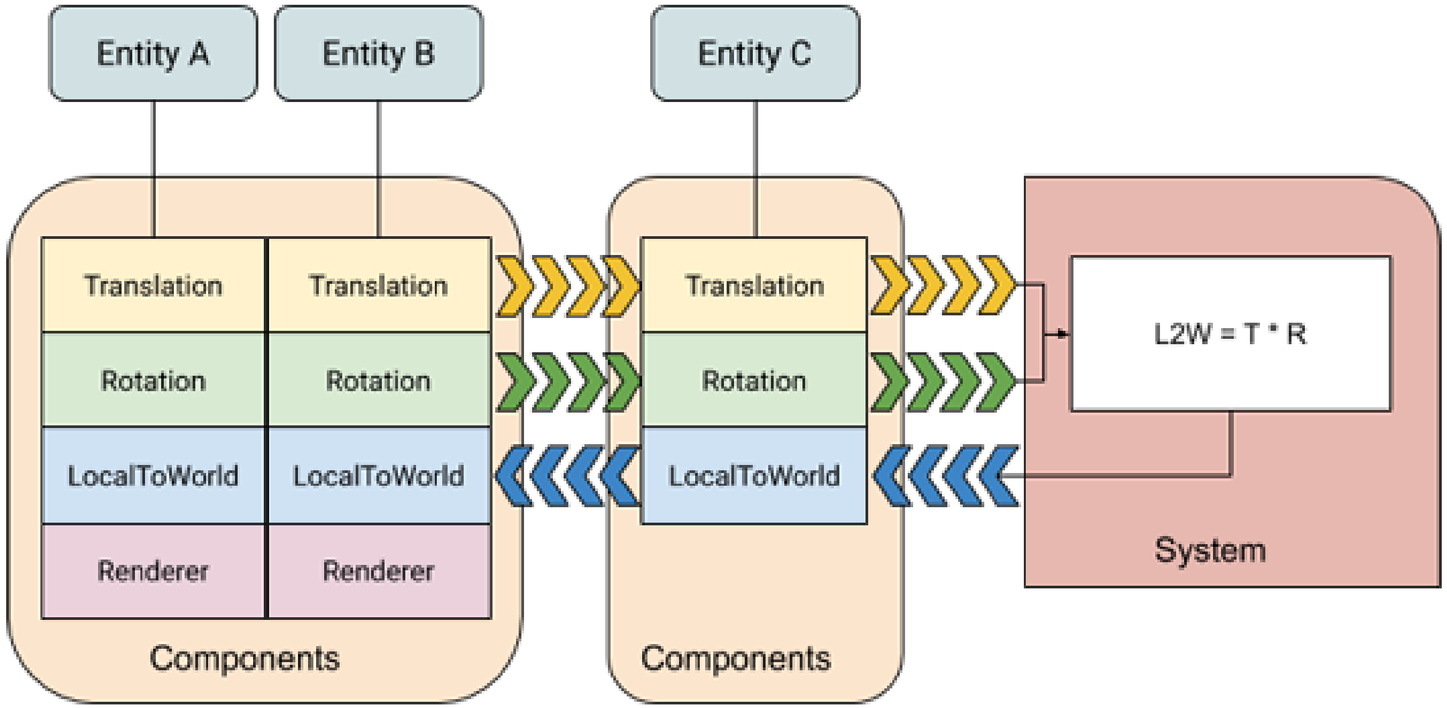
Volée	Inconvenients	Avantages	Conclusion
Protocole de test	- paquet trop petit par rapport aux données demandées - autorité du serveur inconstante - pas de sécurité	- a permis de se familiariser avec ENET	a permis de tester et comprendre la librairie dans son ensemble et d'identifier les points importants a développer pour la suite du projet
Protocole utilisé	- n'utilise pas TLS (Transport Layer Security) - pas de fonction de hashage - taille de paquet fixe	- bonne flexibilité (ajout de nouveaux protocoles..etc...) - accessible (gestion des connexions plus directe) - bonne gestion d'erreur	librairie simple qui permet des connexions fiables. peut encore être ameliorer sur le plan sécurité
Protocole cible	- gestion de la perte de paquets	- utilise TLS - fonction de hashage - taille de paquet flexible	option plus poussée résolvant les problèmes d'évolutivité et de sécurité

C9 : Hypothèse d’architecture et son urbanisme

Architecture	Intégration	Pérennité	Avantages	Inconvénients	Conclusion
Monolithique	- difficile dû a la complexité des autres composants du projet	- faible pérennité dû a l’interconnexion des différents composants	- plus simple a concevoir - plus rapide a mettre en place	- manque de modularité - complexité croissante qui rend le code difficile a maintenir	simple et rapide a concevoir, mais trop d’inconvénients sur le long terme
Microservice	- Facilite l'ajout de fonctionnalités (exemple : système de scoring ou moteur physique)	- Plus facile à maintenir et à faire évoluer : chaque service peut être amélioré indépendamment	- Modularité permettant des mises à jour ciblées (exemple : gestion du réseau séparée du moteur graphique)	- Complexe à synchroniser entre services pour des systèmes en temps réel comme le jeu en réseau	Solution idéale pour un jeu multijoueur avec des systèmes complexes, mais demande un investissement en temps et en ressources techniques.
EDA (Event-Driven Architecture)	- Très adapté à des événements en temps réel comme des collisions, des tirs, ou des événements réseau.	- Bonne évolutivité : ajout d'événements sans modifier l'ensemble du code.	- Asynchrone, donc adapté pour des systèmes où les actions ne dépendent pas toutes d'un ordre strict.	- Difficile à tester et à déboguer en raison de l'asynchronisme.	Idéal pour gérer des interactions complexes en temps réel dans un jeu, mais demande une gestion rigoureuse des événements et de leur impact.

