

jun 09, 18 19:06

## Buffer.cpp

Page 1/1

```

1  #include "Buffer.h"
2
3
4  Buffer::Buffer(size_t max_size): buffer(new char[max_size]),
5      offset(0), max_size(max_size){}
6
7  Buffer::~Buffer(){
8      if (this->buffer){
9          delete[] buffer;
10     }
11 }
12
13 Buffer::Buffer(const Buffer& other): buffer(new char[other.max_size]){
14     for (size_t i = 0; i < other.max_size; i++){
15         this->buffer[i] = other.buffer[i];
16     }
17     this->offset = other.offset;
18     this->max_size = other.max_size;
19 }
20
21 Buffer::Buffer(Buffer&& other): buffer(other.buffer),
22     offset(other.offset), max_size(other.max_size){
23     other.buffer = NULL;
24 }
25
26 void Buffer::setNext(char value){
27     this->buffer[this->offset++] = value;
28 }
29
30 char Buffer::getNext(){
31     return this->buffer[this->offset++];
32 }
33
34 char* Buffer::getPointer(){
35     return this->buffer;
36 }
37
38 void Buffer::incrementOffset(size_t value){
39     this->offset += value;
40 }
41
42 size_t Buffer::getSize() const{
43     return this->offset;
44 }
45
46 size_t Buffer::getMaxSize() const{
47     return this->max_size;
48 }

```

jun 10, 18 19:29

## Buffer.h

Page 1/1

```

1  #ifndef __BUFFER_H__
2  #define __BUFFER_H__
3
4  #include <cstdlib>
5
6  #define MAX_BUF_LEN 200
7
8  /* Clase que representa un buffer de almacenamiento de datos */
9  class Buffer{
10     private:
11         char* buffer;
12         size_t offset;
13         size_t max_size;
14
15     public:
16         /* Constructor */
17         explicit Buffer(size_t max_size = MAX_BUF_LEN);
18
19         /* Destructor */
20         ~Buffer();
21
22         /* Constructor por copia */
23         Buffer(const Buffer& other);
24
25         /* Operador = por copia */
26         Buffer& operator=(const Buffer& other) = delete;
27
28         /* Constructor por movimiento */
29         Buffer(Buffer&& other);
30
31         /* Agrega el valor al buffer */
32         void setNext(char value);
33
34         /* Devuelve el siguiente elemento del buffer */
35         char getNext();
36
37         /* Devuelve un puntero al buffer */
38         char* getPointer();
39
40         /* Incrementa el valor del offset */
41         void incrementOffset(size_t value);
42
43         /* Devuelve el tamaño del buffer */
44         size_t getSize() const;
45
46         /* Devuelve el maximo tamaño del bufffer */
47         size_t getMaxSize() const;
48     };
49
50 #endif

