**R-Type Documentation**

**Sommaire**

I. Engine

1. Composants et entités

2. GameEngine

3. Scène

II. Network

1. Communication

2. Traducteur

3. Protocole

**I. Engine**

1. Composants et entités

Deux types de IComponent existe, le Component et les ScriptedComponent.

**Composant de base (Component)**

Pour créer un composant il faut fournir le GameEngine auquel il sera rattaché tout au long de sa vie ainsi que l’équipe à laquelle il appartiendra. Un IComponent est composé de 5 parties majeures, toutes accessibles via des getters : ARenderer, ICollider, Feature, Transform, IBrain.

La partie ARenderer, n’est pas une partie essentielle. Elle intervient uniquement lors de l’affichage d’un IComponent, cette partie n’est donc pas existante lorsque qu’il n’y a pas de rendu graphique.

Le ICollider représente l’emplacement occupé par un IComponent dans l’espace virtuel du GameEngine, cet objet permet la détection de collision entre les composent. Si deux IComponent d’équipe différente entres en collision, ils s’endommageront. Un ICollider est généralement représentatif de la texture de son ARenderer lorsqu’il y en a un.

Toutes les informations relatives à ses caractéristiques, sont stockées dans la partie Feature. Elles sont accessibles via des getters et certaines sont modifiables via des méthodes propres à l’objet Feature.

Tous les IComponents, sont localisés grâce au Transform, il contient des informations sur sa position, sa rotation et ses variations de taille et ceux, pour chacun des 3 axes (x, y, z).

La partie IBrain est utilisé pour construire des IComponents doté d’intelligence supérieur à un script établie. Le IBrain, lorsqu’il est présent, est appelé à différent moment clef de la vie d’un IComponent, et peut agir sur le composant.

Au de-là de ses différentes parties, un IComponent possède 3 méthodes pour agir dans le GameEngine :

* doTick() : qui doit être appelé une fois par tour d’action pour permettre au composant d’effectuer des opérations tel qu’un déplacement ou une attaque.
* onColliding() : cette méthode doit être appelé lors d’une collision avec un autre IComponent.
* shoot() : lorsqu’un IComponent « tir », c’est cette méthode qui est appelé, elle créée un nouveau IComponent et l’ajoute à son GameEngine.

**Composant Scripté (ScriptedComponent)**

Les composants scripté héritent des Components et sont traités comme des IComponent dans le GameEngine. Leur particularité est qu’ils possèdent une liste d’action qu’ils effectueront lors de leur phase d’action (avec la méthode « doTick() »).

Il existe plusieurs types d’action tel que le déplacement ou encore le tir. Ces actions sont gérées par une gestionnaire de tâche qui s’assure qu’elles sont correctement executer et dans le bon ordre. Lorsqu’une tâche est terminée elle est supprimée du fil d’exécution.

Pour les différentes tâches de déplacement, elles dépendent toutes de la vitesse du IComponent, contenu dans sa partie Feature.

*set\*() = 0;*

*get\*() = 0;*

*doTick() = 0;*

*onColliding() = 0;*

*shoot() = 0;*

IComponent

set\*();

get\*();

doTick();

onColliding();

shoot();

ARender \* \_renderer;

ICollider \*\_collider;

Feature \_feature;

Transform \_transform;

IBrain \*\_brain;

Component

getScript();

doTick();

ScriptObject \_script;

ScriptedComponent

**Entités**

Pour le bon fonctionnement des scènes, plusieurs Entités peuvent être crées sont la forme de librairies dynamique implémentant une fonction précise. La fonction doit se nommer « build() » et elle doit renvoyer un PreComponent qui servira à la création d’un IComponent par la suite.

Un PreComponent apporte principalement des informations quant à la représentation graphique du composant et de ses entités produites lorsqu’il tir ainsi qu’un potentiel IBrain.

**I. Engine**

2. GameEngine

**Mise en place**

Le GameEngine est l’objet au cœur, il s’occupe de lire la scène, gérer les composants, communiquer avec d’autre GameEngine distant et aussi d’appeler un IDisplay quand il y en a un.

Pour commencer un GameEngine, peut être instancié avec ou sans IDisplay en paramètre, dans le cas où un IDisplay serait renseigné, le GameEngine l’utilisera tout au long de son fonctionnement et s’occupera de libérer la mémoire en fin d’exécution. Toujours dans le cas où un IDisplay serait renseigné, l’appel de la méthode « open() » est nécessaire pour l’affichage d’une fenêtre.

Un ICollider peut être fournis au GameEngine afin de définir une limite au-delà de laquelle les IComponents se verraient détruis. A chaque tour de temps, chaque IComponent est testé avec le ICollider, s’il n’y a pas de collision, le IComponent est détruit.

**Lancement d’une scène**

Pour lancer une scène, il suffit d’appeler la méthode « playScene() » en renseignant le chemin d’accès vers le fichier scène et une durée en milliseconde du délai minimal entre deux tours. Si le fichier n’est pas trouvable ou n’est pas utilisable, une exception est produite.

