```cpp
#include "Protocol.h"
#include <cstring>
#include <string>

Protocol::Protocol(Socket&& socket) : socket(std::move(socket)){}

Protocol::Protocol(Protocol&& other) : socket(std::move(other.socket)) {}

Protocol::~Protocol() {}

void Protocol::sendBuffer(Buffer &buffer) {
    uint32_t len_converted = htonl(buffer.getSize());
    this->socket.sendData(&len_converted, sizeof len_converted);
    this->socket.sendData(buffer.getPointer(), buffer.getSize());
}

Buffer Protocol::receiveBuffer() {
    uint32_t len;
    this->socket.receive(&len, sizeof(uint32_t));
    len = ntohl(len);

    Buffer buffer(len);
    this->socket.receive(buffer.getPointer(), len);
    return std::move(buffer);
}

void Protocol::sendIntBuffer(Buffer &buffer, int32_t value) {
    value = htonl(value);
    std::memcpy(buffer.getPointer() + buffer.getSize(), &value, sizeof(value));
    buffer.incrementOffset(sizeof(value));
}

int Protocol::receiveIntBuffer(Buffer &buffer) {
    int32_t value;
    std::memcpy(&value, buffer.getPointer() + buffer.getSize(), sizeof(value));
    buffer.incrementOffset(sizeof(value));
    return ntohl(value);
}

void Protocol::sendStringBuffer(Buffer &buffer, const std::string &string) {
    for (size_t j = 0; j < string.size(); j++){
        buffer.setNext(string[j]);
    }
    buffer.setNext('\0');
}

std::string Protocol::receiveStringBuffer(Buffer &buffer) {
    std::string string;
    char c;
    while ((c = buffer.getNext()) != '\0'){
        string += c;
    }
    return string;
}

void Protocol::sendLength(uint32_t length){
    uint32_t converted = htonl(length);
    this->socket.sendData(&converted, sizeof(uint32_t));
}

Buffer Protocol::sendLengthBuffer(uint32_t length){
    Buffer buffer;
    Protocol::sendIntBuffer(buffer, length);
    return buffer;
}

```



```cpp
size_t Protocol::receiveLength(){
    int32_t length;
    this->socket.receive(&length, sizeof(int32_t));
    return ntohl(length);
}

void Protocol::stop(){
    this->socket.stop();
}

void Protocol::sendString(const std::string &string){
    size_t string_length = string.size();
    this->sendLength(string_length);
    this->socket.sendData(string.c_str(), string_length);
}

std::string Protocol::receiveString(){
    uint32_t length = this->receiveLength();
    char buffer [MAX_STRING_SIZE + 1];
    this->socket.receive(buffer, length);
    buffer[length] = '\0';
    return std::move(std::string(buffer));
}

void Protocol::sendChar(unsigned char c){
    this->socket.sendData(&c, sizeof(unsigned char));
}

unsigned char Protocol::receiveChar(){
    unsigned char c;
    this->socket.receive(&c, sizeof(unsigned char));
    return c;
}