```

jun 10, 18 19:29

## Protocol.cpp

Page 1/2

```

1  #include "Protocol.h"
2  #include <cstring>
3  #include <string>
4
5  Protocol::Protocol(Socket&& socket) : socket(std::move(socket)) {}
6
7  Protocol::Protocol(Protocol&& other) : socket(std::move(other.socket)) {}
8
9  Protocol::~~Protocol() {}
10
11 void Protocol::sendBuffer(Buffer &buffer) {
12     uint32_t len_converted = htonl(buffer.getSize());
13     this->socket.sendData(&len_converted, sizeof len_converted);
14     this->socket.sendData(buffer.getPointer(), buffer.getSize());
15 }
16
17 Buffer Protocol::receiveBuffer() {
18     uint32_t len;
19     this->socket.receive(&len, sizeof(uint32_t));
20     len = ntohl(len);
21
22     Buffer buffer(len);
23     this->socket.receive(buffer.getPointer(), len);
24     return std::move(buffer);
25 }
26
27 void Protocol::sendIntBuffer(Buffer &buffer, int32_t value) {
28     value = htonl(value);
29     std::memcpy(buffer.getPointer() + buffer.getSize(), &value, sizeof(value));
30     buffer.incrementOffset(sizeof(value));
31 }
32
33 int Protocol::receiveIntBuffer(Buffer &buffer) {
34     int32_t value;
35     std::memcpy(&value, buffer.getPointer() + buffer.getSize(), sizeof(value));
36     buffer.incrementOffset(sizeof(value));
37     return ntohl(value);
38 }
39
40 void Protocol::sendStringBuffer(Buffer &buffer, const std::string &string) {
41     for (size_t j = 0; j < string.size(); j++){
42         buffer.setNext(string[j]);
43     }
44     buffer.setNext('\0');
45 }
46
47 std::string Protocol::receiveStringBuffer(Buffer &buffer) {
48     std::string string;
49     char c;
50     while ((c = buffer.getNext()) != '\0'){
51         string += c;
52     }
53     return string;
54 }
55
56 void Protocol::sendLength(uint32_t length){
57     uint32_t converted = htonl(length);
58     this->socket.sendData(&converted, sizeof(uint32_t));
59 }
60
61 Buffer Protocol::sendLengthBuffer(uint32_t length){
62     Buffer buffer;
63     Protocol::sendIntBuffer(buffer, length);
64     return buffer;
65 }
66

```

jun 10, 18 19:29

## Protocol.cpp

Page 2/2

```

67 size_t Protocol::receiveLength(){
68     int32_t length;
69     this->socket.receive(&length, sizeof(int32_t));
70     return ntohl(length);
71 }
72
73 void Protocol::stop(){
74     this->socket.stop();
75 }
76
77 void Protocol::sendString(const std::string &string){
78     size_t string_length = string.size();
79     this->sendLength(string_length);
80     this->socket.sendData(string.c_str(), string_length);
81 }
82
83 std::string Protocol::receiveString(){
84     uint32_t length = this->receiveLength();
85     char buffer [MAX_STRING_SIZE + 1];
86     this->socket.receive(buffer, length);
87     buffer[length] = '\0';
88     return std::move(std::string(buffer));
89 }
90
91 void Protocol::sendChar(unsigned char c){
92     this->socket.sendData(&c, sizeof(unsigned char));
93 }
94
95 unsigned char Protocol::receiveChar(){
96     unsigned char c;
97     this->socket.receive(&c, sizeof(unsigned char));
98     return c;
99 }
100
101 Buffer Protocol::sendFile(File& file){
102     size_t file_size = file.size();
103     Buffer buffer(file_size + 1);
104     file.read_buffer(buffer.getPointer(), file_size);
105     buffer.incrementOffset(file_size);
106     buffer.setNext('\0');
107     return buffer;
108 }

```

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

jun 10, 18 19:29	Protocol.h	Page 2/2
66	<i>/* Retorna el valor del entero recibido almacenado en el buffer */</i>	
67	<b>static</b> int receiveIntBuffer(Buffer &buffer);	
68		
69		
70	<i>/* Almacena el string en el buffer */</i>	
71	<b>static</b> void sendStringBuffer(Buffer &buffer, <b>const</b> std::string &string);	
72		
73	<i>/* Retorna el string recibido que se encuentra almacenado en el buffer */</i>	
74	<i>/</i>	
75	<b>static</b> std::string receiveStringBuffer(Buffer &buffer);	
76		
77	<i>/* Envia la longitud */</i>	
78	void sendLength(uint32_t length);	
79		
80	<i>/* Envia la longitud */</i>	
81	<b>static</b> Buffer sendLengthBuffer(uint32_t length);	
82		
83	<i>/* Recibe la longitud y la retorna */</i>	
84	size_t receiveLength();	
85		
86	<i>/* Cierra la comunicacion */</i>	
87	void stop();	
88		
89	<i>/* Envia un char */</i>	
90	void sendChar(unsigned char c);	
91		
92	<i>/* Recibe un char */</i>	
93	unsigned char receiveChar();	
94		
95	<i>/* Envia un string */</i>	
96	void sendString( <b>const</b> std::string &string);	
97		
98	<i>/* Recibe un string */</i>	
99	std::string receiveString();	
100		
101	<i>/* Envia el archivo,</i>	
102	<i>Post: el archivo no se modifica */</i>	
103	<b>static</b> Buffer sendFile(File& file);	
104	};	
105		
106	<b>#endif</b>	