**Méthodes disponibles**

Sans lancer de scène, il est possible de manuellement intéragir avec le GameEngine, les méthodes suivantes sont utilisables à cet effet :

* turn() : Cette méthode permet de manuellement gérer les tours, à la lecture d’une scène, cette méthode est appelée à intervalle régulier en fonction du délai transmit lors du lancement de la lecture de la scène.
* addComponent() : Cette méthode permet d’ajouter un nouveau composant à la partie. Cette méthode est appelée par le gestionnaire d’acteur durant la lecture d’une scène.
* getNearestComponent() : A l’aide de cette méthode, le GameEngine, renverra la position du IComponent le plus proche de la position transmise en paramètre et correspondant à l’équipe recherchée.
* getComponentByID() : Permet de trouver un IComponent par son ID
* killComponent() : En transmettant un ID, le GameEngine va chercher le IComponent correspondant et l’ajouter sur une liste de IComponent qu’il détruira au prochain tour (sauf si vous précisez une suppression imédiate)

**IDisplay et ARenderer**

Ces deux objets sont les principaux responsables d’un rendu graphique de ce qu’il se passe dans le GameEngine. Il est possible d’implémenter ces deux objets en ajoutant une librairie graphique.

Pour assurer le bon fonctionnement du programme, il est important que le IDisplay et le ARenderer implémentent tous les deux la même librairie graphique.

Pour notre exemple actuel, ces deux objets ont été implémenter avec la librairie SFML dans les objets SfmlDisplay et SfmlRenderer.

**I. Engine**

3. Scène

**Organisation d’une scène**

Les scènes sont divisées en deux parties essentielles. Tout d’abord les acteurs, puis les actions.

La partie acteur est simplement la liste des chemins d’accès de toutes les librairies vers les entités utilisé au cours de la scène. Chaque élément de la liste représente une Entité.

La partie des actions est une liste d’objet SceneApparition. Cet objet apporte des informations sur l’acteur à utiliser, son Transform ainsi que sont script. Si aucun scripte n’est renseigné un simple Component est créé, sinon, c’est un ScriptedComponent qui est instancié et sont script est proprement défini.

**Format de fichier**

Les fichiers scènes sont écris en binaire et sont au format « .scn ». Pour ce qui est du contenu, l’ordre des informations est important.

Tout d’abord un nombre entier positif sur 4 octets, représentant le nombre d’acteur à suivre. Pour chaque acteur, le chemin d’accès vers sa librairie précédé de la taille de ce chemin.

Après la liste des acteurs, les SceneApparitions, doivent être directement écrite en binaire terminer par une SceneApparition avec un delta à -1.

SceneApparition FIN

SceneApparition 01

SceneApparitions

Fichier scène

Nombre d’acteur

Chemin d’accès 00

Chemin d’accès N

Chemin d’accès 01

SceneApparition 00

Acteurs

SceneApparition 02

SceneApparition 03

Scripte SceneApparition 03

SceneApparition 04

Scripte SceneApparition 04

SceneApparition 05

**Lecture**

La lecture d’une scène se fait à l’aide de 2 objets, un objet SceneLoader et un objet Scene.

L’objet Scene s’occupe d’interagir avec les deux parties, il charge et garde en mémoire les Entités puis lis une à une les SceneApparition contenu dans le fichier scène.

L’objet SceneLoader contrôle l’objet Scene et l’utilise pour récupérer et exécuter les SceneApparitions dans le bon ordre. Lorsqu’une SceneApparition doit être appliquée (c’est-à-dire que son délai est passé), il instancie nouvel objet, l’ajoute au GameEngine et passe au SceneApparition suivant.

**II. Network**

1. Communication

Le rtype est composé de deux exécutables : le serveur et le client.  
Tout deux sont en UDP.

**Le Serveur**

Le serveur du rtype se sépare en deux parties distinctes : la gestion des lobby et la gestion des parties en cours. Etant donné que le serveur est multi-threadé, ces deux parties fonctionnent en simultanées.

Tout d’abord, la partie gérant les lobby. Cette partie a une socket (cf boost::asio::ip::udp::socket) bind sur le port 13 sur laquelle le serveur pourra reçevoir certaines informations des clients connectés. Lors de sa première connection, le joueur enverra un message au serveur afin que ce dernier soit mis au courant de sa connection. Le serveur ajoutera donc un nouvel objet Client à sa liste de clients connectés. Une fois le client inscrit dans la liste des connections, il pourra envoyer des commandes lui permettant d’effectuer des actions sa la listes des Lobby actuels. Le client pourra donc demander au serveur de créer un lobby, d’en rejoindre un ou encore modifier son status s’il est dans un lobby. Tout cela est géré par un objet ClientManagement. Une fois que tous les clients dans un lobby sont prêts, le serveur se chargera d’envoyer le début de la game à tous les clients qui devront changer de port d’écoute afin de pouvoir jouer et reçevoir les informations de la partie lancée.