C10 : Traduction des spécifications en une solution cohérente

1 - implémentation d’un ECS



C10 : Traduction des spécifications en une solution cohérente

1 - implémentation d'un ECS

```
class System {
    public:
  You, 1 second ago • Uncommitted changes
        ~System();

        template <class Func>
        void forEach(Func &&function);

        Registry &getRegistry();

        Registry const &getRegistry() const;

        std::vector<Entity> const &getEntities() const;

        void detachAll();

        std::size_t getEntityCount() const noexcept;

        virtual void onStart();

        virtual void onUpdate(float deltaTime);

        virtual void onShutdown();

        virtual void onEntityAttachment(Entity entity);

        virtual void onEntityDettachment(Entity entity);

        virtual void onEntityEnabled(Entity entity);

        virtual void onEntitydisabled(Entity entity);

    protected:
        System() = default;

        Filter &getFilter();
```

Classe Abstraite

C10 : Traduction des spécifications en une solution cohérente

1 - implémentation d’un ECS

```
class InputSystem : public ecs::System {
    public:
        InputSystem();
        virtual ~InputSystem();

        void onStart() override;

        void onShutdown() override;

        void onUpdate(float deltaTime) override;

    private:
        InputComponent _input;
};
```

Classe Abstraite

Héritage

C10 : Traduction des spécifications en une solution cohérente

1 - implémentation d’un ECS

Classe Abstraite

```
ecs::Entity EntityFactory::createProjectile(
    ecs::Registry& registry, float x, float y, float vx, float vy)      You,
{
    ecs::Entity projectile = registry.spawnEntity(uuid::generate_uuid_v4());
    projectile.addComponent<position>({x, y});
    projectile.addComponent<velocity>({vx, vy});
    projectile.addComponent<typeComponent>({EntityType::PROJECTILE});
    projectile.addComponent<Faction>({1});
    projectile.addComponent<Health>({1, 1});
    projectile.addComponent<Hitbox>({51, 26});
    return projectile;
}
```

Héritage

Factory

C11 : Segmentation du code

1 - Organisation du code

Les systèmes

```
system/
├── AudioManager.hpp
├── AudioSystem.hpp
├── InputSystem.hpp
├── MovementSystem.hpp
└── RendererSystem.hpp
```

Les composants

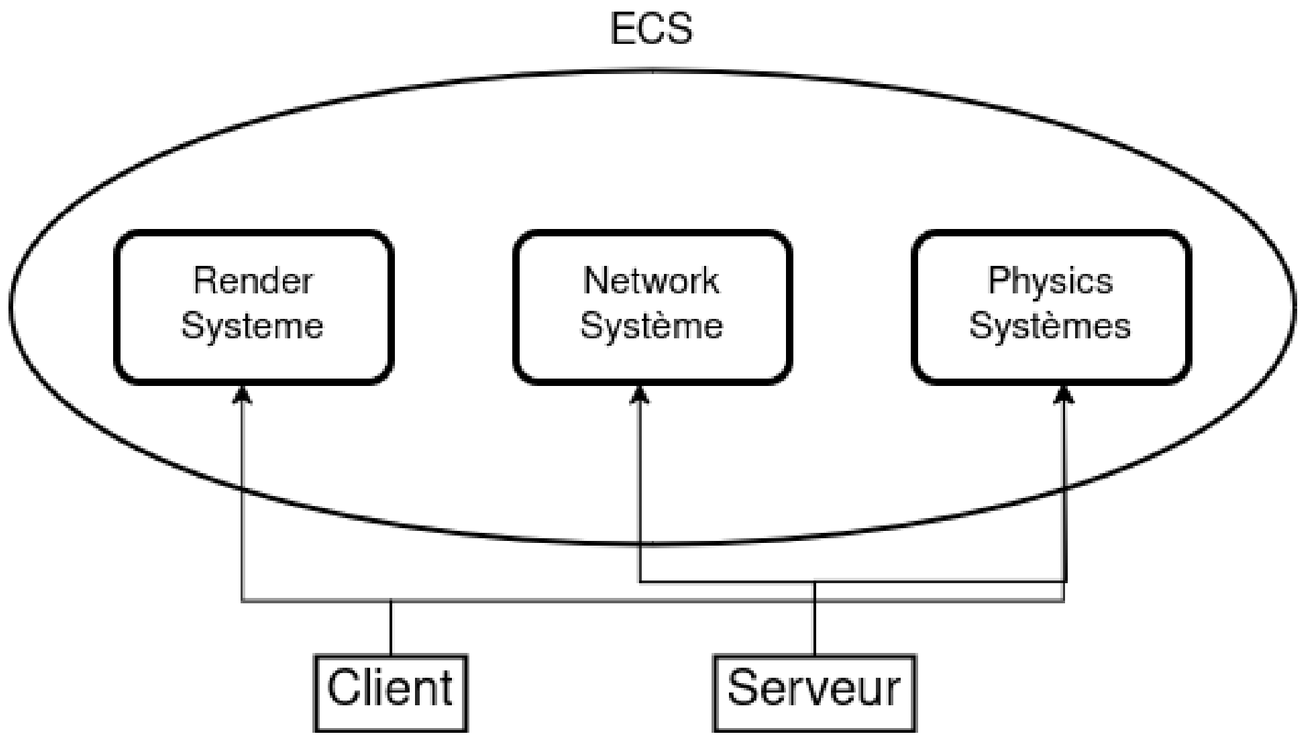
```
components/
├── component.hpp
├── components.hpp
├── graphicsComponents.hpp
├── HealthComponent.hpp
├── InputComponent.hpp
├── InputMappingComponent.hpp
├── networksComponents.hpp
├── physicsComponents.hpp
├── typeComponent.hpp
└── vectors.hpp
```