jun 09, 18 19:06

## Socket.cpp

Page 1/3

```

1  #include <stdexcept>
2  #include <cstring>
3  #include "Socket.h"
4
5  #define SERVER 0
6  #define CLIENT 1
7
8  Socket::Socket(Socket::Client client) : fd(-1) {
9      Addrinfo addrinfo(client.getHost(), client.getService(), 0);
10     *this = addrinfo.connectOrBind(CLIENT);
11 }
12
13 Socket::Socket(Socket::Server server) : fd(-1) {
14     char* host = NULL; //ANY
15     int flag = AI_PASSIVE;
16     Addrinfo addrinfo(host, server.getService(), flag);
17     Socket sckt = addrinfo.connectOrBind(SERVER);
18
19     if ((listen(sckt.fd, server.getMaxClientWait())) == -1){
20         throw SocketException("Error en listen");
21     }
22
23     *this = std::move(sckt);
24 }
25
26 Socket::Socket(int fd) : fd(fd){}
27
28 Socket::~Socket() {
29     this->stop();
30 }
31
32 Socket::Socket(Socket&& other) : fd(other.fd){
33     other.fd = -1;
34 }
35
36 Socket& Socket::operator=(Socket&& other){
37     if (this != &other) {
38         this->fd = other.fd;
39         other.fd = -1;
40     }
41     return *this;
42 }
43
44 int Socket::sendData(const void *data, size_t size){
45     size_t total_send = 0;
46     int actual_send;
47
48     while (total_send < size){
49         int len = size - total_send;
50         actual_send = send(this->fd, (char*)data + total_send, len, MSG_NOSIGNAL
51 );
52         if (actual_send < 0){
53             throw SocketException("Error en send");
54         }
55         if (actual_send == 0){
56             throw SocketException("Error en send: socket cerrado");
57         }
58         total_send += actual_send;
59     }
60     return total_send;
61 }
62
63 int Socket::receive(void* buffer, size_t size){
64     size_t total_recv = 0;
65     int actual_recv;

```

jun 09, 18 19:06

## Socket.cpp

Page 2/3

```

66     while (total_recv < size){
67         int len = size - total_recv;
68         actual_recv = recv(this->fd, (char*)buffer + total_recv, len, MSG_NOSIGN
69 AL);
70         if (actual_recv < 0){
71             throw SocketException("Error en receive");
72         }
73         if (actual_recv == 0){
74             throw SocketException("Error en recv: socket cerrado");
75         }
76         total_recv += actual_recv;
77     }
78     return total_recv;
79 }
80
81 Socket Socket::acceptClient(){
82     int client = accept(this->fd, NULL, NULL);
83     if (client == -1){
84         throw SocketException("Error en accept");
85     }
86     return std::move(Socket(client));
87 }
88
89 void Socket::stop(){
90     if (this->fd != -1){
91         shutdown(this->fd, SHUT_RDWR);
92         close(this->fd);
93         this->fd = -1;
94     }
95 }
96
97 Socket::Addrinfo::Addrinfo(const char* host, const char* service, int flag){
98     //Configuracion para getaddrinfo
99     struct addrinfo hints;
100     std::memset(&hints, 0, sizeof(struct addrinfo));
101     hints.ai_family = AF_INET; // IPv4
102     hints.ai_socktype = SOCK_STREAM; // TCP
103     hints.ai_flags = flag;
104
105     if ((getaddrinfo(host, service, &hints, &this->addrinfo)) != 0){
106         throw SocketException("Error en getaddrinfo");
107     }
108 }
109
110 Socket Socket::Addrinfo::connectOrBind(int action) const{
111     int conection = -1;
112     struct addrinfo* res = this->addrinfo;
113
114     while (res != NULL){
115         //Recorro todas las direcciones posibles hasta que se conecte a una
116         Socket sckt(socket(res->ai_family, res->ai_socktype, res->ai_protocol));
117         if (sckt.fd != -1){
118             if (action == SERVER){
119                 // Activo la opcion de Reusar la Direccion en caso de que esta
120                 // no este disponible por un TIME_WAIT
121                 int val = 1;
122                 int opt = setsockopt(sckt.fd, SOL_SOCKET, SO_REUSEADDR, &val, si
123 zeof(val));
124                 if (opt != -1){
125                     conection = bind(sckt.fd, res->ai_addr, res->ai_addrlen);
126                 }
127             } else if (action == CLIENT){
128                 conection = connect(sckt.fd, res->ai_addr, res->ai_addrlen);
129             }
130             if (conection != -1){
131                 return std::move(sckt); //Conexion correcta