```

```
1   #ifndef __PROTOCOL_H__
2   #define __PROTOCOL_H__
3
4   #include <string>
5   #include "Buffer.h"
6   #include "Socket.h"
7
8   #define MAX_STRING_SIZE 200
9
10  #define CREATE_GAME_ACTION 0
11  #define JOIN_GAME_ACTION 1
12  #define START_GAME_ACTION 2
13
14  #define MOVING_OBJECT 0
15  #define DEAD_OBJECT 1
16  #define ACTION 2
17
18
19  #define START_TURN 3
20  #define END_TURN 4
21  #define MOVE_ACTION 5
22  #define CHANGE_WEAPON_ACTION 6
23  #define SHOOT_WEAPON_ACTION 7
24  #define SHOOT_WEAPON 8
25  #define SHOOT_SELF_DIRECTED 9
26  #define MOVE_SCOPE 10
27
28  #define END_GAME 11
29
30  #define MOVE_RIGHT 1
31  #define MOVE_LEFT -1
32  #define JUMP 2
33  #define ROLLBACK 3
34
35  #define WORM_TYPE 0
36  #define WEAPON_TYPE 1
37  #define GIRDER_TYPE 2
38
39  /* Clase que se encarga de enviar y recibir mensajes por socket
40   * utilizando un formato determinado */
41  class Protocol {
42      private:
43          Socket socket;
44
45      public:
46          /* Constructor */
47          explicit Protocol(Socket&& socket);
48
49          /* Constructor por copia */
50          Protocol(Protocol&& other);
51
52          /* Destructor */
53          ~Protocol();
54
55          /* Envia el contenido del buffer */
56          virtual void sendBuffer(Buffer &buffer);
57
58          /* Recibe un mensaje, lo almacena en un buffer y lo retorna */
59          Buffer receiveBuffer();
60
61
62          /* Agrega el valor al buffer cumpliendo las caracteristicas del protocol
    o */
63          static void sendIntBuffer(Buffer &buffer, int32_t value);
64
65          /* Retorna el valor del entero recibido almacenado en el buffer */
```



```
66          static int receiveIntBuffer(Buffer &buffer);
67
68
69          /* Almacena el string en el buffer  */
70          static void sendStringBuffer(Buffer &buffer, const std::string &string);
71
72          /* Retorna el string recibido que se encuentra almacenado en el buffer *
    /
73          static std::string receiveStringBuffer(Buffer &buffer);
74
75
76          /* Envia la longitud */
77          void sendLength(uint32_t length);
78
79          /* Envia la longitud */
80          static Buffer sendLengthBuffer(uint32_t length);
81
82          /* Recibe la longitud y la retorna */
83          size_t receiveLength();
84
85          /* Cierra la comunicacion */
86          void stop();
87
88          /* Envia un char */
89          void sendChar(unsigned char c);
90
91          /* Recibe un char */
92          unsigned char receiveChar();
93
94          /* Envia un string */
95          void sendString(const std::string &string);
96
97          /* Recibe un string */
98          std::string receiveString();
99  };
100
101 #endif
```

jun 09, 18 19:03 **Socket.cpp** Page 1/3

```cpp
#include <stdexcept>
#include <cstring>
#include "Socket.h"

#define SERVER 0
#define CLIENT 1

Socket::Socket(Socket::Client client) : fd(-1) {
    Addrinfo addrinfo(client.getHost(), client.getService(), 0);
    *this = addrinfo.connectOrBind(CLIENT);
}

Socket::Socket(Socket::Server server) : fd(-1) {
    char* host = NULL; //ANY
    int flag = AI_PASSIVE;
    Addrinfo addrinfo(host, server.getService(), flag);
    Socket sckt = addrinfo.connectOrBind(SERVER);

    if ((listen(sckt.fd, server.getMaxClientWait())) == -1){
        throw SocketException("Error en listen");
    }

    *this = std::move(sckt);
}

Socket::Socket(int fd): fd(fd){}

Socket::~Socket(){
    this->stop();
}

Socket::Socket(Socket&& other): fd(other.fd){
    other.fd = -1;
}

Socket& Socket::operator=(Socket&& other){
    if (this != &other) {
        this->fd = other.fd;
        other.fd = -1;
    }
    return *this;
}

int Socket::sendData(const void *data, size_t size){
    size_t total_send = 0;
    int actual_send;

    while (total_send < size){
        int len = size - total_send;
        actual_send = send(this->fd, (char*)data + total_send, len, MSG_NOSIGNAL
);
        if (actual_send < 0){
            throw SocketException("Error en send");
        }
        if (actual_send == 0){
            throw SocketException("Error en send: socket cerrado");
        }
        total_send += actual_send;
    }
    return total_send;
}

int Socket::receive(void* buffer, size_t size){
    size_t total_recv = 0;
    int actual_recv;

