

jun 25, 18 19:28

ClientHandler.cpp

Page 1/2

```

1  #include "ClientHandler.h"
2  #include "MapsList.h"
3  #include <iostream>
4  #include <string>
5
6  ClientHandler::ClientHandler(ServerProtocol&& client,
7                               GamesList& games, std::mutex& mutex_cout):
8      client(std::move(client)), games(games),
9      connected(false), mutex_cout(mutex_cout){}
10
11 ClientHandler::~ClientHandler(){}
12
13 void ClientHandler::run(){
14     try{
15         while(!this->connected){
16             char action = this->client.getProtocol().receiveChar();
17             std::string player_name = this->client.getProtocol().receiveString();
18
19             this->client.setName(player_name);
20             if (action == CREATE_GAME_ACTION) {
21                 this->createGame();
22             } else if (action == JOIN_GAME_ACTION) {
23                 this->joinGame();
24             }
25         } catch(const SocketException& e){
26             std::lock_guard<std::mutex> lock(this->mutex_cout);
27             std::cout << "[INFO] Un cliente se desconecto" << std::endl;
28         } catch(const std::exception& e){
29             std::lock_guard<std::mutex> lock(this->mutex_cout);
30             std::cout << "[ERROR] Error con un cliente: " << e.what() << std::endl;
31         }
32         this->running = false;
33     }
34
35 void ClientHandler::stop(){
36     this->client.getProtocol().stop();
37 }
38
39 void ClientHandler::createGame(){
40     maps_list_t maps_list = MapsList::getAllMaps();
41
42     size_t size = maps_list.size();
43     this->client.getProtocol().sendLength(size);
44
45     for (size_t i = 0; i < size; i++){
46         this->client.getProtocol().sendString(maps_list[i]);
47     }
48
49     if (size == 0){
50         return;
51     }
52
53     std::string map = this->client.getProtocol().receiveString();
54     if (map.empty()){
55         return;
56     }
57     std::string game_name = this->client.getProtocol().receiveString();
58     int max_players = this->client.getProtocol().receiveLength();
59
60     this->games.checkGames();
61
62     bool result = this->games.addGame(game_name, map, max_players, this->client);
63
64     if (!result){
65         this->client.getProtocol().sendChar(false);

```

jun 25, 18 19:28

ClientHandler.cpp

Page 2/2

```

65     } else {
66         this->connected = true;
67     }
68 }
69
70 void ClientHandler::joinGame(){
71     games_list_t games_list = this->games.getJoinableGames(this->client.getName());
72
73     size_t size = games_list.size();
74     this->client.getProtocol().sendLength(size);
75
76     for (size_t i = 0; i < size; i++){
77         this->client.getProtocol().sendString(games_list[i]);
78     }
79
80     if (size == 0){
81         return;
82     }
83
84     std::string game_name = this->client.getProtocol().receiveString();
85     if (game_name.empty()){
86         return;
87     }
88
89     bool result = this->games.addPlayer(game_name, this->client);
90
91     if (!result){
92         this->client.getProtocol().sendChar(false);
93     } else {
94         this->connected = true;
95     }
96 }

```

jun 12, 18 14:03

ClientHandler.h

Page 1/1

```

1  #ifndef __CLIENTHANDLER_H__
2  #define __CLIENTHANDLER_H__
3
4  #include "Socket.h"
5  #include "Server.h"
6  #include "Thread.h"
7  #include "Player.h"
8  #include "GamesList.h"
9  #include <mutex>
10
11 class ClientHandler: public Thread{
12     private:
13         Player client;
14         GamesList& games;
15         bool connected;
16         std::mutex& mutex_cout;
17
18         /* Crea una partida nueva */
19         void createGame();
20
21         /* Agrega un jugador a una partida */
22         void joinGame();
23
24     public:
25         /* Constructor */
26         ClientHandler(ServerProtocol& cli, GamesList& games, std::mutex& mtx_co
ut);
27
28         /* Destructor */
29         ~ClientHandler();
30
31         /* Ejecuta el client handler */
32         void run();
33
34         /* Se desconecta abruptamente del cliente */
35         void stop();
36 };
37
38 #endif