Les évènements

```
event/
├── WindowCloseEvent.hpp
└── WindowResizeEvent.hpp
```

C11 : Segmentation du code

2 - Séparation des responsabilité



C12 : Concevoir des solutions efficaces et scalables adaptées à chaque problème

1 - ECS

- Permettre l'ajout rapide et efficace de nouvelles fonctionnalités
- Concevoir une architecture modulaire, performante et maintenable

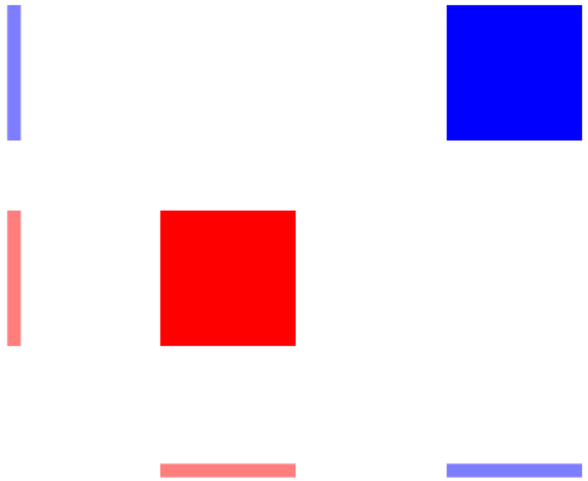
2 - Scripts de vagues d’ennemies

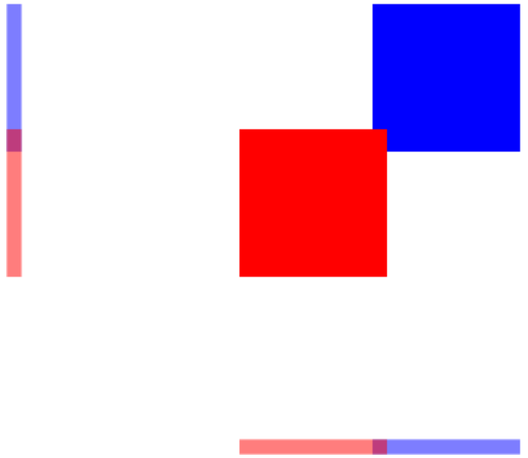
- Facilite l’implémentation de nouvelles vagues
- N’est pas directement dans l’exécutable
- Modifiable par l’utilisateur

C12 : Concevoir des solutions efficaces et scalables adaptées à chaque problème

3 - Algorithme de collision

- Utilisation des fonctions de collisions de la SDL
- Implémentation de notre propre algorithme





C13 : Choisir des solutions de persistance adaptées aux contraintes du projet

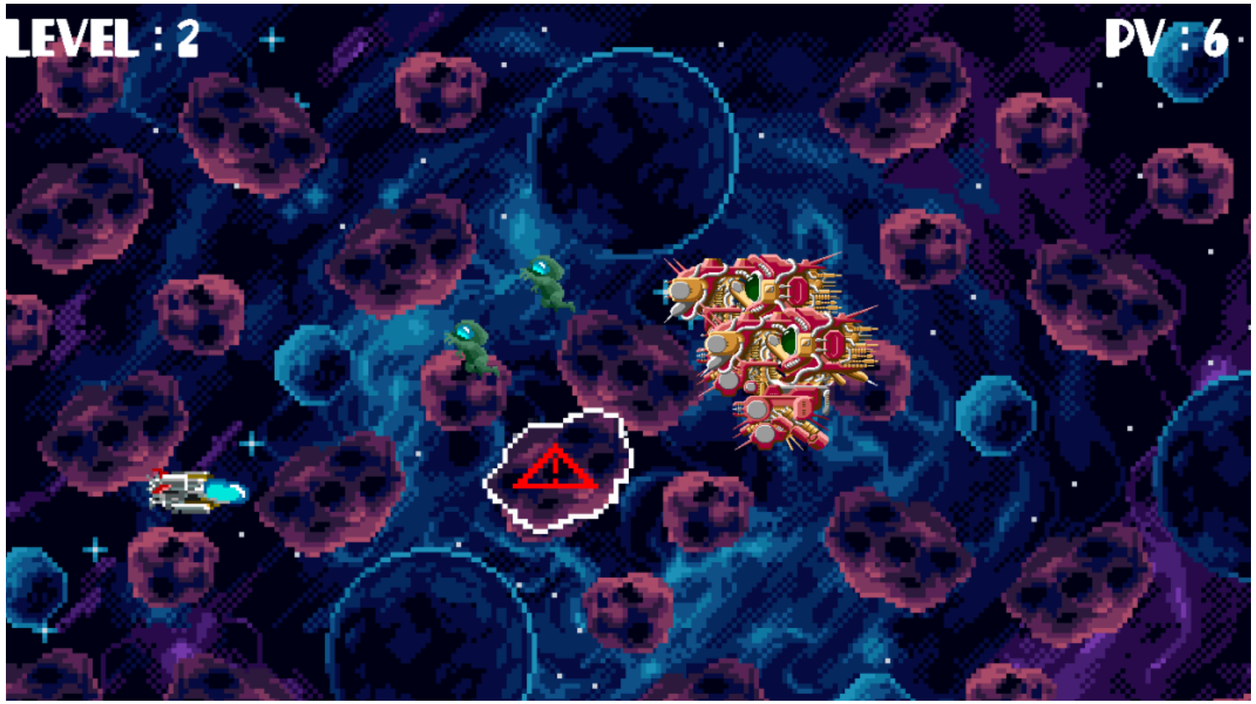
Solutions	Performance	Intégration	Scalabilité	Sécurité
Fichier texte ou binaire	Faible (I/O disque)	Simple (petites données)	Très faible	Faible (pas de contrôle natif)
Format de fichier structuré (JSON / YAML)	Moyenne (parser JSON / YAML)	Facile (petites données)	Faible	Moyenne (chiffrement manuel requis)
Base de donnée relationelle	Élevée (indexation)	Moyenne	Bonne (scalabilité verticale)	Élevée (chiffrement, contrôle d'accès)
Base de donnée non relationelle	Très élevée (sharding)	Moyenne	Excellente (scalabilité horizontale)	Bonne (authentification, chiffrement)

