```
<R-TYPE>                                                     <A. Moros>
Protocol                                                      <Epitech>
Intended status: <Start>
```

<R-TYPE>
<R-Type protocol RFC>

Status of this Memo

Copyright Notice

Abstract

<This document is here to provide you a description of the internet protocol use in the R-TYPE project. R-Type is a famous shoot-em up space game. But we make it interesting by add internet multiplayer part to the game. And this is the description of the internet package that transit.>

Table of Contents

1. Introduction..3
2. Communication...4
   2.1. Pre-Game...5
      2.1.1. Create account..5
      2.1.2. Login...6
      2.1.3. Get game list...6
      2.1.4. Create game...7
      2.1.5. Join game...7
   2.2. In-Game..7
      2.2.1. Send World..7
      2.2.2. Move..9
      2.2.3. Fire..9
   2.3. Response...9
3. Authors' Addresses...10

1. Introduction

R-type is a famous game. For the r-type epitech style we have to put some online multiplayer games. The game are made buy a maximum of four simultaneously players.
this RFC will provides an exemple of how to use the communication between the client and the server.
for that we can split the protocol in two part. The first one is the “pre-game” part. The second part is the “in-game” part.

<little figure of client/server>

```
   +------------------------------------------------+
   |                                                |
   |                                                |
   |                                                |
   |               +-------------+                  |
   |               |             |                  |
   |               |   SERVER    |                  |
   |               |             |                  |
   |               +-------------+---+              |
   |              / |         |       \             |
   |             /  |         |        \            |
   |            /   |         |         \           |
   |  +--------+  +--------+  +--------+ +--------+ |
   |  | CLIENT |  | CLIENT |  | CLIENT | | CLIENT | |
   |  +--------+  +--------+  +--------+ +--------+ |
   |                                                |
   |                                                |
   |                                                |
   +------------------------------------------------+
```

2. Communication

All the package will be send with the following header :

```
typedef struct

{

  int id;

  int size;

}  package;
```

Where id represents the type of command and size the size of the following parameter.

The List of id is the following one:

```
enum type_cmd

{

   REGISTER,

   LOGIN,

   GET_GAME_LIST,

   GAME_LIST,

   JOIN_GAME,

   CREATE_GAME,

   SEND_WORLD,

   MOVE,

   FIRE,

   RESPONSE

};
```

The header and the parameter have to be sent at once.

All this commands will be answered by the server with the struct :

```
typedef struct

{

  int response;

}  response;
```

Fill with the respond status of the previous command.

2.1. Pre-Game

There is a lot of pre-game communication process. This part will provide an exhaustive explication of this commands.

This communication is TCP/IP because you have to know if the command had been successful.

2.1.1. Create account

This command will create a new account on the server. You can connect to your account anytime with the same login and password.

To register you need to fill properly yhis structure and send it to server :

```
typedef  struct

{

  char   login[50];

  char   passwd[50];

}  create_account;
```

2.1.2. Login

This command is similar to the create account but it will just connect a client to a previously created account.

```
typedef  struct

{

  char   login[50];

  char   passwd[50];

}  login;
```

2.1.3. Get game list

To have a resume of the currently played game on the server the client has to send the header structure with the GET_GAME_LIST_CODE.

As a response, the Server will send a GAME_LIST header followed this structures (don’t forget to send all of these at once):

```
typedef  struct

{

  int nb_parties;

}  parties;


typedef  struct

{

  char   partyName[50];

  int nb_players;

}  party;
```

2.1.4. Create game

You can create a new game for you and your friends. For that you had to send the struct:

```
typedef  struct

{

  char   party_name[50];

}  create_game;
```

2.1.5. Join game

Once you see a game you want to join you simply have to send the following structure fill with the name of the game wanted :

```
typedef  struct

{

  char   party_name[50];

}  join_game;
```

2.2. In-Game

The communication in game will be an UDP protocol.

The header is the same that the in-game protocol.

2.2.1. Send World

The serveur will send regulary the world to the client.

This package is compose by two type of structure.

The first one is here to describe the world :

```
typedef  struct

{

  int nb_monstre;

  int nb_obstacle;

  int nb_ship;

}   monde_param;
```

As you can imagine, the amount of monster, obstacle and ship on the current world will be fill in nb_monster, nb_obstacle and nb_ship.

The second structure will be added to the package as many time as the previously defined number in “monde_param”.

For exemple if the world containe two monsters, two obstacle and one ship the structure will be followed by (2+2+1) 5 strucutre to describe them.

Here is the describe structure :

```
typedef  struct

{

  int          id;

  int          type;

  std::pair<float x, float y> position;

}  drawable;
```

2.2.2. Move

You have to send move command whenever you wand the ship to change position. The structure is simple :

```
typedef  struct

{

  std::pair<float x, float y> velocity

}  move;
```

2.2.3. Fire

As a good shoot-em up you will to blast a lots of enemy. And for that you will need to simply send a header with FIRE code.

2.3. Response

Whenever the server will received a cmd he will respond. And he got a strucur for that :

```
typedef struct

{

  int response;

}  response;
```

The int respond will contain the type of return. The different type of return are contain on the following enum :

```
enum reponse_type

{

   OK,

   INVALID_LOGIN,
   INVALID_PASSWORD,

   INVALID_GAME_NAME,

   CANT_JOIN_GAME

};
```

3. Authors' Addresses

<Alexandre> <Moros>
<Epitech>
<Dagobah>

Email: alexander.moros@epitech.eu

<Thiery> <Berger>
<Epitech>
<Nowhere>

Phone: <optional>
Email: thiery.berger@epitech.eu

<<Les autres se rajouterons.>>