```

jun 25, 18 19:28

GamesList.cpp

Page 1/2

```

1  #include "GamesList.h"
2  #include "Path.h"
3  #include "Server.h"
4  #include <iostream>
5  #include <string>
6
7  typedef std::unordered_map<std::string, std::unique_ptr<Game>>::iterator games_i
t;
8
9  GamesList::GamesList(Server& server, std::mutex& mutex_cout):
10     server(server), mutex_cout(mutex_cout){}
11
12  GamesList::~~GamesList(){
13     for (games_it it = this->games.begin(); it != this->games.end(); ++it){
14         it->second->join();
15         std::lock_guard<std::mutex> lock(this->mutex_cout);
16         std::cout << "[INFO] Partida terminada: " << it->first << std::endl;
17     }
18 }
19
20 bool GamesList::addGame(string& name, string& map, int max, Player& player){
21     std::lock_guard<std::mutex> lock(this->mutex);
22     auto it = this->games.find(name);
23     if (it != this->games.end()){
24         return false;
25     }
26
27     try{
28         Game* g = new Game(max, SERVER_CONFIG_FILE, MAPS_PATH + map, this->serve
r);
29         std::unique_ptr<Game> game(g);
30         this->games[name] = std::move(game);
31         std::lock_guard<std::mutex> lock(this->mutex_cout);
32         std::cout << "[INFO] Nueva partida creada: " << name << std::endl;
33     } catch(const std::exception& e){
34         std::lock_guard<std::mutex> lock(this->mutex_cout);
35         std::cout << "[ERROR] Error al crear partida: " << name;
36         std::cout << "->" << e.what() << std::endl;
37         return false;
38     }
39
40     std::string player_name = player.getName();
41     bool result = this->games[name]->addPlayer(player);
42     if (result){
43         std::lock_guard<std::mutex> lock(this->mutex_cout);
44         std::cout << "[INFO] El jugador '" << player_name;
45         std::cout << "' se unio a la partida '" << name << "'" << std::endl;
46     }
47
48     return result;
49 }
50
51 games_list_t GamesList::getJoinableGames(const std::string& player_name){
52     std::lock_guard<std::mutex> lock(this->mutex);
53     games_list_t joinables;
54
55     for (games_it it = this->games.begin(); it != this->games.end(); ++it){
56         if (it->second->playerCanJoin(player_name)){
57             joinables.push_back(it->first);
58         }
59     }
60     return std::move(joinables);
61 }
62
63 bool GamesList::addPlayer(const std::string& game_name, Player& player){
64     std::lock_guard<std::mutex> lock(this->mutex);

```

jun 25, 18 19:28

GamesList.cpp

Page 2/2

```

65     std::string player_name = player.getName();
66     bool result = this->games[game_name]->addPlayer(player);
67     if (result){
68         std::lock_guard<std::mutex> lock(this->mutex_cout);
69         std::cout << "[INFO] El jugador '" << player_name;
70         std::cout << "' se unio a la partida '" << game_name << "'" << std::endl;
71     }
72     if (this->games[game_name]->isFull()){
73         std::lock_guard<std::mutex> lock(this->mutex_cout);
74         std::cout << "[INFO] Partida iniciada: " << game_name << std::endl;
75         this->games[game_name]->start();
76     }
77     return result;
78 }
79
80 void GamesList::checkGames(){
81     std::lock_guard<std::mutex> lock(this->mutex);
82     auto it = this->games.begin();
83     while (it != this->games.end()){
84         if (! it->second->isRunning()){
85             it->second->join();
86             std::lock_guard<std::mutex> lock(this->mutex_cout);
87             std::cout << "[INFO] Partida terminada: " << it->first << std::endl;
88             it = this->games.erase(it);
89         } else {
90             ++it;
91         }
92     }
93 }

```

jun 12, 18 14:03

GamesList.h

Page 1/1

```

1  #ifndef __GAMESLIST_H__
2  #define __GAMESLIST_H__
3
4  #include <vector>
5  #include <string>
6  #include <unordered_map>
7  #include <memory>
8  #include <mutex>
9  #include "Game.h"
10
11 typedef std::vector<std::string> games_list_t;
12
13 class Server;
14
15 class GamesList{
16     private:
17         Server& server;
18         std::unordered_map<std::string, std::unique_ptr<Game>> games;
19         std::mutex mutex;
20         std::mutex& mutex_cout;
21
22         typedef const std::string string;
23
24     public:
25         /* Constructor */
26         GamesList(Server& server, std::mutex& mutex_cout);
27
28         /* Destructor */
29         ~GamesList();
30
31         /* Agrega una patida nueva a la lista */
32         bool addGame(string& game_name, string& map, int max_players, Player& pl
33         ayer);
34
35         /* Devuelve una lista con las partidas a las cuales se puede
36         * unir el jugador */
37         games_list_t getJoinableGames(const std::string& player_name);
38
39         /* Agrega un jugador a la partida */
40         bool addPlayer(const std::string& game_name, Player& player);
41
42         /* Verifica las partidas que terminaron */
43         void checkGames();
44
45     };
46
47 #endif

```

jun 12, 18 14:03

MapsList.cpp

Page 1/1

```

1  #include "MapsList.h"
2  #include "Path.h"
3  #include <string>
4
5  maps_list_t MapsList::getAllMaps(){
6      maps_list_t maps_list;
7
8      struct dirent* entry;
9      DIR* dir = opendir(MAPS_PATH.c_str());
10     if (!dir){
11         std::move(maps_list);
12     }
13
14     while((entry = readdir(dir)){ // NOLINT (La solucion que propone esta depr
ecated)
15         std::string file(entry->d_name);
16         if (file.rfind(YAML_EXTENSION) != std::string::npos){
17             maps_list.push_back(file);
18         }
19     }
20
21     closedir(dir);
22     return std::move(maps_list);
23 }

