```cpp
#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[] 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;
}

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;
}
```

```cpp
#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);

        /* 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
```

```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;
}

Buffer Protocol::sendFile(File& file){
    size_t file_size = file.size();
    Buffer buffer(file_size + 1);
    file.read_buffer(buffer.getPointer(), file_size);
    buffer.incrementOffset(file_size);
    buffer.setNext('\0');
    return buffer;
}
```

```
#ifndef __PROTOCOL_H__
#define __PROTOCOL_H__

#include <string>
#include "Buffer.h"
#include "Socket.h"
#include "File.h"

#define MAX_STRING_SIZE 200

#define CREATE_GAME_ACTION 0
#define JOIN_GAME_ACTION 1
#define START_GAME_ACTION 2

#define MOVING_OBJECT 0
#define DEAD_OBJECT 1
#define ACTION 2


#define START_TURN 3
#define END_TURN 4
#define MOVE_ACTION 5
#define CHANGE_WEAPON_ACTION 6
#define SHOOT_WEAPON_ACTION 7
#define SHOOT_WEAPON 8
#define SHOOT_SELF_DIRECTED 9
#define MOVE_SCOPE 10

#define END_GAME 11

#define MOVE_RIGHT 1
#define MOVE_LEFT -1
#define JUMP 2
#define ROLLBACK 3

#define WORM_TYPE 0
#define WEAPON_TYPE 1
#define GIRDER_TYPE 2

/* Clase que se encarga de enviar y recibir mensajes por socket
 * utilizando un formato determinado */
class Protocol {
    private:
        Socket socket;

    public:
        /* Constructor */
        explicit Protocol(Socket&& socket);

        /* Constructor por copia */
        Protocol(Protocol&& other);

        /* Destructor */
        ~Protocol();

        /* Envia el contenido del buffer */
        virtual void sendBuffer(Buffer &buffer);

        /* Recibe un mensaje, lo almacena en un buffer y lo retorna */
        Buffer receiveBuffer();


        /* Agrega el valor al buffer cumpliendo las caracteristicas del protocol
o */
        static void sendIntBuffer(Buffer &buffer, int32_t value);

```

```
        /* Retorna el valor del entero recibido almacenado en el buffer */
        static int receiveIntBuffer(Buffer &buffer);


        /* Almacena el string en el buffer  */
        static void sendStringBuffer(Buffer &buffer, const std::string &string);

        /* Retorna el string recibido que se encuentra almacenado en el buffer *
/
        static std::string receiveStringBuffer(Buffer &buffer);


        /* Envia la longitud */
        void sendLength(uint32_t length);

        /* Envia la longitud */
        static Buffer sendLengthBuffer(uint32_t length);

        /* Recibe la longitud y la retorna */
        size_t receiveLength();

        /* Cierra la comunicacion */
        void stop();

        /* Envia un char */
        void sendChar(unsigned char c);

        /* Recibe un char */
        unsigned char receiveChar();

        /* Envia un string */
        void sendString(const std::string &string);

        /* Recibe un string */
        std::string receiveString();

        /* Envia el archivo,
        Post: el archivo no se modifica */
        static Buffer sendFile(File& file);
};

#endif
```

```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;