```

jun 09, 18 19:06

## 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 19:06

## 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 19:06

## SocketException.h

Page 1/1

```

1  #ifndef __SOCKETEXCEPTION_H__
2  #define __SOCKETEXCEPTION_H__
3
4  #include <exception>
5  #include <string>
6
7  class SocketException: public std::exception{
8      private:
9          std::string msg;
10
11      public:
12          //Crea la excepcion
13          explicit SocketException(std::string msg);
14
15          //Destruye la excepcion
16          virtual ~SocketException();
17
18          //Devuelve el mensaje de error
19          virtual const char* what() const noexcept;
20 };
21
22 #endif

```

jun 09, 18 19:06

## Socket.h

Page 1/2

```

1  #ifndef __SOCKET_H__
2  #define __SOCKET_H__
3
4  #include <sys/types.h>
5  #include <sys/socket.h>
6  #include <netdb.h>
7  #include <unistd.h>
8  #include "SocketException.h"
9
10 class Socket{
11     private:
12         int fd;
13
14         /* Constructor privado del socket */
15         explicit Socket(int fd);
16
17         /* Clase privada para encapsular getaddrinfo y hacerlo RAII */
18         class Addrinfo;
19
20     public:
21         /* Clase que encapsula los parametros que se deben recibir del servidor */
22         class Server;
23
24         /* Clase que encapsula los parametros que se deben recibir del cliente */
25         class Client;
26
27         /* Constructor del socket para el cliente
28          * Crea el socket y lo conecta
29          * Lanza SocketException si ocurre un error */
30         explicit Socket(Socket::Client client);
31
32         /* Constructor del socket para el servidor
33          * Crea el socket y lo asocia al puerto indicado
34          * Lanza SocketException si ocurre un error */
35         explicit Socket(Socket::Server server);
36
37         /* Envia todos los datos del buffer data a traves del socket
38          * Devuelve la cantidad de datos que se pudieron enviar
39          * Lanza SocketException en caso de error */
40         int sendData(const void *data, size_t size);
41
42         /* Recibe datos del socket hasta alcanzar el max
43          * o hasta que se cierra la conexion
44          * Devuelve la cantidad de datos que se pudieron recibir
45          * Lanza SocketException en caso de error */
46         int receive(void* buffer, size_t size);
47
48         /* Establece una conexion con un cliente */
49         Socket acceptClient();
50
51         /* Interrumpe las conexiones del socket */
52         void stop();
53
54         /* Cierra y destruye el socket */
55         ~Socket();
56
57         /* Constructor y pasaje por movimiento */
58         Socket(Socket&& other);
59         Socket& operator=(Socket&& other);
60
61         /* Bloqueo la copia */
62         Socket(Socket& other) = delete;
63         bool operator=(Socket& other) = delete;
64 };

```

jun 09, 18 19:06

## 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 corresponda
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 26, 18 20:47