```

may 29, 18 23:38

MapsList.h

Page 1/1

```

1  #ifndef __MAPSLIST_H__
2  #define __MAPSLIST_H__
3
4  #include <dirent.h>
5  #include <vector>
6  #include <string>
7
8  typedef std::vector<std::string> maps_list_t;
9
10 class MapsList{
11     public:
12         /* Devuelve una lista con todos los mapas */
13         static maps_list_t getAllMaps();
14 };
15
16 #endif

```

jun 25, 18 19:28

Server.cpp

Page 1/2

```

1  #include <string>
2  #include <memory>
3  #include <iostream>
4  #include "Server.h"
5  #include "ClientHandler.h"
6
7  #define MAX_CLIENT_WAIT 100
8
9  Server::Server(const std::string& service, std::mutex& mutex_cout):
10     socket(Socket::Server(service.c_str(), MAX_CLIENT_WAIT)),
11     games_list(*this, mutex_cout), mutex_cout(mutex_cout){}
12
13  Server::~Server(){
14     for (std::unique_ptr<Thread>& client: this->clients){
15         client->stop();
16         client->join();
17     }
18 }
19
20 void Server::run(){
21     while (this->running){
22         try{
23             Socket client = this->socket.acceptClient();
24             {
25                 std::lock_guard<std::mutex> lock(this->mutex_cout);
26                 std::cout << "[INFO] Nuevo cliente conectado." << std::endl;
27             }
28             ServerProtocol protocol(std::move(client));
29             this->addConnectedClient(std::move(protocol));
30
31             this->check();
32         } catch(const std::exception& e){
33             if (this->running){
34                 std::lock_guard<std::mutex> lock(this->mutex_cout);
35                 std::cout << "[ERROR] " << e.what() << std::endl;
36             }
37         }
38     }
39 }
40
41 void Server::stop(){
42     this->running = false;
43     this->socket.stop();
44 }
45
46 void Server::check(){
47     //Elimino threads que ya terminaron
48     this->games_list.checkGames();
49     std::lock_guard<std::mutex> lock(this->mutex);
50     auto it = this->clients.begin();
51     while (it != this->clients.end()){
52         if (!(*it)->isRunning()){
53             (*it)->join();
54             it = this->clients.erase(it);
55         } else {
56             ++it;
57         }
58     }
59 }
60
61 void Server::addConnectedClient(ServerProtocol&& protocol){
62     std::lock_guard<std::mutex> lock(this->mutex);
63     Thread* t = new ClientHandler(std::move(protocol), games_list, mutex_cout);
64     std::unique_ptr<Thread> th(t);
65     th->start();
66     this->clients.push_back(std::move(th));

```

jun 25, 18 19:28

Server.cpp

Page 2/2

```

67 }

```

jun 07, 18 21:50

Server.h

Page 1/1

```

1  #ifndef __SERVER_H__
2  #define __SERVER_H__
3
4  #include <string>
5  #include <list>
6  #include <memory>
7  #include <mutex>
8  #include "Socket.h"
9  #include "Thread.h"
10 #include "GamesList.h"
11
12 class Server: public Thread{
13     private:
14         Socket socket;
15         std::list<std::unique_ptr<Thread>> clients;
16         GamesList games_list;
17         std::mutex& mutex_cout;
18         std::mutex mutex;
19
20         /* Elimina los clientes que terminaron su comunicacion
21          * de la lista */
22         void check();
23
24     public:
25         /* Crea el server y lo asocia al puerto indicado */
26         Server(const std::string& service, std::mutex& mutex_cout);
27
28         /* Desconecta el server */
29         ~Server();
30
31         /* Ejecuta el server */
32         void run();
33
34         /* Avisa al server que debe dejar de ejecutarse */
35         void stop();
36
37         /*Agrega un nuevo cliente ya conectado */
38         void addConnectedClient(ServerProtocol&& protocol);
39 };
40
41 #endif