```

jun 09, 18 19:03 **Socket.cpp** Page 2/3

```cpp
    while (total_recv < size){
        int len = size - total_recv;
        actual_recv = recv(this->fd, (char*)buffer + total_recv, len, MSG_NOSIGN
AL);
        if (actual_recv < 0){
            throw SocketException("Error en receive");
        }
        if (actual_recv == 0){
            throw SocketException("Error en recv: socket cerrado");
        }
        total_recv += actual_recv;
    }
    return total_recv;
}

Socket Socket::acceptClient(){
    int client = accept(this->fd, NULL, NULL);
    if (client == -1){
        throw SocketException("Error en accept");
    }
    return std::move(Socket(client));
}

void Socket::stop(){
    if (this->fd != -1){
        shutdown(this->fd, SHUT_RDWR);
        close(this->fd);
        this->fd = -1;
    }
}

Socket::Addrinfo::Addrinfo(const char* host, const char* service, int flag){
    //Configuracion para getaddrinfo
    struct addrinfo hints;
    std::memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;  // IPv4
    hints.ai_socktype = SOCK_STREAM;  // TCP
    hints.ai_flags = flag;

    if ((getaddrinfo(host, service, &hints, &this->addrinfo)) != 0){
        throw SocketException("Error en getaddrinfo");
    }
}

Socket Socket::Addrinfo::connectOrBind(int action) const{
    int conection = -1;
    struct addrinfo* res = this->addrinfo;

    while (res != NULL){
        //Recorro todas las direciones posibles hasta que se conecte a una
        Socket sckt(socket(res->ai_family, res->ai_socktype, res->ai_protocol));
        if (sckt.fd != -1){
            if (action == SERVER){
                // Activo la opcion de Reusar la Direccion en caso de que esta
                // no este disponible por un TIME_WAIT
                int val = 1;
                int opt = setsockopt(sckt.fd, SOL_SOCKET, SO_REUSEADDR, &val, si
zeof(val));
                if (opt != -1){
                    conection = bind(sckt.fd, res->ai_addr, res->ai_addrlen);
                }
            } else if (action == CLIENT){
                conection = connect(sckt.fd, res->ai_addr, res->ai_addrlen);
            }
            if (conection != -1){
                return std::move(sckt);  //Conexion correcta
```

jun 09, 18 19:03 **Socket.cpp** Page 3/3

```
130                 }
131             }
132             res = res->ai_next;
133         }
134
135         throw SocketException("El socket no pudo conectarse");
136     }
137
138     Socket::Addrinfo::~Addrinfo(){
139         freeaddrinfo(this->addrinfo);
140     }
141
142     Socket::Server::Server(const char* service, int max_client_wait):
143         service(service), max_client_wait(max_client_wait){}
144
145     Socket::Server::~Server(){}
146
147     const char* Socket::Server::getService() const{
148         return this->service;
149     }
150
151     int Socket::Server::getMaxClientWait() const{
152         return this->max_client_wait;
153     }
154
155     Socket::Client::Client(const char* host, const char* service):
156         host(host), service(service){}
157
158     Socket::Client::~Client(){}
159
160     const char* Socket::Client::getHost() const{
161         return this->host;
162     }
163
164     const char* Socket::Client::getService() const{
165         return this->service;
166     }
```

jun 09, 18 19:03 **SocketException.cpp** Page 1/1

```
1   #include "SocketException.h"
2   #include <string>
3
4   SocketException::SocketException(std::string msg): msg(msg){
5       this->msg.insert(0, "Error en el socket: ");
6   }
7
8   SocketException::~SocketException(){}
9
10  const char* SocketException::what() const noexcept{
11      return this->msg.c_str();
12  }
```

jun 09, 18 19:03 **SocketException.h** Page 1/1

```
#ifndef __SOCKETEXCEPTION_H__
#define __SOCKETEXCEPTION_H__

#include <exception>
#include <string>

class SocketException: public std::exception{
    private:
        std::string msg;

    public:
        //Crea la excepcion
        explicit SocketException(std::string msg);

        //Destruye la excepcion
        virtual ~SocketException();

        //Devuelve el mensaje de error
        virtual const char* what() const noexcept;
};

#endif
```

