R-TYPE

Documentation

Antoine Desruet

Alan Sigal

Thomas Tricaud

Tom Rives

EPITECH.
L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

Binary Protocol TCP

To communicate with the game-server, you must use this protocol for R-Type.

Each message is composed of a header (8 bytes), and a body of variable size which is filled in the header.

Header (8 bytes)

- Enumeration (2 bytes) : Type of message
- ResponseCode (2 bytes) : Result of the request
- BodySize (4 bytes)

Body (variable size)

- Content of response

Enumeration MessageType

```
enum class MessageType : short
{
    SetPlayerName = 0,
    CreateGame = 1,
    JoinGame = 2,
    LeaveGame = 3,
    GetGamesList = 4,
    GetPlayersInGame = 5,
    StartGame = 6,

    GameRegister = 7,
    GameCommand = 8,

    EntityUpdate = 9,
    EntityDestruction = 10,
    GameInfo = 11,
};
```

CLIENT -> SERVER

- **SetPlayerName**
    a. Size 10 bytes
        i. Username (10 bytes)
    b. Response
        i. **200** Player name is setted up
        ii. **4XX** Error
        iii. **500** Internal Error

- **CreateGame** (Create a game party)
    a. Size 12 bytes
        i. Party name (12 bytes)
    b. Response
        i. **200** Party created
        ii. **4XX** Error
        iii. **500** Internal Error

- **JoinGame** (Join a game party)
    a. Size 12 bytes
        i. Party name (12 bytes)
    b. Response
        i. **200** OK
        ii. **4XX** Error
        iii. **500** Internal Error

- **LeaveGame (Leave the game party)**
    a. Size 12 bytes
    b. Response
        i. **200** OK
        ii. **4XX** Error
        iii. **500** Internal Error

- **GetGameList (Get list of game party)**
    a. Size 13 * (number of games available) bytes
    b. Réponse
        i. **200** OK
            1. Game List
        ii. **4XX** Error
        iii. **500** Internal Error

Binary Protocol UDP

To communicate during the game party, you must use this protocol UDP for R-Type.

Same the protocol TCP, each message is composed of a header (8 bytes), and a body of variable size which is filled in the header.

Header (8 bytes)

- Enumeration Message Type (2 bytes) : Type of message
- ResponseCode (2 bytes) : Result of the request
- BodySize (4 bytes)

Body (variable size)

- Content of response

Enumeration MessageType

```
enum class MessageType : short
{
    SetPlayerName = 0,
    CreateGame = 1,
    JoinGame = 2,
    LeaveGame = 3,
    GetGamesList = 4,
    GetPlayersInGame = 5,
    StartGame = 6,

    GameRegister = 7,
    GameCommand = 8,

    EntityUpdate = 9,
    EntityDestruction = 10,
    GameInfo = 11,
};
```

CLIENT -> SERVER

- **Game Register** (the client identifies itself to the server)
    a. Size 0 bytes
- **Game Command** (the client sends an action to the server)
    a. Size 4 bytes
        i. Enumeration ControlGame (4 bytes)

SERVER -> CLIENT

- **Entity Update** (Send to the client the entities to be displayed)
    a. Size 80 bytes
        i. Structure NetworkEntityInformation (80 bytes)
- **Entity Destruction** (Send to the client the ID of the entity to be destroyed)
    a. Size 4 bytes
        i. ID entity (4 bytes)
- **Game Info** (Send to the client some informations about the game party: *Round, Score*)
    a. Size 8 bytes
        i. Number of rounds (4 bytes)
        ii. Score (4 bytes)

Enumeration ControlGame

```
enum ControlGame: int {
    UP,
    DOWN,
    LEFT,
    RIGHT,
    SPACE,
    ESCAPE,
    ENTER,
    DELETE,
    NONE,
};
```

Structure NetworkEntityInformation

```
struct NetworkEntityInformation
{
    std::size_t entity;
    Position position;
    Acceleration acceleration;
    Speed speed;
    Color color;
    Rotate rotate;
    Texture textureType;
    Scale scale;
    std::size_t totalTextureRect;
    float animationSpeed;
    sf::IntRect textureRect;
};
```

Interface INetwork

This class allows you to implement the network part for your R-Type project.
These methods present in this Interface must be implement in your Class Network.

```
class INetwork {
    public:
        virtual ~INetwork() = default;
        virtual void sendMessage(Message<MessageType> &message) = 0;
        virtual void readMessageHeader() = 0;
        virtual void readMessageBody() = 0;
        virtual void writeMessageHeader(Message<MessageType> &message) = 0;
        virtual void writeMessageBody(Message<MessageType> &message) = 0;
};
```

- **sendMessage** : This function aims to send a message to destinate.
- **readMessageHeader** : This function aims to get the header of the message.
- **readMessageBody** : This function aims to get the content of the message.
- **writeMessageHeader** : This function aims to send the header of your message.
- **writeMessageBody** : This function aims to send the content of your message.