```

jun 25, 18 19:28

DataSender.cpp

Page 1/3

```

1  #include "DataSender.h"
2  #include <map>
3  #include <string>
4  #include <vector>
5
6  typedef std::vector<Player> Players;
7
8  DataSender::DataSender(World& world, Players& players, GameParameters& params):
9      objects(world.getObjectsList()), girders(world.getGirdersList()),
10      players(players), mutex(world.getMutex()), active(false),
11      sleep_time(params.getDataSenderSleep()){
12      for (size_t i = 0; i < this->players.size(); i++){
13          std::unique_ptr<PlayerDataSender> s(new PlayerDataSender(this->playe
14          rs[i]));
15          this->players_data_senders.push_back(std::move(s));
16          this->players_data_senders[i]->start();
17      }
18
19  DataSender::~DataSender(){
20      for (size_t i = 0; i < this->players.size(); i++){
21          this->players_data_senders[i]->stop();
22          this->players_data_senders[i]->join();
23      }
24  }
25
26  void DataSender::run(){
27      while(this->running){
28          std::this_thread::sleep_for(std::chrono::milliseconds(this->sleep_time))
29      ;
30
31      std::lock_guard<std::mutex> lock(this->mutex);
32      this->active = false;
33      std::list<physical_object_ptr>::iterator it = this->objects.begin();
34
35      while(it != this->objects.end()){
36          if ((*it)->isDead() && !(*it)->getBody()){
37              Buffer data = ServerProtocol::sendDeadObject(*it);
38
39              this->sendBuffer(data);
40              it = this->objects.erase(it);
41              continue;
42          }
43
44          if ((*it)->isMoving()){
45              Buffer data = ServerProtocol::sendObject(*it);
46              this->sendBuffer(data);
47              this->active = true;
48          }
49          ++it;
50      }
51      this->notifyAll();
52  }
53
54  void DataSender::sendBackgroundImage(Buffer& image){
55      this->sendBuffer(image);
56      this->notifyAll();
57  }
58
59  void DataSender::sendStartGame(){
60      Buffer data = ServerProtocol::sendStartGame();
61      this->sendBuffer(data);
62      this->notifyAll();
63  }
64

```

jun 25, 18 19:28

DataSender.cpp

Page 2/3

```

65 void DataSender::sendTurnData(int turn_time, int time_after_shoot){
66     Buffer data = ServerProtocol::sendTurnData(turn_time, time_after_shoot);
67     this->sendBuffer(data);
68     this->notifyAll();
69 }
70
71 void DataSender::sendPlayersId(){
72     Buffer length = ServerProtocol::sendLengthBuffer(this->players.size());
73     this->sendBuffer(length);
74     for (Player& player: this->players){
75         Buffer data = ServerProtocol::sendPlayerId(player);
76         this->sendBuffer(data);
77     }
78     this->notifyAll();
79 }
80
81 void DataSender::sendGirders(){
82     Buffer length = ServerProtocol::sendLengthBuffer(this->girders.size());
83     this->sendBuffer(length);
84     for (physical_object_ptr& girder: this->girders){
85         Buffer data = ServerProtocol::sendGirder(girder);
86         this->sendBuffer(data);
87     }
88     this->notifyAll();
89 }
90
91 void DataSender::sendWeaponsAmmo(std::map<std::string, unsigned int>& weapons){
92     Buffer length = ServerProtocol::sendLengthBuffer(weapons.size());
93     this->sendBuffer(length);
94     std::map<std::string, unsigned int>::iterator it;
95     for (it = weapons.begin(); it != weapons.end(); ++it){
96         Buffer data = ServerProtocol::sendWeaponAmmo(it->first, it->second);
97         this->sendBuffer(data);
98     }
99     this->notifyAll();
100 }
101
102 void DataSender::sendStartTurn(int worm_id, int player_id, float wind){
103     Buffer data = ServerProtocol::sendStartTurn(worm_id, player_id, wind);
104     this->sendBuffer(data);
105     this->notifyAll();
106 }
107
108 void DataSender::sendWeaponChanged(const std::string &weapon){
109     Buffer data = ServerProtocol::sendWeaponChanged(weapon);
110     this->sendBuffer(data);
111     this->notifyAll();
112 }
113
114 void DataSender::sendWeaponShot(const std::string& weapon){
115     Buffer data = ServerProtocol::sendWeaponShot(weapon);
116     this->sendBuffer(data);
117     this->notifyAll();
118 }
119
120 void DataSender::sendMoveAction(char action){
121     if (action == MOVE_RIGHT || action == MOVE_LEFT){
122         return;
123     }
124     Buffer data = ServerProtocol::sendMoveAction(action);
125     this->sendBuffer(data);
126     this->notifyAll();
127 }
128
129 void DataSender::sendUpdateScope(int angle) {
130     Buffer data = ServerProtocol::sendUpdateScope(angle);