C14 : Choisir des structures de données optimales pour performance et évolutivité

Solutions	Performance	Maintenabilité	Scalabilité	Complexité spatiale
Vector	Accès rapide (O(1) en lecture)	Simple	Faible (redimensionnement coûteux)	Flexible (mémoire contiguë)
Ordered Map	Recherche O(log(n))	Moyenne à complexe	Bonne	Élevé (arbre équilibré)
Unordered Map	Recherche rapide O(log(1))	Moyenne à complexe	Bonne	Élevé
Bitset	Très rapide (opérations bit à bit)	Moyenne à complexe	Faible à moyenne	Très faible (optimisé pour la mémoire)

C15 : Concevoir des interfaces accessibles garantissant une bonne experience

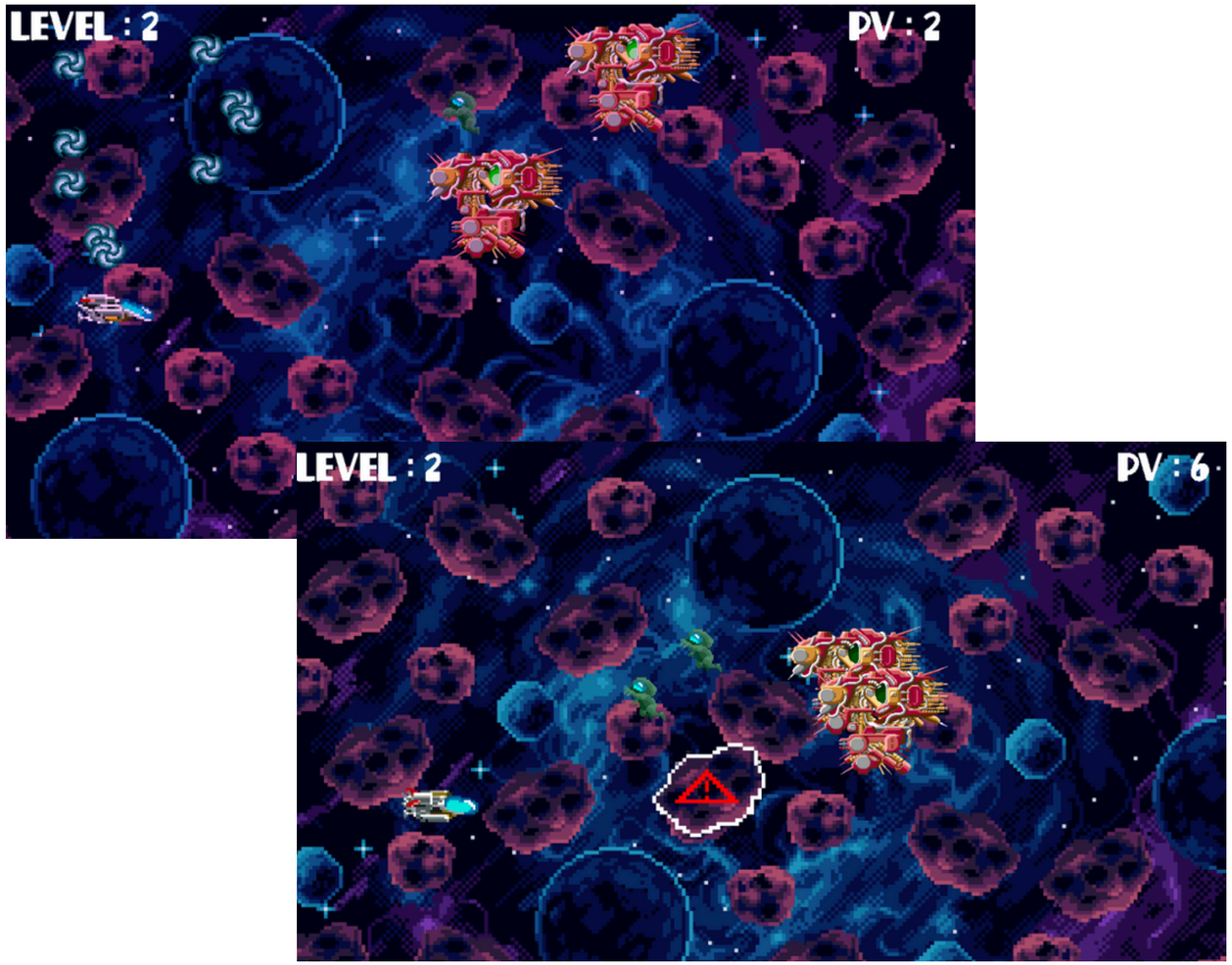
1 - interface textuelle



- Indications relatives à la partie et au joueur
  - points de vie
  - niveau
- Points d’amélioration
  - police d’écriture plus nette
  - ajout de fonds noirs pour un meilleur contraste

C15 : Concevoir des interfaces accessibles garantissant une bonne experience

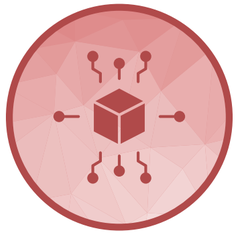
2 - interface graphique



- Informations visuelles claires
  - couleurs opposées entre les ennemies et le fond
  - mise en surbrillance d’éléments de décord interactif
  - filtres de couleurs pour les formes de daltonisme génériques (rouge, vert, bleu, contraste des couleurs)

- Points d’amélioration
  - ajout d’un menu principal et d’options

C16 : Sécurisation et Intégrité des Données



La validations strictes des paquets réseau garantissent que seules les données conformes sont traitées, évitant ainsi les corruptions ou les dysfonctionnements.

- **Autorité du serveur**
- **Interpolation, extrapolation.**
- **Validation des paquets entrants avant leur traitement**
- **L’utilisation de mécanismes tels que les UUID**
- **Gestion des erreurs et des exceptions**

ISO/IEC 27001 : Gestion de la sécurité de l'information.
ISO/IEC 29147 : Gestion des vulnérabilités.

Technologie	Vulnérabilités identifiées
ENet (bibliothèque réseau UDP)	- Absence de chiffrement natif. - Risque d’injection de données, altération des paquets.

IEC ISO

C16 : Sécurisation et Intégrité des Données

1 - Validation des commandes et autorité du serveur :

Le serveur reçoit les entrées des clients **(PACKET_INPUT)** et applique **uniquement** celles qui sont **valides**.

```
if (inputFlags & 0x01) vel.vy = -200.0f;
```

Broadcast de l’état du jeu :

M**ises à jour aux clients** via des **paquets** comme **PACKET_ENTITY_UPDATE** ou **PACKET_GAME_STATE**.