jun 09, 18 19:03 **Socket.h** Page 1/2

```
#ifndef __SOCKET_H__
#define __SOCKET_H__

#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>
#include "SocketException.h"

class Socket{
    private:
        int fd;

        /* Constructor privado del socket */
        explicit Socket(int fd);

        /* Clase privada para encapsular getaddrinfo y hacerlo RAII */
        class Addrinfo;

    public:
        /* Clase que encapsula los parametros que se deben recibir del servidor
*/
        class Server;

        /* Clase que encapsula los parametros que se deben recibir del cliente *
/
        class Client;

        /* Constructor del socket para el cliente
         * Crea el socket y lo conecta
         * Lanza SocketException si ocurre un error */
        explicit Socket(Socket::Client client);

        /* Constructor del socket para el servidor
         * Crea el socket y lo asocia al puerto inidicado
         * Lanza SocketException si ocurre un error */
        explicit Socket(Socket::Server server);

        /* Envia todos los datos del buffer data a traves del socket
         * Devuelve la cantidad de datos que se pudieron enviar
         * Lanza SocketException en caso de error */
        int sendData(const void *data, size_t size);

        /* Recibe datos del socket hasta alcanzar el max
         * o hasta que se cierra la conexion
         * Devuelve la cantidad de datos que se pudieron recibir
         * Lanza SocketException en caso de error */
        int receive(void* buffer, size_t size);

        /* Establece una conexion con un cliente */
        Socket acceptClient();

        /* Interrumpe las conexiones del socket */
        void stop();

        /* Cierra y destruye el socket */
        ~Socket();

        /* Constructor y pasaje por movimiento */
        Socket(Socket&& other);
        Socket& operator=(Socket&& other);

        /* Bloqueo la copia */
        Socket(Socket& other) = delete;
        bool operator=(Socket& other) = delete;
};
```

jun 09, 18 19:03 **Socket.h** Page 2/2

```
65
66  class Socket::Addrinfo{
67      private:
68          struct addrinfo* addrinfo;
69
70      public:
71          /* Constructor */
72          Addrinfo(const char* host, const char* service, int flag);
73
74          /* Crea el socket y lo conecta o bindea segun correponda
75           * Devuelve el socket creado
76           * y lanza una excepcion si no pudo crearse */
77          Socket connectOrBind(int action) const;
78
79          /* Destructor */
80          ~Addrinfo();
81  };
82
83  class Socket::Server{
84      private:
85          const char* service;
86          int max_client_wait;
87
88      public:
89          /* Constructor */
90          Server(const char* service, int max_client_wait);
91
92          /* Destructor */
93          ~Server();
94
95          /* Devuelve el servicio */
96          const char* getService() const;
97
98          /* Devuelve la cantidad maxima de clientes en espera */
99          int getMaxClientWait() const;
100 };
101
102 class Socket::Client{
103     private:
104         const char* host;
105         const char* service;
106
107     public:
108         /* Constructor */
109         Client(const char* host, const char* service);
110
111         /* Destructor */
112         ~Client();
113
114         /* Devuelve el Host */
115         const char* getHost() const;
116
117         /* Devuelve el servicio */
118         const char* getService() const;
119 };
120
121 #endif
```

may 29, 18 23:38 **Thread.cpp** Page 1/1

```
1   #include "Thread.h"
2
3   Thread::Thread(): running(false){}
4
5   Thread::~Thread(){}
6
7   void Thread::start(){
8       this->running = true;
9       this->thread = std::thread(&Thread::run, this);
10  }
11
12  void Thread::join(){
13      if(this->thread.joinable()){
14          this->thread.join();
15      }
16  }
17
18  bool Thread::isRunning() const{
19      return this->running;
20  }
21
22  void Thread::stop(){
23      this->running = false;
24  }
```