## 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 28, 18 1:22

## Thread.h

Page 1/1

```

1  #ifndef __THREAD_H__
2  #define __THREAD_H__
3
4  #include <thread>
5
6  class Thread{
7  private:
8      std::thread thread;
9
10     protected:
11         bool running;
12
13     public:
14         /* Constructor */
15         Thread();
16
17         /* Destructor */
18         virtual ~Thread();
19
20         /* Constructor por copia */
21         Thread(const Thread&) = delete;
22
23         /* Operador = por copia */
24         Thread& operator=(const Thread&) = delete;
25
26         /* Constructor por movimiento */
27         Thread(Thread&& other) = delete;
28
29         /* Operador = por movimiento */
30         Thread& operator=(Thread&& other) = delete;
31
32         /* Inicia la ejecucion del thread */
33         void start();
34
35         /* Hace join con el thread */
36         void join();
37
38         /* Devuelve true si el thread esta ejecutandose,
39          * false si ya termino*/
40         bool isRunning() const;
41
42         /* Metodo de ejecucion del thread */
43         virtual void run() = 0;
44
45         /* Termina abruptamente la ejecucion del thread */
46         virtual void stop();
47     };
48
49
50 #endif

```

jun 09, 18 19:06

## File.cpp

Page 1/1

```

1  #include <string>
2  #include "File.h"
3
4  const std::string FILE_ERROR("Error al abrir el archivo: ");
5
6  File::File() {}
7
8  File::File(std::string name, file_mode mode):
9      file(name, mode){
10     if (!this->file.is_open()){
11         throw std::runtime_error(FILE_ERROR + name);
12     }
13 }
14
15 File::~File() {}
16
17 void File::write_buffer(const char* buffer, size_t size){
18     this->file.write(buffer, size);
19 }
20
21 void File::read_buffer(char* buffer, size_t size){
22     this->file.read(buffer, size);
23 }
24
25 std::string File::get_line(){
26     std::string line;
27     std::getline(this->file, line);
28     return std::move(line);
29 }
30
31 size_t File::size(){
32     size_t offset = this->file.tellg(); //Guardo el offset actual
33     this->file.seekg(0, this->file.end);
34     size_t file_length = this->file.tellg();
35     this->file.seekg(offset, this->file.beg); //Vuelvo al offset original
36     return file_length;
37 }
38
39 size_t File::gcount() const{
40     return this->file.gcount();
41 }
42
43 bool File::eof() const{
44     return this->file.eof();
45 }
46
47 File::File(File&& other):file(std::move(other.file)){}
48
49 File& File::operator=(File&& other){
50     if (this != &other){
51         this->file = std::move(other.file);
52     }
53     return *this;
54 }

```



jun 09, 18 19:06

## File.h

Page 1/1

```

1  #ifndef __FILE_H__
2  #define __FILE_H__
3
4  #include <fstream>
5  #include <string>
6  #include <ios>
7
8  #define FILE_READ_MODE std::fstream::in
9  #define FILE_WRITE_MODE std::fstream::out | std::fstream::trunc
10
11 typedef std::ios_base::openmode file_mode;
12
13
14 class File{
15     private:
16         std::fstream file;
17
18     public:
19         //Crea un archivo vacio
20         File();
21
22         //Crea el archivo, sino se puede lanza excepcion
23         File(std::string name, file_mode mode);
24
25         //Destruye el archivo
26         ~File();
27
28         //Escribe el buffer en el archivo
29         void write_buffer(const char* buffer, size_t size);
30
31         //Lee parte del archivo y lo guarda en el buffer
32         void read_buffer(char* buffer, size_t size);
33
34         //Devuelve la siguiente linea del archivo
35         std::string get_line();
36
37         //Devuelve el tamaño del archivo
38         //Post: no se modifica nada del archivo
39         size_t size();
40
41         //Devuelve la cantidad de caracteres leídos en la última operación
42         size_t gcount() const;
43
44         //Devuelve true si se alcanza EOF, false si no
45         bool eof() const;
46
47         //Constructor por movimiento
48         File(File&& other);
49
50         File& operator=(File&& other);
51
52         //Pasaje por copia
53         File(const File& other) = delete;
54         File& operator=(const File& other) = delete;
55
56         //Escribe en el archivo el objeto
57         template <class T>
58         File& operator<<(const T& object){
59             this->file << object;
60             return *this;
61         }
62 };
63
64 #endif