Cela corrige les erreurs ou **désynchronisations** entre les clients.

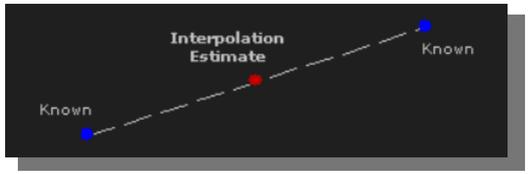
```
// Prépare les données du paquet
uint8_t buffer[1 + sizeof(uint16_t) + sizeof(int)] = {0};
buffer[0] = PACKET_GAME_STATE;  // Type de paquet
std::memcpy(buffer + 1, &currentWave, sizeof(uint16_t));
std::memcpy(buffer + 1 + sizeof(uint16_t), &playerHealth, sizeof(int));

// Envoie le paquet à tous les clients
ENetPacket* packet = enet_packet_create(buffer, sizeof(buffer), ENET_PACKET_FLAG_RELIABLE)
for (auto& peer : connectedPeers_) {
    enet_peer_send(peer, 0, packet);
}
enet_host_flush(host_);
```

C16 : Sécurisation et Intégrité des Données

2 - Interpolation et extrapolation

Interpolation :

Lissage des mouvements des entités.
Calcule des positions intermédiaires entre ces deux points.



Extrapolation :

Prédit l'état futur d'une entité en se basant sur sa vitesse et sa direction actuelles.
Si une entité est à **(x1, y1)** avec une **vitesse (vx, vy)**, on peut estimer sa position à t + dt avec :

```
x_future = x1 + vx * dt;
y_future = y1 + vy * dt;
```

C16 : Sécurisation et Intégrité des Données

3 - Validation des paquets entrants avant leur traitement

Paquet null ?

Taille suffisante ?

Paquet connu ?

```cpp
bool ServerNetworkManager::validatePacket(const ENetPacket* packet, size_t expectedSize) {
    if (!packet) {
        std::cerr << "Invalid packet: Packet is null." << std::endl;
        return false;
    }

    if (packet->dataLength < expectedSize) {
        std::cerr << "Invalid packet: Data length (" << packet->dataLength
                  << ") is smaller than expected size (" << expectedSize << ")." << std::endl;
        return false;
    }

    uint8_t packetType = packet->data[0];
    if (packetType != PACKET_PLAYER_ID &&
        packetType != PACKET_INITIAL_ACK &&
        packetType != PACKET_INPUT &&
        packetType != PACKET_ENTITY_UPDATE &&
        packetType != PACKET_ENTITY_REMOVE &&
        packetType != PACKET_GAME_STATE) {
        std::cerr << "Invalid packet: Unknown packet type " << static_cast<int>(packetType) << std::endl;
        return false;
    }

    return true;
}
```

C16 : Sécurisation et Intégrité des Données

4 - Protection contre la corruption des données

```
1    std::array<uint8_t, 16> uuidBytes = uuid::uuidStringToByteArray(playerUUID);
```

Les **UUID** utilisés pour identifier les entités garantissent l'unicité des objets,
limitant les risques de confusion ou de données corrompues.