```

jun 25, 18 19:28

DataSender.cpp

Page 3/3

```

131     this->sendBuffer(data);
132     this->notifyAll();
133 }
134
135 void DataSender::sendEndGame(const std::string& winner){
136     Buffer data = ServerProtocol::sendEndGame(winner);
137     this->sendBuffer(data);
138     this->notifyAll();
139 }
140
141 void DataSender::sendEndTurn(){
142     Buffer data = this->players[0].getProtocol().sendEndTurn();
143     this->sendBuffer(data);
144     this->notifyAll();
145 }
146
147 bool DataSender::isActive(){
148     std::lock_guard<std::mutex> lock(this->mutex);
149     return this->active;
150 }
151
152 void DataSender::sendBuffer(const Buffer& buffer){
153     for (size_t i = 0; i < this->players.size(); i++){
154         if (this->players[i].isConnected()){
155             this->players_data_senders[i]->sendData(buffer);
156         }
157     }
158 }
159
160 void DataSender::notifyAll(){
161     for (size_t i = 0; i < this->players.size(); i++){
162         if (this->players[i].isConnected()){
163             this->players_data_senders[i]->notify();
164         }
165     }
166 }

```

jun 19, 18 13:51	DataSender.h	Page 1/2
<pre> 1 #ifndef __DATASENDER_H__ 2 #define __DATASENDER_H__ 3 4 #include "Thread.h" 5 #include "World.h" 6 #include "PhysicalObject.h" 7 #include "Player.h" 8 #include "ServerProtocol.h" 9 #include "PlayerDataSender.h" 10 #include "Buffer.h" 11 #include <list> 12 #include <memory> 13 #include <vector> 14 #include <map> 15 #include <string> 16 17 // Clase que se encarga de enviar datos a los jugadores 18 class DataSender: public Thread{ 19 private: 20 std::list<physical_object_ptr>& objects; 21 std::list<physical_object_ptr>& girders; 22 std::vector<Player>& players; 23 std::vector<std::unique_ptr<PlayerDataSender>> players_data_senders; 24 std::mutex& mutex; 25 bool active; 26 int sleep_time; 27 28 // Envia la informacion del buffer a todos los jugadores 29 void sendBuffer(const Buffer& buffer); 30 void notifyAll(); 31 32 public: 33 DataSender(World& world, std::vector<Player>& players, GameParameters& p 34 aram); 35 ~DataSender(); 36 37 //Envia constantemente los datos de los objetos 38 void run() override; 39 40 //Envia la imagen de fondo 41 void sendBackgroundImage(Buffer& image); 42 43 //Envia los datos del turno 44 void sendTurnData(int turn_time, int time_after_shoot); 45 46 //Envia los datos de los jugadores 47 void sendPlayersId(); 48 49 //Envia los datos de las vigas 50 void sendGirders(); 51 52 //Envia las municiones de las armas 53 void sendWeaponsAmmo(std::map<std::string, unsigned int>& weapons); 54 55 //Envia que el jugador cambio de arma 56 void sendWeaponChanged(const std::string &weapon); 57 58 //Envia que el gusano actual salto 59 void sendMoveAction(char action); 60 61 //Envia que el jugador cambio el angulo de la mira 62 void sendUpdateScope(int angle); 63 64 //Envia que el jugador disparo un arma 65 void sendWeaponShot(const std::string& weapon); </pre>		

jun 19, 18 13:51	DataSender.h	Page 2/2
<pre> 66 //Envia la senial de comienzo del juego 67 void sendStartGame(); 68 69 //Envia la senial de que inicia un nuevo turno 70 void sendStartTurn(int worm_id, int player_id, float wind); 71 72 //Envia la senial de terminar turno 73 void sendEndTurn(); 74 75 //Envia la senial de que el juego termino 76 void sendEndGame(const std::string& winner); 77 78 //Devuelve true si sigue enviando datos 79 bool isActive(); 80 }; 81 82 #endif 83 </pre>		