Ensuite, la partie gérant les games. Cette partie comporte deux modules : le premier permettant de reçevoir les déplacements des joueurs en écoutant sur la socket bind sur le port 14, et le second permettant de gérer les parties en cours. Le premier module s’occupera de déplacer le vaisseau du joueur en fonction du déplacement désiré (haut, bas, gauche, droite, tir). Le second quand à lui s’occupera de traduire les déplacements effectués par les vaisseaux ennemi ainsi que les intéractions ayant eu lieu (vaisseau ennemi mort, vaisseau joueur mort…). Une fois ces intéractions traduites sous forme de string, le serveur l’enverra aux clients de la parite afin qu’ils puissent mettre à jour leur partie.

**Le Client**

Le client du rtype correspond à l’interface que les utilisateurs verront. Lorsque ce dernier est exécuté, il s’occupera de démarrer la connection avec le serveur selon l’adresse et le port (le port par défaut du serveur fourni étant 13) entrés en paramètres. Une fois la connection effectuée, le client aura accès à plusieurs bouttons : Rejoindre Lobby, Créer un nouveau Lobby ou quitter le jeu.

Selon son choix (cf Protocol), le client enverra l’action effectuée au serveur qui s’occupera de son traitement et renverra une réponse au client qui verra son affichage actualisé selon la réponse du serveur.

Une fois que le client est dans un lobby et reçoit l’information que la partie commence, il se reconnectera au serveur à partir d’un nouveau port (ici le port 14) afin de pouvoir envoyer les actions que le joueur voudra effectuer ainsi que pour reçevoir les informations relatifs aux positions des vaisseaux ennemis/alliés et des tirs. Le joueur verra donc le scroller avec les informations de la partie représentées graphiquement sur son écran. Afin de pouvoir actualiser les déplacements des vaisseaux ennemis/alliés/tirs, le client du rtype reçevra régulièrement une chaine de caractère pouvant être directement traduite par son GameEngine afin d’actualiser les différentes actions réalisées au cours de la partie.

**II. Network**

2. Traducteur

**Translater et ITranslate**

Afin de permettre la communication entre deux GameEngine, chaque GameEngine possède un Translater qui permet le stockage et la sérialisation de différents événements, sous la forme de ITranslate. Le Translater mais en résonance deux GameEngine mais aussi leurs SceneLoader.

Un Translater émetteur doit tout d’abord transmettre la liste des acteurs afin que le Translater récepteur soit capable de faire apparaitre les même Entités. Une fois les deux Translater calibrés, le Translater émetteur n’a plus qu’a envoyer ses ITask à intervalle régulier.

**Sauvegarde des événements importants**

Les événements sont les suivants :

* + TransSpawn : représente l’ajout d’un IComponent au GameEngine. Cet objet véhicule l’information sur l’ID du composant au sein du GameEngine ainsi que son numéro d’acteur pour être instancié.
  + TransDeath : représente la mort d’un IComponent présent dans le GameEngine.
  + TransShoot : cet objet est créé lorsqu’un IComponent tir, à cause son script ou de son IBrain.
  + TransPosition : cet objet est le plus fréquemment envoyé, il sert à donner la position de chaque entité présente dans les GameEngine.

**Sérialisation et désérialisation**

Les ITask ont deux utilisations différentes. D’un côté, le Translater émetteur, instancie des ITask qui ont pour but de stocker des informations, et de l’autre côté, le Translater récepteur, applique les ITask afin de modifier le GameEngine.

Chaque ITask peut être sérialiser via la méthode « serialize() ». Le Translater récepteur à un rôle de factory de ITask. Tous les types d’ITask sont instanciable avec les informations à véhiculer ou avec un « std::istream » et un GameEngine.

**II. Network**

3. Protocole

**Gestion des lobby (partie non lancée)**

CLIENT : NOUVEAU CLIENT (lors de la connection uniquement)

SERVEUR : ID {ClientId}

CLIENT : LOBBY INFO

SERVEUR : {Lobby Nb} {Lobby Name} {Nb Clients in Lobby} {Game Level} {Nb Place Max In Lobby} (x Lobby Nb)

CLIENT : LOBBY NEW {Lobby Name} {Game Level} {Max Place}

SERVEUR : OK (si la création s’est bien déroulée)

ERROR (si un lobby du même nom existe déjà)

CLIENT : LOBBY JOIN {Lobby Name}

SERVEUR : OK (le client a pu rejoindre le lobby)

ERROR (le lobby est rempli ou le client est déjà dans un lobby)

CLIENT : LOBBY READY/UNREADY

SERVEUR : OK (le client a changé son status)

ERROR (le client n’est pas dans un lobby)

Si le client envoie une autre commande, le serveur répondra par UNKNOW COMMAND.

SERVEUR : GAME START

CLIENT : modification du port d’écoute

**Gestion des parties**

CLIENT : 49/50/51/52/53 (correspondant à haut, bas, gauche, droite, tir)

SERVEUR : (aucune réponse, réalise le déplacement du vaisseau joueur)

SERVEUR : string en byte contenant les mises à jours des composants présents sur la scène

CLIENT : traduction de la string pour actualiser les composants à afficher