may 29, 18 23:38 **Thread.h** Page 1/1

```
#ifndef __THREAD_H__
#define __THREAD_H__

#include <thread>

class Thread{
    private:
        std::thread thread;

    protected:
        bool running;

    public:
        /* Constructor */
        Thread();

        /* Destructor */
        virtual ~Thread();

        /* Constructor por copia */
        Thread(const Thread&) = delete;

        /* Operador = por copia */
        Thread& operator=(const Thread&) = delete;

        /* Constructor por movimiento */
        Thread(Thread&& other) = delete;

        /* Operador = por movimiento */
        Thread& operator=(Thread&& other) = delete;


        /* Inicia la ejecucion del thread */
        void start();

        /* Hace join con el thread */
        void join();

        /* Devuelve true si el thread esta ejecutandose,
         * false si ya termino*/
        bool isRunning() const;

        /* Metodo de ejecucion del thread */
        virtual void run() = 0;

        /* Termina abruptamente la ejecucion del thread */
        virtual void stop();
};

#endif
```

jun 19, 18 13:45 **Buffer.cpp** Page 1/1

```
#include "Buffer.h"


Buffer::Buffer(size_t max_size): buffer(new char[max_size]),
    offset(0), max_size(max_size){}

Buffer::~Buffer(){
    if (this->buffer){
        delete[] this->buffer;
    }
}

Buffer::Buffer(const Buffer& other): buffer(new char[other.max_size]){
    for (size_t i = 0; i < other.max_size; i++){
        this->buffer[i] = other.buffer[i];
    }
    this->offset = other.offset;
    this->max_size = other.max_size;
}

Buffer::Buffer(Buffer&& other): buffer(other.buffer),
    offset(other.offset), max_size(other.max_size){
    other.buffer = NULL;
}

Buffer& Buffer::operator=(Buffer&& other){
    if (this->buffer){
        delete[] this->buffer;
    }
    this->buffer = other.buffer;
    other.buffer = NULL;
    this->offset = other.offset;
    this->max_size = other.max_size;
    return *this;
}

void Buffer::setNext(char value){
    this->buffer[this->offset++] = value;
}

char Buffer::getNext(){
    return this->buffer[this->offset++];
}

char* Buffer::getPointer(){
    return this->buffer;
}

void Buffer::incrementOffset(size_t value){
    this->offset += value;
}

size_t Buffer::getSize() const{
    return this->offset;
}

size_t Buffer::getMaxSize() const{
    return this->max_size;
}
```

jun 19, 18 13:43 **Buffer.h** Page 1/1

```
#ifndef __BUFFER_H__
#define __BUFFER_H__

#include <cstddef>

#define MAX_BUF_LEN 200

/* Clase que representa un buffer de almacenamiento de datos */
class Buffer{
    private:
        char* buffer;
        size_t offset;
        size_t max_size;

    public:
        /* Constructor */
        explicit Buffer(size_t max_size = MAX_BUF_LEN);

        /* Destructor */
        ~Buffer();

        /* Constructor por copia */
        Buffer(const Buffer& other);

        /* Operador = por copia */
        Buffer& operator=(const Buffer& other) = delete;

        /* Constructor por movimiento */
        Buffer(Buffer&& other);

        /* Operador = por movimiento */
        Buffer& operator=(Buffer&& other);

        /* Agrega el valor al buffer */
        void setNext(char value);

        /* Devuelve el siguiente elemento del buffer */
        char getNext();

        /* Devuelve un puntero al buffer */
        char* getPointer();

        /* Incrementa el valor del offset */
        void incrementOffset(size_t value);

        /* Devuelve el tamaÃ±o del buffer */
        size_t getSize() const;

        /* Devuelve el maximo tamanio del bufffer */
        size_t getMaxSize() const;
};

#endif
```

jun 12, 18 14:03 **Math.cpp** Page 1/1

```
#include "Math.h"
#include <cmath>

const float PI = 3.14159265;

float Math::degreesToRadians(int angle){
    return angle * PI / 180;
}

int Math::radiansToDegrees(float angle){
    return angle * 180 / PI;
}

float Math::cosDegrees(int angle){
    return cos(Math::degreesToRadians(angle));
}

float Math::sinDegrees(int angle){
    return sin(Math::degreesToRadians(angle));
}
```