```

jun 10, 18 19:29

## Math.cpp

Page 1/1

```

1  #include "Math.h"
2  #include <cmath>
3
4  const float PI = 3.14159265;
5
6  float Math::degreesToRadians(int angle){
7      return angle * PI / 180;
8  }
9
10 int Math::radiansToDegrees(float angle){
11     return angle * 180 / PI;
12 }
13
14 float Math::cosDegrees(int angle){
15     return cos(Math::degreesToRadians(angle));
16 }
17
18 float Math::sinDegrees(int angle){
19     return sin(Math::degreesToRadians(angle));
20 }

```

may 28, 18 1:22

**Math.h**

Page 1/1

```

1  #ifndef __MATHUTILS_H__
2  #define __MATHUTILS_H__
3
4  class Math{
5      public:
6          /* Transforma grados geometricos a radianes */
7          static float degreesToRadians(int angle);
8
9          /* Transforma radianes a grados geometricos */
10         static int radiansToDegrees(float angle);
11
12         /* Devuelve el resultado del coseno para el angulo
13          * en grados geometricos */
14         static float cosDegrees(int angle);
15
16         /* Devuelve el resultado del seno para el angulo
17          * en grados geometricos */
18         static float sinDegrees(int angle);
19     };
20
21 #endif

```

jun 10, 18 19:29

**ConfigFields.h**

Page 1/2

```

1  #ifndef __CONFIGFIELDS_H__
2  #define __CONFIGFIELDS_H__
3
4  #include <string>
5
6  //config del server
7
8  const std::string SERVER_PORT ("port");
9
10 const std::string DATA_SENDER_SLEEP ("data_sender_sleep");
11
12 const std::string GAME_WAIT_WORLD_SLEEP ("game_waiting_world_sleep");
13
14 const std::string WORLD_SLEEP_AFTER_STEP ("world_sleep_after_step");
15
16 const std::string WORLD_TIME_STEP ("world_time_step");
17
18 const std::string TURN_TIME ("turn_time");
19
20 const std::string TIME_AFTER_SHOOT ("time_after_shoot");
21
22 const std::string WORMS_LIFE_TO_ADD ("worms_life_to_add");
23
24 const std::string WORM_VELOCITY ("worm_velocity");
25
26 const std::string WORM_EXPLOSION_VELOCITY ("worm_explosion_velocity");
27
28 const std::string WORM_JUMP_VELOCITY ("worm_jump_velocity");
29
30 const std::string WORM_ROLLBACK_VELOCITY ("worm_rollback_velocity");
31
32 const std::string WORM_JUMP_HEIGHT ("worm_jump_height");
33
34 const std::string WORM_ROLLBACK_HEIGHT ("worm_rollback_height");
35
36 const std::string WEAPONS_VELOCITY ("weapons_velocity");
37
38 const std::string WEAPON_DAMAGE ("weapon_damage");
39
40 const std::string WEAPON_RADIUS ("weapon_radius");
41
42 const std::string WEAPON_FRAGMENTS ("weapon_fragments");
43
44 const std::string WIND_MIN_VELOCITY ("wind_min_velocity");
45
46 const std::string WIND_MAX_VELOCITY ("wind_max_velocity");
47
48 const std::string GRAVITY ("gravity");
49
50 const std::string AIR_MISSILES_SEPARATION ("air_missiles_separation");
51
52 const std::string WORM_HEIGHT_TO_DAMAGE ("worm_height_to_damage");
53
54 const std::string WORM_MAX_HEIGHT_DAMAGE ("worm_max_height_damage");
55
56 const std::string GIRDER_ANGLE_FRICTION ("max_girder_rotation_friction");
57
58 const std::string WORLD_MAX_HEIGHT ("world_max_height");
59
60 //config del editor
61
62 const std::string BACKGROUND_IMAGE ("background_image");
63
64 const std::string WORMS_LIFE ("worms_life");
65
66 const std::string WORMS_DATA ("worms");