Ils sont générés en utilisant des algorithmes prenant en compte des informations uniques comme :

- L'adresse MAC du matériel.
- L'horodatage précis.
- Un générateur pseudo-aléatoire.

Robustesse : Gestion fiable des entités dans des environnements réseau.
Évolutivité : Ajouter facilement de nouvelles entités sans craindre les conflits.
Fiabilité des communications : Les paquets réseau identifient précisément les entités concernées, même en cas de charge élevée.

C16 : Sécurisation et Intégrité des Données

5 - Possibilité d’implémentation pour renforcer la sécurité sur le réseau.

- **Chiffrement des données sensibles :** Utiliser **AES-256** pour sécuriser les échanges réseau. (Librairie OpenSSL)
- **Authentification des paquets :** Ajouter une **signature numérique** ou un **HMAC** pour valider l’origine. ***(hash-based message authentication code)***
- **Base de données chiffrée :** Protéger les données stockées avec des solutions comme **SQLCipher**.
- **Transport sécurisé :** Implémenter **TLS** pour **crypter** les communications client-serveur.
- **Journalisation sécurisée :** Chiffrer les logs contenant des informations sensibles**.**
- **Expiration des tokens :** Limiter la durée de validité des clés d’accès ou des sessions**.**

C16 : Sécurisation et Intégrité des Données

Exemple d'implementation d'un HMAC (*Hash-based Message Authentication Code*)

Fonction de hachage cryptographique

```cpp
#include <openssl/hmac.h>

std::string generateHMAC(const std::string& data, const std::string& key) {
    unsigned char* result;
    unsigned int len = 20;

    result = HMAC(EVP_sha256(), key.data(), key.size(),
                  (unsigned char*)data.data(), data.size(), nullptr, nullptr);

    return std::string((char*)result, len);
}

bool validateHMAC(const std::string& data, const std::string& receivedHMAC, const std::string& key) {
    std::string calculatedHMAC = generateHMAC(data, key);
    return calculatedHMAC == receivedHMAC;
}
```

Intégrité : Vérifie que les données n'ont pas été modifiées.
Authenticité : S'assure que les données proviennent d'une source légitime

C17 : Sélection des solutions techniques adaptées

1 - JSON

Ennemis : Les vagues d’ennemies sont configurées via des fichiers JSON, indépendants du code.

C17 : Sélection des solutions techniques adaptées

2 - Architecture ECS

Logique indépendante : Chaque système fonctionne indépendamment des autres. L’ajout d’un nouveau système est donc très facile.

Composition VS Héritage : Une entité est décrite par ses composants et non pas par des classes/héritages spécifiques.

Event Driven : Les systèmes s’exécutent en fonction des évènements postés par le jeu. Chaque évènement est indépendant.

3 - Couche réseau ENET

Gestion des connexions : ENet gère les connexions de manière persistante.

Fiabilité accrue : ENet encapsule les connexions pour garantir la livraison et l’ordre des paquets.

C18 : Rédiger le code à l'aide du langage informatique adapté au logiciel

1 - Formatage du code.

PascalCase

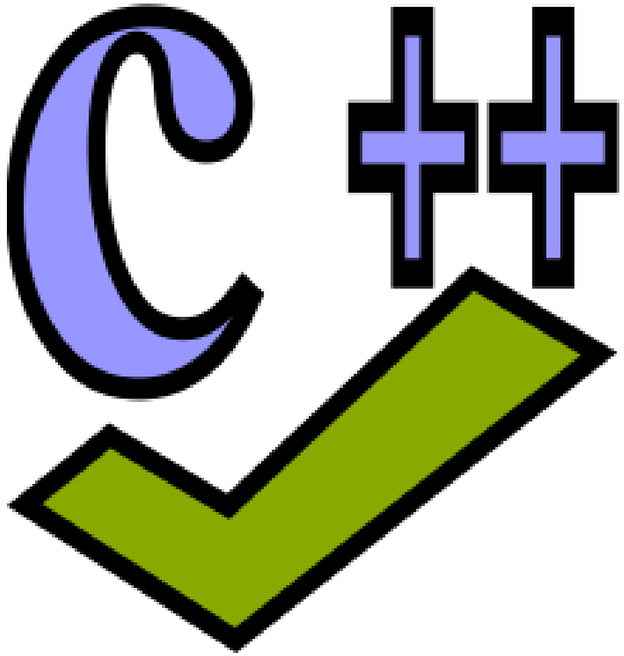
```
class AudioSystem
```

camelCase

```
void onStart() override;
void onShutdown() override;
void onUpdate(float deltaTime) override;
```