jun 12, 18 14:03

PlayerDataReceiver.cpp

Page 1/2

```

1  #include "PlayerDataReceiver.h"
2  #include <string>
3
4  PlayerDataReceiver::PlayerDataReceiver(Player& player, DataSender& sender):
5      player(player), data_sender(sender), is_my_turn(false){}
6
7  PlayerDataReceiver::~PlayerDataReceiver() {}
8
9  void PlayerDataReceiver::run() {
10     try{
11         while (this->running) {
12             Buffer data = this->player.getProtocol().receiveBuffer();
13             std::lock_guard<std::mutex> lock(this->mutex);
14             if (this->is_my_turn) {
15                 this->analyzeReceivedData(data);
16             }
17         }
18     } catch(const std::exception& e) {
19         this->player.disconnect();
20     }
21 }
22
23 void PlayerDataReceiver::beginTurn() {
24     std::lock_guard<std::mutex> lock(this->mutex);
25     this->is_my_turn = true;
26 }
27
28 void PlayerDataReceiver::endTurn() {
29     std::lock_guard<std::mutex> lock(this->mutex);
30     this->is_my_turn = false;
31 }
32
33 void PlayerDataReceiver::analyzeReceivedData(Buffer& buffer) {
34     char action = buffer.getNext();
35
36     if (action == ACTION) {
37         char worm_action = buffer.getNext();
38         if (worm_action == MOVE_ACTION) {
39             char move = buffer.getNext();
40             if (this->player.getCurrentWorm().move(move)) {
41                 this->data_sender.sendMoveAction(move);
42             }
43         } else if (worm_action == CHANGE_WEAPON_ACTION) {
44             std::string weapon(ServerProtocol::receiveStringBuffer(buffer));
45             this->data_sender.sendWeaponChanged(weapon);
46             this->player.changeWeapon(weapon);
47         } else if (worm_action == MOVE_SCOPE) {
48             int32_t angle = ServerProtocol::receiveIntBuffer(buffer);
49             this->data_sender.sendUpdateScope(angle);
50         } else if (worm_action == SHOOT_WEAPON) {
51             int angle = ServerProtocol::receiveIntBuffer(buffer);
52             int power = ServerProtocol::receiveIntBuffer(buffer);
53             int time = ServerProtocol::receiveIntBuffer(buffer);
54             const std::string& weapon = this->player.getCurrentWorm().getCurrent
Weapon();
55             this->data_sender.sendWeaponShot(weapon);
56             this->player.getCurrentWorm().shoot(angle, power, time);
57         } else if (worm_action == SHOOT_SELF_DIRECTED) {
58             int pos_x = ServerProtocol::receiveIntBuffer(buffer) / UNIT_TO_SEND;
59             int pos_y = ServerProtocol::receiveIntBuffer(buffer) / UNIT_TO_SEND;
60             const std::string& weapon = this->player.getCurrentWorm().getCurrent
Weapon();
61             this->data_sender.sendWeaponShot(weapon);
62             this->player.getCurrentWorm().shoot(b2Vec2(pos_x, pos_y));
63         }
64     }

```

jun 12, 18 14:03

PlayerDataReceiver.cpp

Page 2/2

65 }

jun 12, 18 14:03

PlayerDataReceiver.h

Page 1/1

```

1  #ifndef __PLAYERDATARECEIVER_H__
2  #define __PLAYERDATARECEIVER_H__
3
4  #include "Thread.h"
5  #include "Player.h"
6  #include "DataSender.h"
7  #include <mutex>
8
9  /* Clase que se encarga de recibir datos del jugador
10   * y de analizarlos */
11  class PlayerDataReceiver: public Thread{
12  private:
13      Player& player;
14      DataSender& data_sender;
15      bool is_my_turn;
16      std::mutex mutex;
17
18      /* Analiza los datos recibidos */
19      void analyzeReceivedData(Buffer& data);
20
21  public:
22      /* Constructor */
23      PlayerDataReceiver(Player& player, DataSender& data_sender);
24
25      /* Destructor */
26      ~PlayerDataReceiver();
27
28      /* Comienza a recibir datos */
29      void run() override;
30
31      /* Comienza el turno */
32      void beginTurn();
33
34      /* Termina el turno */
35      void endTurn();
36 };
37
38 #endif

```

jun 09, 18 19:03

PlayerDataSender.cpp

Page 1/1

```

1  #include "PlayerDataSender.h"
2
3  PlayerDataSender::PlayerDataSender(Player& player): player(player){}
4
5  PlayerDataSender::~PlayerDataSender(){}
6
7  void PlayerDataSender::run(){
8      while (true){
9          std::unique_lock<std::mutex> lock(this->mutex);
10         while (this->queue.empty() && this->running){
11             this->condition_variable.wait(lock);
12         }
13
14         if (!this->running){
15             break;
16         }
17         try{
18             this->player.getProtocol().sendBuffer(this->queue.front());
19             this->queue.pop();
20         } catch(const SocketException& e){
21             this->player.disconnect();
22         }
23     }
24 }
25
26 void PlayerDataSender::sendData(Buffer buffer){
27     std::unique_lock<std::mutex> lock(this->mutex);
28     this->queue.push(buffer);
29 }
30
31 void PlayerDataSender::notify(){
32     this->condition_variable.notify_one();
33 }
34
35 void PlayerDataSender::stop(){
36     Thread::stop();
37     this->notify();
38 }

```

jun 12, 18 14:03

PlayerDataSender.h

Page 1/1

```

1  #ifndef __PLAYERDATASENDER_H__
2  #define __PLAYERDATASENDER_H__
3
4  #include "Thread.h"
5  #include "Player.h"
6  #include "Buffer.h"
7  #include <mutex>
8  #include <condition_variable>
9  #include <queue>
10
11 //Cola bloqueante para enviar datos a un jugador
12 class PlayerDataSender: public Thread{
13     private:
14         std::mutex mutex;
15         std::condition_variable condition_variable;
16         Player& player;
17         std::queue<Buffer> queue;
18
19     public:
20         explicit PlayerDataSender(Player& player);
21
22         ~PlayerDataSender();
23
24         //Envia datos al jugador
25         void run() override;
26
27         //Agrega un nuevo dato a la cola
28         void sendData(Buffer buffer);
29
30         //Notifica que hay nuevos datos
31         void notify();
32
33         //Termina el envio de datos
34         void stop() override;
35 };
36
37 #endif