may 29, 18 23:38 **Math.h** Page 1/1

```
#ifndef __MATHUTILS_H__
#define __MATHUTILS_H__

class Math{
    public:
        /* Transforma grados geometricos a radianes */
        static float degreesToRadians(int angle);

        /* Transforma radianes a grados geometricos */
        static int radiansToDegrees(float angle);

        /* Devuelve el resultado del coseno para el angulo
         * en grados geometricos */
        static float cosDegrees(int angle);

        /* Devuelve el resultado del seno para el angulo
         * en grados geometricos */
        static float sinDegrees(int angle);
};

#endif
```

jun 12, 18 14:03 **ConfigFields.h** Page 1/2

```
#ifndef __CONFIGFIELDS_H__
#define __CONFIGFIELDS_H__

#include <string>

//config del server

const std::string SERVER_PORT("port");

const std::string DATA_SENDER_SLEEP("data_sender_sleep");

const std::string GAME_WAIT_WORLD_SLEEP("game_waiting_world_sleep");

const std::string WORLD_SLEEP_AFTER_STEP("world_sleep_after_step");

const std::string WORLD_TIME_STEP("world_time_step");

const std::string TURN_TIME("turn_time");

const std::string TIME_AFTER_SHOOT("time_after_shoot");

const std::string WORMS_LIFE_TO_ADD("worms_life_to_add");

const std::string WORM_VELOCITY("worm_velocity");

const std::string WORM_EXPLOSION_VELOCITY("worm_explosion_velocity");

const std::string WORM_JUMP_VELOCITY("worm_jump_velocity");

const std::string WORM_ROLLBACK_VELOCITY("worm_rollback_velocity");

const std::string WORM_JUMP_HEIGHT("worm_jump_height");

const std::string WORM_ROLLBACK_HEIGHT("worm_rollback_height");

const std::string WEAPONS_VELOCITY("weapons_velocity");

const std::string WEAPON_DAMAGE("weapon_damage");

const std::string WEAPON_RADIUS("weapon_radius");

const std::string WEAPON_FRAGMENTS("weapon_fragments");

const std::string WIND_MIN_VELOCITY("wind_min_velocity");

const std::string WIND_MAX_VELOCITY("wind_max_velocity");

const std::string GRAVITY("gravity");

const std::string AIR_MISSILES_SEPARATION("air_missiles_separation");

const std::string WORM_HEIGHT_TO_DAMAGE("worm_height_to_damage");

const std::string WORM_MAX_HEIGHT_DAMAGE("worm_max_height_damage");

const std::string GIRDER_ANGLE_FRICTION("max_girder_rotation_friction");

const std::string WORLD_MAX_HEIGHT("world_max_height");

//config del editor

const std::string BACKGROUND_IMAGE("background_image");

const std::string WORMS_LIFE("worms_life");

const std::string WORMS_DATA("worms");
```

```
67
68  const std::string GIRDERS_DATA("girders");
69
70  const std::string WEAPON_AMMO("weapon_ammo");
71
72  #endif
```



```
1   #ifndef __GAMEPLAYERSDEF_H__
2   #define __GAMEPLAYERSDEF_H__
3
4   #include <vector>
5   #include <string>
6
7   const size_t min_players = 2;
8   const size_t max_players = 5;
9
10  const std::vector<std::string> colors =
11              {"black", "blue", "green", "red", "orange"};
12
13  #endif
```

jun 25, 18 19:28 **ObjectSizes.h** Page 1/1

```
#ifndef __OBJECTSIZES_H__
#define __OBJECTSIZES_H__

const int SCALE_FACTOR = 60.0; //1 meter --- x pixels

const float UNIT_TO_SEND = 100.0; //1 cm

//in meters

const float worm_size = 0.5;

const float weapon_size = 0.25;

const float girder_height = 0.4;

const float scope_size = 1;

const int WORM_IMAGE_WIDTH = 30;

// map size

const int map_height = 5000;

const int map_width = 20000;

const int water_length = 180;

const int water_height = 30;

#endif
```