snake_case (class member)

```
std::stack<Entity::Id> _reusable_entity;
```



C18 : Rédiger le code à l'aide du langage informatique adapté au logiciel

2 - Organisation du code

```
include
├── components
│   ├── component.hpp
│   ├── components.hpp
│   ├── graphicsComponents.hpp
│   ├── HealthComponent.hpp
│   ├── InputComponent.hpp
│   ├── InputMappingComponent.hpp
│   ├── networksComponents.hpp
│   ├── physicsComponents.hpp
│   ├── typeComponent.hpp
│   └── vectors.hpp
├── event
│   ├── WindowCloseEvent.hpp
│   └── WindowResizeEvent.hpp
├── GameEngine.hpp
├── system
│   ├── AudioManager.hpp
│   ├── AudioSystem.hpp
│   ├── InputSystem.hpp
│   ├── MovementSystem.hpp
│   └── RendererSystem.hpp
├── UUID.hpp
└── Window.hpp
```

```
src
├── GameEngine.cpp
├── system
│   ├── AudioManager.cpp
│   ├── AudioSystem.cpp
│   ├── InputSystem.cpp
│   ├── MovementSystem.cpp
│   └── RendererSystem.cpp
├── UUID.cpp
└── Window.cpp
```

C18 : Rédiger le code à l'aide du langage informatique adapté au logiciel

3 - Gestion des erreurs et exceptions

```cpp
void ecs::Registry::updateEntities()
{
    auto const actions = std::move(_actions);
    _actions = decltype(_actions){};

    for (auto const &action : actions) {
        try {
            applyAction(action);
        } catch (std::exception const &e) {
            std::cerr << e.what() << std::endl;
        }
    }
}
```

throw d’erreur très présent pour facilement repérer les erreurs

C18 : Rédiger le code à l'aide du langage informatique adapté au logiciel

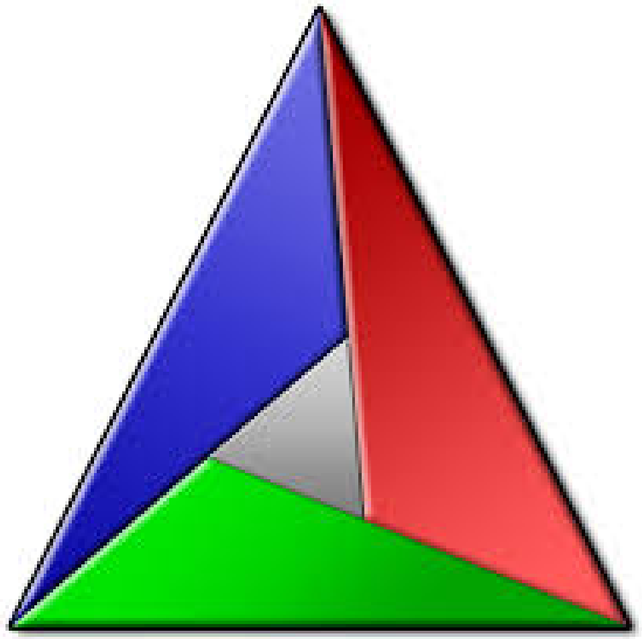
4 - Sérialisation des données

```cpp
void ServerNetworkManager::sendEntityUpdates(const std::vector<EntityState>& states) {
    const uint16_t count = static_cast<uint16_t>(states.size());

    std::vector<uint8_t> data;
    data.reserve(1 + 2 + states.size() * sizeof(EntityState));

    data.push_back(PACKET_ENTITY_UPDATE);

    auto countPtr = reinterpret_cast<const uint8_t*>(&count);
    data.insert(data.end(), countPtr, countPtr + sizeof(uint16_t));

    for (auto& st : states) {
        auto ptr = reinterpret_cast<const uint8_t*>(&st);
        data.insert(data.end(), ptr, ptr + sizeof(EntityState));
    }

    ENetPacket* packet = enet_packet_create(data.data(), data.size(), ENET_PACKET_FLAG_UNSEQUENCED);
    for (auto& peer : connectedPeers_) {
        enet_peer_send(peer, 0, packet);
    }
    enet_host_flush(host_);
}
```

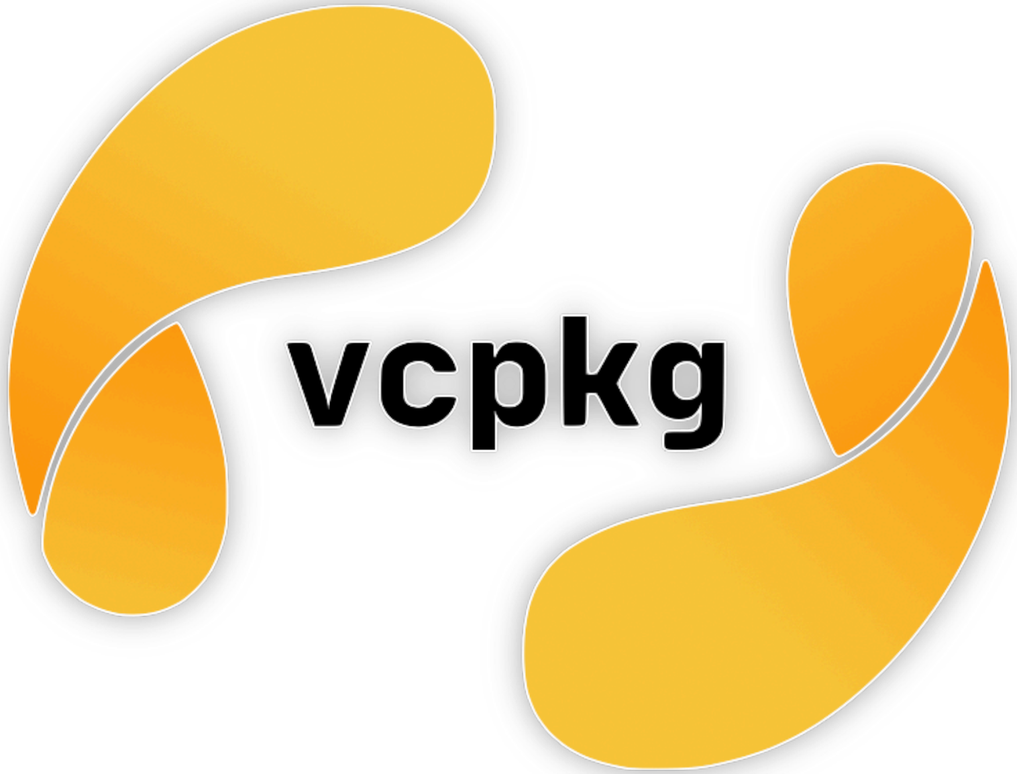
Ce qu’on aurait pu rajouter :