```

jun 10, 18 19:29

**ConfigFields.h**

Page 2/2

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

jun 10, 18 19:29

**GamePlayers.h**

Page 1/1

```
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 03, 18 21:36

## ObjectSizes.h

Page 1/1

```
1  #ifndef __OBJECTSIZES_H__
2  #define __OBJECTSIZES_H__
3
4  const int SCALE_FACTOR = 60.0; //1 meter --- x pixels
5
6  const float UNIT_TO_SEND = 100.0; //1 cm
7
8  //in meters
9
10 const float worm_size = 0.5;
11
12 const float weapon_size = 0.25;
13
14 const float girder_height = 0.4;
15
16 const int WORM_IMAGE_WIDTH = 30;
17
18 // map size
19
20 const int map_height = 5000;
21
22 const int map_width = 20000;
23
24 const int water_length = 180;
25
26 const int water_height = 30;
27
28 #endif
```

may 22, 18 15:36

## ObjectTypes.h

Page 1/1

```
1  #ifndef __OBJECTTYPES_H__
2  #define __OBJECTTYPES_H__
3
4  #include <string>
5
6  const std::string TYPE_WORM("Worm");
7
8  const std::string TYPE_WEAPON("Weapon");
9
10 const std::string TYPE_GIRDER("Girder");
11
12 const std::string TYPE_BORDER("Border");
13
14 #endif
```

jun 09, 18 14:41

## Path.h

Page 1/1

```

1  #ifndef __PATH_H__
2  #define __PATH_H__
3
4  #include <string>
5
6  #ifndef ROOT_PATH
7  #define ROOT_PATH "."
8  #endif
9
10 const std::string YAML_EXTENSION("yaml");
11
12 //general
13
14 const std::string RESOURCES(std::string(ROOT_PATH) + "/resources/");
15
16 const std::string IMAGES_PATH(RESOURCES + "Images/");
17
18 const std::string SOUNDS_PATH(RESOURCES + "Sounds/");
19
20 const std::string GLADE_PATH(RESOURCES + "Glade/");
21
22 const std::string BACKGROUND_PATH(RESOURCES + "Background/");
23
24 const std::string ANIMATIONS_PATH(IMAGES_PATH + "Animations/");
25
26 const std::string CONFIG_PATH(std::string(ROOT_PATH) + "/config/");
27
28 const std::string MAPS_PATH(std::string(CONFIG_PATH) + "Maps/");
29
30 const std::string CLIENT_WINDOW_NAME("Worms");
31
32 const std::string EDITOR_WINDOW_NAME("Worms - Editor");
33
34 const std::string ICON_PATH(IMAGES_PATH + "icon.png");
35
36 //client
37
38 const std::string GIRDER_PATH(IMAGES_PATH + "Girder/girder_");
39
40 const std::string BULLETS_PATH(IMAGES_PATH + "Bullets/");
41
42 const std::string WORMS_PATH(IMAGES_PATH + "Worms/");
43
44 const std::string WEAPONS_PATH(IMAGES_PATH + "Weapons_icons/");
45
46 const std::string SCOPE_IMAGE(IMAGES_PATH + "Scope/Scope.gif");
47
48 const std::string VICTORY_ANIMATION(ANIMATIONS_PATH + "Victory.gif");
49
50 const std::string EXPLOSION_ANIMATION(ANIMATIONS_PATH + "Explosion.png");
51
52 const std::string TITLE_MENU_IMAGE(IMAGES_PATH + "Game_title.png");
53
54 const std::string BACKGROUND_MENU_IMAGE(IMAGES_PATH + "Background_worm.png");
55
56 const std::string BAT_HIT_ANIMATION(ANIMATIONS_PATH + "Bat_hit.png");
57
58 //server
59
60 const std::string SERVER_CONFIG_FILE(CONFIG_PATH + "server_config.yaml");
61
62
63 #endif

```

may 31, 18 18:24

## WeaponNames.h

Page 1/1