```

jun 12, 18 14:03

ServerProtocol.cpp

Page 1/3

```

1  #include "ServerProtocol.h"
2  #include "Game.h"
3  #include "Weapon.h"
4  #include "Girder.h"
5  #include "ObjectSizes.h"
6  #include "Player.h"
7  #include "DataSender.h"
8  #include <string>
9
10 ServerProtocol::ServerProtocol(Socket&& socket): Protocol(std::move(socket)){}
11
12 ServerProtocol::ServerProtocol(ServerProtocol&& other):
13     Protocol(std::move(other)) {}
14
15 ServerProtocol::~ServerProtocol(){}
16
17 Buffer ServerProtocol::sendObject(physical_object_ptr& object){
18     Buffer buffer;
19     buffer.setNext(MOVING_OBJECT);
20
21     const std::string& type = object->getType();
22     if (type == TYPE_WORM){
23         ServerProtocol::send_worm(object, buffer);
24     } else if (type == TYPE_WEAPON){
25         ServerProtocol::send_weapon(object, buffer);
26     }
27     return std::move(buffer);
28 }
29
30 Buffer ServerProtocol::sendDeadObject(physical_object_ptr& object){
31     Buffer buffer;
32     buffer.setNext(DEAD_OBJECT);
33
34     const std::string& type = object->getType();
35     if (type == TYPE_WORM){
36         buffer.setNext(WORM_TYPE);
37     } else if (type == TYPE_WEAPON){
38         buffer.setNext(WEAPON_TYPE);
39     }
40
41     uint32_t id = object->getId();
42     ServerProtocol::sendIntBuffer(buffer, id);
43
44     return std::move(buffer);
45 }
46
47 void ServerProtocol::send_worm(physical_object_ptr& object, Buffer& buffer){
48     Worm* worm = (Worm*)object.get();
49     buffer.setNext(WORM_TYPE);
50     int32_t id = worm->getId();
51
52     b2Vec2 position = worm->getPosition();
53
54     ServerProtocol::sendIntBuffer(buffer, id);
55     ServerProtocol::sendIntBuffer(buffer, worm->getPlayerId());
56     ServerProtocol::sendIntBuffer(buffer, position.x * UNIT_TO_SEND);
57     ServerProtocol::sendIntBuffer(buffer, position.y * UNIT_TO_SEND);
58     ServerProtocol::sendIntBuffer(buffer, worm->getLife());
59     buffer.setNext(worm->getDir());
60     buffer.setNext(worm->isColliding());
61 }
62
63 void ServerProtocol::send_weapon(physical_object_ptr& object, Buffer& buffer){
64     buffer.setNext(WEAPON_TYPE);
65     ServerProtocol::sendIntBuffer(buffer, object->getId());
66 }

```

jun 12, 18 14:03

ServerProtocol.cpp

Page 2/3

```

67
68     b2Vec2 position = object->getPosition();
69     Weapon* weapon = (Weapon*)object.get();
70     std::string name = weapon->getName();
71
72     ServerProtocol::sendStringBuffer(buffer, name);
73     ServerProtocol::sendIntBuffer(buffer, position.x * UNIT_TO_SEND);
74     ServerProtocol::sendIntBuffer(buffer, position.y * UNIT_TO_SEND);
75 }
76
77 Buffer ServerProtocol::sendStartGame() {
78     Buffer buffer;
79     buffer.setNext(START_GAME_ACTION);
80     return buffer;
81 }
82
83 Buffer ServerProtocol::sendEndTurn() {
84     Buffer buffer;
85     buffer.setNext(END_TURN);
86     return buffer;
87 }
88
89 Buffer ServerProtocol::sendStartTurn(int worm_id, int player_id, float wind) {
90     Buffer buffer;
91     buffer.setNext(START_TURN);
92     ServerProtocol::sendIntBuffer(buffer, worm_id);
93     ServerProtocol::sendIntBuffer(buffer, player_id);
94     ServerProtocol::sendIntBuffer(buffer, wind * UNIT_TO_SEND);
95     return buffer;
96 }
97
98 Buffer ServerProtocol::sendTurnData(int turn_time, int time_after_shoot) {
99     Buffer buffer;
100     ServerProtocol::sendIntBuffer(buffer, turn_time);
101     ServerProtocol::sendIntBuffer(buffer, time_after_shoot);
102     return buffer;
103 }
104
105 Buffer ServerProtocol::sendPlayerId(const Player& player) {
106     Buffer buffer;
107     ServerProtocol::sendIntBuffer(buffer, player.getId());
108     ServerProtocol::sendStringBuffer(buffer, player.getName());
109     return buffer;
110 }
111
112 Buffer ServerProtocol::sendGirder(physical_object_ptr& object) {
113     Girder* girder = (Girder*)object.get();
114
115     Buffer buffer;
116     ServerProtocol::sendIntBuffer(buffer, girder->getSize());
117
118     b2Vec2 position = object->getPosition();
119     ServerProtocol::sendIntBuffer(buffer, position.x * UNIT_TO_SEND);
120     ServerProtocol::sendIntBuffer(buffer, position.y * UNIT_TO_SEND);
121     ServerProtocol::sendIntBuffer(buffer, girder->getRotation());
122     return buffer;
123 }
124
125 Buffer ServerProtocol::sendWeaponAmmo(const std::string& weapon_name, int ammo) {
126     Buffer buffer;
127     ServerProtocol::sendStringBuffer(buffer, weapon_name);
128     ServerProtocol::sendIntBuffer(buffer, ammo);
129     return buffer;
130 }
131
132 Buffer ServerProtocol::sendWeaponChanged(const std::string &weapon) {

