

Jun 20, 18 14:57

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 20, 18 14:57

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

```

Jun 20, 18 14:57	Protocol.h	Page 1/2
<pre> 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 características del protocol 63 o */ 64 static void sendIntBuffer(Buffer &buffer, int32_t value); 65 /* Retorna el valor del entero recibido almacenado en el buffer */ </pre>		

Jun 20, 18 14:57	Protocol.h	Page 2/2
<pre> 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 74 static std::string receiveStringBuffer(Buffer &buffer); 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 </pre>		

Jun 07, 18 15:21

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 07, 18 15:21

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 07, 18 15:21

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 }

```

May 30, 18 20:03

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 }

```

May 26, 18 12:13

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

```

May 28, 18 18:21

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 };

```

May 28, 18 18:21

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 15:23

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 18:21

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 20, 18 14:57

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[] this->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 Buffer& Buffer::operator=(Buffer&& other){
27     if (this->buffer){
28         delete[] this->buffer;
29     }
30     this->buffer = other.buffer;
31     other.buffer = NULL;
32     this->offset = other.offset;
33     this->max_size = other.max_size;
34     return *this;
35 }
36
37 void Buffer::setNext(char value){
38     this->buffer[this->offset++] = value;
39 }
40
41 char Buffer::getNext() {
42     return this->buffer[this->offset++];
43 }
44
45 char* Buffer::getPointer(){
46     return this->buffer;
47 }
48
49 void Buffer::incrementOffset(size_t value){
50     this->offset += value;
51 }
52
53 size_t Buffer::getSize() const{
54     return this->offset;
55 }
56
57 size_t Buffer::getMaxSize() const{
58     return this->max_size;
59 }

```

Jun 20, 18 14:57

Buffer.h

Page 1/1

```

1  #ifndef __BUFFER_H__
2  #define __BUFFER_H__
3
4  #include <cstdint>
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     /* Operador = por movimiento */
32     Buffer& operator=(Buffer&& other);
33
34     /* Agrega el valor al buffer */
35     void setNext(char value);
36
37     /* Devuelve el siguiente elemento del buffer */
38     char getNext();
39
40     /* Devuelve un puntero al buffer */
41     char* getPointer();
42
43     /* Incrementa el valor del offset */
44     void incrementOffset(size_t value);
45
46     /* Devuelve el tamaño del buffer */
47     size_t getSize() const;
48
49     /* Devuelve el maximo tamaño del bufffer */
50     size_t getMaxSize() const;
51 };
52
53 #endif

```

Jun 10, 18 15:21

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 18:21

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 15:29

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 15:29

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 15:29

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 20, 18 14:57

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 float scope_size = 1;
17
18 const int WORM_IMAGE_WIDTH = 30;
19
20 // map size
21
22 const int map_height = 5000;
23
24 const int map_width = 20000;
25
26 const int water_length = 180;
27
28 const int water_height = 30;
29
30 #endif

```

May 16, 18 12:35

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 20, 18 14:57

Path.h

Page 1/2

```

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 MENU_PATH(IMAGES_PATH + "Menu/");
19
20 const std::string SOUNDS_PATH(RESOURCES + "Sounds/");
21
22 const std::string GLADE_PATH(RESOURCES + "Glade/");
23
24 const std::string BACKGROUND_PATH(RESOURCES + "Background/");
25
26 const std::string ANIMATIONS_PATH(IMAGES_PATH + "Animations/");
27
28 const std::string CONFIG_PATH(std::string(ROOT_PATH) + "/config/");
29
30 const std::string MAPS_PATH(std::string(CONFIG_PATH) + "Maps/");
31
32 const std::string CLIENT_WINDOW_NAME("Worms");
33
34 const std::string EDITOR_WINDOW_NAME("Worms - Editor");
35
36 const std::string ICON_PATH(IMAGES_PATH + "icon.png");
37
38 //client
39
40 const std::string GIRDER_PATH(IMAGES_PATH + "Girder/girder_");
41
42 const std::string BULLETS_PATH(IMAGES_PATH + "Bullets/");
43
44 const std::string WORMS_PATH(IMAGES_PATH + "Worms/");
45
46 const std::string WEAPONS_PATH(IMAGES_PATH + "Weapons_icons/");
47
48 const std::string SCOPE_IMAGE(IMAGES_PATH + "Scope/Scope.gif");
49
50 const std::string VICTORY_ANIMATION(ANIMATIONS_PATH + "Victory.gif");
51
52 const std::string EXPLOSION_ANIMATION(ANIMATIONS_PATH + "Explosion.png");
53
54 const std::string TITLE_MENU_IMAGE(MENU_PATH + "Game_title.png");
55
56 const std::string BACKGROUND_MENU_IMAGE(MENU_PATH + "Background_worm.png");
57
58 const std::string BAT_HIT_ANIMATION(ANIMATIONS_PATH + "Bat_hit.png");
59
60 const std::string WINNER_IMAGE(MENU_PATH + "Winner.png");
61
62 const std::string LOSER_IMAGE(MENU_PATH + "Loser.png");
63
64 const std::string TIE_IMAGE(MENU_PATH + "Tie.png");
65
66

```

Jun 20, 18 14:57

Path.h

Page 2/2

```

67 const std::string WARNING_IMAGE(MENU_PATH + "Warning.png");
68
69 //server
70
71 const std::string SERVER_CONFIG_FILE(CONFIG_PATH + "server_config.yaml");
72
73 #endif

```