```

1  #ifndef __WEAPONNAMES_H__
2  #define __WEAPONNAMES_H__
3
4  #include <string>
5
6  const std::string AIR_ATTACK_NAME("AirAttack");
7
8  const std::string AIR_ATTACK_MISSILE_NAME("AirAttackMissile");
9
10 const std::string BANANA_NAME("Banana");
11
12 const std::string BAT_NAME("Bat");
13
14 const std::string BAZOOKA_NAME("Bazooka");
15
16 const std::string DYNAMITE_NAME("Dynamite");
17
18 const std::string GREEN_GRENADE_NAME("GreenGrenade");
19
20 const std::string HOLY_GRENADE_NAME("HolyGrenade");
21
22 const std::string MORTAR_NAME("Mortar");
23
24 const std::string MORTAR_FRAGMENTS_NAME("MortarFragment");
25
26 const std::string RED_GRENADE_NAME("RedGrenade");
27
28 const std::string RED_GRENADE_FRAGMENTS_NAME("RedGrenadeFragment");
29
30 const std::string TELEPORT_NAME("Teleportation");
31
32 const std::string FRAGMENT("Fragment");
33
34
35 const std::string DEFAULT_WEAPON(BAZOOKA_NAME);
36
37 const int MAX_WEAPON_ANGLE = 90;
38
39 const int MIN_WEAPON_ANGLE = -90;
40
41 const int DEFAULT_ANGLE = 48;
42
43 #endif

```

jun 10, 18 20:00

**Table of Content**

Page 1/1

1	<b>Table of Contents</b>					
2	1 <i>Buffer.cpp</i> .....	sheets	1 to	1 ( 1)	pages	1- 1 49 lines
3	2 <i>Buffer.h</i> .....	sheets	1 to	1 ( 1)	pages	2- 2 51 lines
4	3 <i>Protocol.cpp</i> .....	sheets	2 to	2 ( 1)	pages	3- 4 109 lines
5	4 <i>Protocol.h</i> .....	sheets	3 to	3 ( 1)	pages	5- 6 107 lines
6	5 <i>Socket.cpp</i> .....	sheets	4 to	5 ( 2)	pages	7- 9 167 lines
7	6 <i>SocketException.cpp</i> .	sheets	5 to	5 ( 1)	pages	10- 10 13 lines
8	7 <i>SocketException.h</i> ...	sheets	6 to	6 ( 1)	pages	11- 11 23 lines
9	8 <i>Socket.h</i> .....	sheets	6 to	7 ( 2)	pages	12- 13 122 lines
10	9 <i>Thread.cpp</i> .....	sheets	7 to	7 ( 1)	pages	14- 14 25 lines
11	10 <i>Thread.h</i> .....	sheets	8 to	8 ( 1)	pages	15- 15 51 lines
12	11 <i>File.cpp</i> .....	sheets	8 to	8 ( 1)	pages	16- 16 55 lines
13	12 <i>File.h</i> .....	sheets	9 to	9 ( 1)	pages	17- 17 65 lines
14	13 <i>Math.cpp</i> .....	sheets	9 to	9 ( 1)	pages	18- 18 21 lines
15	14 <i>Math.h</i> .....	sheets	10 to	10 ( 1)	pages	19- 19 22 lines
16	15 <i>ConfigFields.h</i> .....	sheets	10 to	11 ( 2)	pages	20- 21 73 lines
17	16 <i>GamePlayers.h</i> .....	sheets	11 to	11 ( 1)	pages	22- 22 14 lines
18	17 <i>ObjectSizes.h</i> .....	sheets	12 to	12 ( 1)	pages	23- 23 29 lines
19	18 <i>ObjectTypes.h</i> .....	sheets	12 to	12 ( 1)	pages	24- 24 15 lines
20	19 <i>Path.h</i> .....	sheets	13 to	13 ( 1)	pages	25- 25 64 lines
21	20 <i>WeaponNames.h</i> .....	sheets	13 to	13 ( 1)	pages	26- 26 44 lines