may 29, 18 23:38 **ObjectTypes.h** Page 1/1

```
#ifndef __OBJECTTYPES_H__
#define __OBJECTTYPES_H__

#include <string>

const std::string TYPE_WORM("Worm");

const std::string TYPE_WEAPON("Weapon");

const std::string TYPE_GIRDER("Girder");

const std::string TYPE_BORDER("Border");

#endif
```

```
#ifndef __PATH_H__
#define __PATH_H__

#include <string>

#ifndef ROOT_PATH
#define ROOT_PATH "."
#endif

const std::string YAML_EXTENSION(".yaml");


//general

const std::string RESOURCES(std::string(ROOT_PATH) + "/resources/");

const std::string IMAGES_PATH(RESOURCES + "Images/");

const std::string MENU_PATH(IMAGES_PATH + "Menu/");

const std::string SOUNDS_PATH(RESOURCES + "Sounds/");

const std::string GLADE_PATH(RESOURCES + "Glade/");

const std::string BACKGROUND_PATH(RESOURCES + "Background/");

const std::string ANIMATIONS_PATH(IMAGES_PATH + "Animations/");

const std::string CONFIG_PATH(std::string(ROOT_PATH) + "/config/");

const std::string MAPS_PATH(std::string(CONFIG_PATH) + "Maps/");

const std::string CLIENT_WINDOW_NAME("Worms");

const std::string EDITOR_WINDOW_NAME("Worms − Editor");

const std::string ICON_PATH(IMAGES_PATH + "icon.png");

//client

const std::string GIRDER_PATH(IMAGES_PATH + "Girder/girder_");

const std::string BULLETS_PATH(IMAGES_PATH + "Bullets/");

const std::string WORMS_PATH(IMAGES_PATH + "Worms/");

const std::string WEAPONS_PATH(IMAGES_PATH + "Weapons_icons/");

const std::string SCOPE_IMAGE(IMAGES_PATH + "Scope/Scope.gif");

const std::string VICTORY_ANIMATION(ANIMATIONS_PATH + "Victory.gif");

const std::string EXPLOSION_ANIMATION(ANIMATIONS_PATH + "Explosion.png");

const std::string TITLE_MENU_IMAGE(MENU_PATH + "Game_title.png");

const std::string BACKGROUND_MENU_IMAGE(MENU_PATH + "Background_worm.png");

const std::string WAITING_ROOM_IMAGE(MENU_PATH + "Waiting_room.png");

const std::string BAT_HIT_ANIMATION(ANIMATIONS_PATH + "Bat_hit.png");

const std::string WINNER_IMAGE(MENU_PATH + "Winner.png");

const std::string LOSER_IMAGE(MENU_PATH + "Loser.png");

```



```
const std::string TIE_IMAGE(MENU_PATH + "Tie.png");

const std::string WARNING_IMAGE(MENU_PATH + "Warning.png");

//server

const std::string SERVER_CONFIG_FILE(CONFIG_PATH + "server_config.yaml");

#endif
```

may 31, 18 21:43 **WeaponNames.h** Page 1/1

```
#ifndef __WEAPONNAMES_H__
#define __WEAPONNAMES_H__

#include <string>

const std::string AIR_ATTACK_NAME("AirAttack");

const std::string AIR_ATTACK_MISSILE_NAME("AirAttackMissile");

const std::string BANANA_NAME("Banana");

const std::string BAT_NAME("Bat");

const std::string BAZOOKA_NAME("Bazooka");

const std::string DYNAMITE_NAME("Dynamite");

const std::string GREEN_GRENADE_NAME("GreenGrenade");

const std::string HOLY_GRENADE_NAME("HolyGrenade");

const std::string MORTAR_NAME("Mortar");

const std::string MORTAR_FRAGMENTS_NAME("MortarFragment");

const std::string RED_GRENADE_NAME("RedGrenade");

const std::string RED_GRENADE_FRAGMENTS_NAME("RedGrenadeFragment");

const std::string TELEPORT_NAME("Teleportation");

const std::string FRAGMENT("Fragment");


const std::string DEFAULT_WEAPON(BAZOOKA_NAME);

const int MAX_WEAPON_ANGLE = 90;

const int MIN_WEAPON_ANGLE = -90;

const int DEFAULT_ANGLE = 48;

#endif
```

jun 26, 18 14:00 **Table of Content** Page 1/1

Table of Contents