- Compression de la donnée
- Vérifier l’intégrité des packets

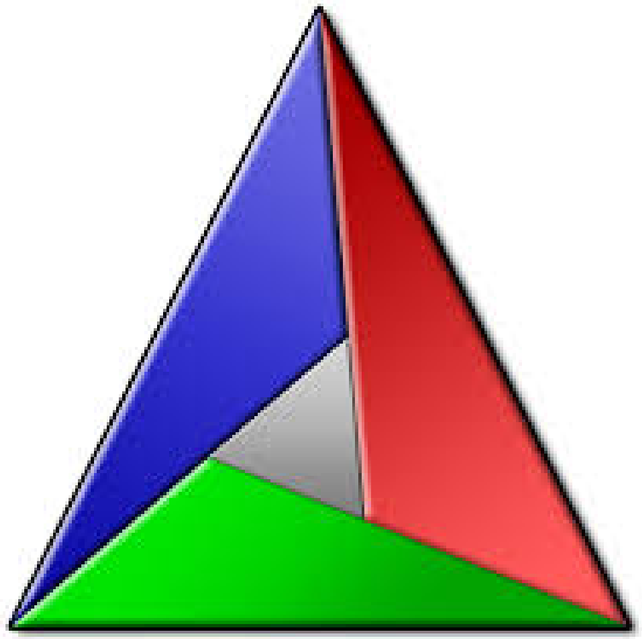
C19 : Intégrer l'usage de codes tiers au code



CMake



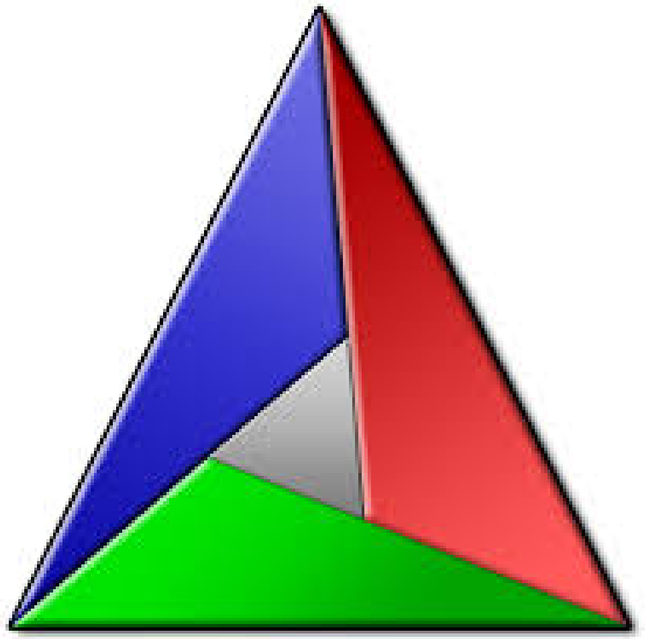
C19 : Intégrer l'usage de codes tiers au code



CMake

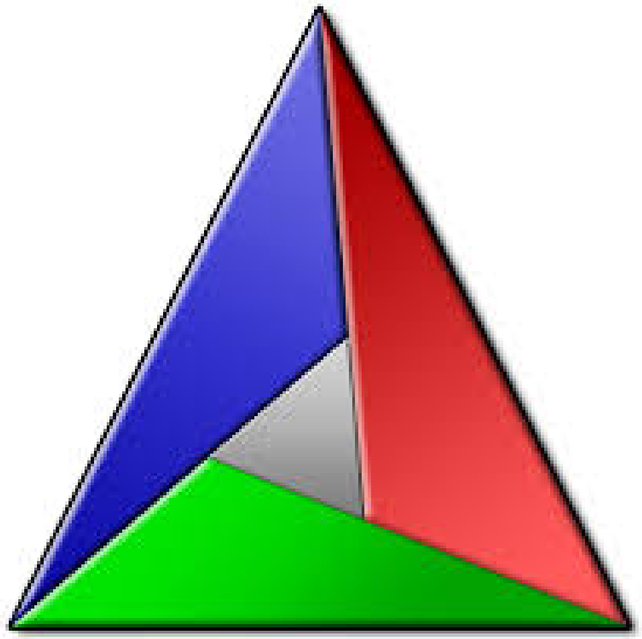
doxygen

C19 : Intégrer l'usage de codes tiers au code



CMake

C19 : Intégrer l'usage de codes tiers au code



CMake