```

```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
            }
        }
        res = res->ai_next;
    }

    throw SocketException("El socket no pudo conectarse");
}

Socket::Addrinfo::~Addrinfo(){
    freeaddrinfo(this->addrinfo);
}

Socket::Server::Server(const char* service, int max_client_wait):
    service(service), max_client_wait(max_client_wait){}

Socket::Server::~Server(){}

const char* Socket::Server::getService() const{
    return this->service;
}

int Socket::Server::getMaxClientWait() const{
    return this->max_client_wait;
}

Socket::Client::Client(const char* host, const char* service):
    host(host), service(service){}

Socket::Client::~Client(){}

const char* Socket::Client::getHost() const{
    return this->host;
}

const char* Socket::Client::getService() const{
    return this->service;
}
```

```cpp
#include "SocketException.h"
#include <string>

SocketException::SocketException(std::string msg): msg(msg){
    this->msg.insert(0, "Error en el socket: ");
}

SocketException::~SocketException(){}

const char* SocketException::what() const noexcept{
    return this->msg.c_str();
}
```

```
#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
```

```
#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;
};
```

```

class Socket::Addrinfo{
    private:
        struct addrinfo* addrinfo;

    public:
        /* Constructor */
        Addrinfo(const char* host, const char* service, int flag);

        /* Crea el socket y lo conecta o bindea segun correponda
         * Devuelve el socket creado
         * y lanza una excepcion si no pudo crearse */
        Socket connectOrBind(int action) const;

        /* Destructor */
        ~Addrinfo();
};

class Socket::Server{
    private:
        const char* service;
        int max_client_wait;

    public:
        /* Constructor */
        Server(const char* service, int max_client_wait);

        /* Destructor */
        ~Server();

        /* Devuelve el servicio */
        const char* getService() const;

        /* Devuelve la cantidad maxima de clientes en espera */
        int getMaxClientWait() const;
};

class Socket::Client{
    private:
        const char* host;
        const char* service;

    public:
        /* Constructor */
        Client(const char* host, const char* service);

        /* Destructor */
        ~Client();

        /* Devuelve el Host */
        const char* getHost() const;

        /* Devuelve el servicio */
        const char* getService() const;
};

#endif
```

```
#include "Thread.h"

Thread::Thread(): running(false){}

Thread::~Thread(){}

void Thread::start(){
    this->running = true;
    this->thread = std::thread(&Thread::run, this);
}

void Thread::join(){
    if(this->thread.joinable()){
        this->thread.join();
    }
}

bool Thread::isRunning() const{
    return this->running;
}

void Thread::stop(){
    this->running = false;
}
```

```
#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
```

```
#include <string>
#include "File.h"

const std::string FILE_ERROR("Error al abrir el archivo: ");

File::File(){}

File::File(std::string name, file_mode mode):
    file(name, mode){
        if (!this->file.is_open()){
            throw std::runtime_error(FILE_ERROR + name);
        }
    }

File::~File(){}

void File::write_buffer(const char* buffer, size_t size){
    this->file.write(buffer, size);
}

void File::read_buffer(char* buffer, size_t size){
    this->file.read(buffer, size);
}

std::string File::get_line(){
    std::string line;
    std::getline(this->file, line);
    return std::move(line);
}

size_t File::size(){
    size_t offset = this->file.tellg(); //Guardo el offset actual
    this->file.seekg(0, this->file.end);
    size_t file_length = this->file.tellg();
    this->file.seekg(offset, this->file.beg); //Vuelvo al offset original
    return file_length;
}

size_t File::gcount() const{
    return this->file.gcount();
}

bool File::eof() const{
    return this->file.eof();
}

File::File(File&& other):file(std::move(other.file)){}

File& File::operator=(File&& other){
    if (this != &other){
        this->file = std::move(other.file);
    }
    return *this;
}
```

```cpp
#ifndef __FILE_H__
#define __FILE_H__

#include <fstream>
#include <string>
#include <ios>

#define FILE_READ_MODE std::fstream::in
#define FILE_WRITE_MODE std::fstream::out | std::fstream::trunc

typedef std::ios_base::openmode file_mode;


class File{
    private:
        std::fstream file;

    public:
        //Crea un archivo vacio
        File();

        //Crea el archivo, sino se puede lanza excepcion
        File(std::string name, file_mode mode);

        //Destruye el archivo
        ~File();

        //Escribe el buffer en el archivo
        void write_buffer(const char* buffer, size_t size);

        //Lee parte del archivo y lo guarda en el buffer
        void read_buffer(char* buffer, size_t size);

        //Devuelve la siguiente linea del archivo
        std::string get_line();

        //Devuelve el tamanio del archivo
        //Post: no se modifica nada del archivo
        size_t size();

        //Devuelve la cantidad de caracteres leidos en la ultima operacion
        size_t gcount() const;

        //Devuelve true si se alcanzo EOF, false si no
        bool eof() const;

        //Constructor por movimiento
        File(File&& other);

        File& operator=(File&& other);

        //Pasaje por copia
        File(const File& other) = delete;
        File& operator=(const File& other) = delete;

        //Escribe en el archivo el objeto
        template <class T>
        File& operator<<(const T& object){
            this->file << object;
            return *this;
        }
};

#endif
```

```cpp
#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));
}
```

```
#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
```

```
#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");

const std::string GIRDERS_DATA("girders");

const std::string WEAPON_AMMO("weapon_ammo");

#endif
```

```
#ifndef __GAMEPLAYERSDEF_H__
#define __GAMEPLAYERSDEF_H__

#include <vector>
#include <string>

const size_t min_players = 2;
const size_t max_players = 5;

const std::vector<std::string> colors =
            {"black", "blue", "green", "red", "orange"};

#endif
```

```
#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 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
```

```
#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 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(IMAGES_PATH + "Game_title.png");

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

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

//server

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

#endif
```

```
#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
```

Table of Contents