```

jun 12, 18 14:03

ServerProtocol.cpp

Page 3/3

```

133     Buffer buffer;
134     buffer.setNext(CHANGE_WEAPON_ACTION);
135     ServerProtocol::sendStringBuffer(buffer, weapon);
136     return buffer;
137 }
138
139 Buffer ServerProtocol::sendWeaponShot(const std::string &weapon) {
140     Buffer buffer;
141     buffer.setNext(SHOOT_WEAPON_ACTION);
142     ServerProtocol::sendStringBuffer(buffer, weapon);
143     return buffer;
144 }
145
146 Buffer ServerProtocol::sendMoveAction(char action) {
147     Buffer buffer;
148     buffer.setNext(MOVE_ACTION);
149     buffer.setNext(action);
150     return buffer;
151 }
152
153 Buffer ServerProtocol::sendUpdateScope(int angle) {
154     Buffer buffer;
155     buffer.setNext(MOVE_SCOPE);
156     ServerProtocol::sendIntBuffer(buffer, angle);
157     return buffer;
158 }
159
160 Buffer ServerProtocol::sendEndGame(const std::string& winner) {
161     Buffer buffer;
162     buffer.setNext(END_GAME);
163     ServerProtocol::sendStringBuffer(buffer, winner);
164     return buffer;
165 }

```

jun 12, 18 14:03

ServerProtocol.h

Page 1/2

```

1  #ifndef __SERVERPROTOCOL_H__
2  #define __SERVERPROTOCOL_H__
3
4  #include "Socket.h"
5  #include "Protocol.h"
6  #include "PhysicalObject.h"
7  #include <mutex>
8  #include <string>
9
10 class Player;
11
12 class ServerProtocol : public Protocol{
13     private:
14         //Carga los datos del gusano en el buffer
15         static void send_worm(physical_object_ptr& object, Buffer& buffer);
16
17         //Carga los datos del arma en el buffer
18         static void send_weapon(physical_object_ptr& weapon, Buffer& buffer);
19
20     public:
21         explicit ServerProtocol(Socket&& socket);
22         ServerProtocol(ServerProtocol&& other);
23         ~ServerProtocol();
24
25         //Carga un nuevo objeto en el buffer
26         static Buffer sendObject(physical_object_ptr& object);
27
28         //Carga la informacion de un objeto muerto en el buffer
29         static Buffer sendDeadObject(physical_object_ptr& object);
30
31         //Carga la informacion de comienzo de juego
32         static Buffer sendStartGame();
33
34         //Carga la informacion de nuevo turno en el buffer
35         static Buffer sendStartTurn(int worm_id, int player_id, float wind);
36
37         //Carga la informacion del turno en el buffer
38         static Buffer sendTurnData(int turn_time, int time_after_shoot);
39
40         //Carga la informacion de un nuevo jugador en el buffer
41         static Buffer sendPlayerId(const Player& player);
42
43         //Carga la informacion de una viga en el buffer
44         static Buffer sendGirder(physical_object_ptr& girder);
45
46         //Carga la informacion de un arma en el buffer
47         static Buffer sendWeaponAmmo(const std::string& weapon_name, int ammo);
48
49         //Carga la informacion de cambio de arma en el buffer
50         static Buffer sendWeaponChanged(const std::string& weapon);
51
52         //Carga la informacion de arma disparada en el buffer
53         static Buffer sendWeaponShot(const std::string& weapon);
54
55         //Carga la informacion de que el gusano salto
56         static Buffer sendMoveAction(char action);
57
58         //Carga la informacion de cambio de angulo en el buffer
59         static Buffer sendUpdateScope(int angle);
60
61         //Carga la informacion de fin del juego en el buffer
62         static Buffer sendEndGame(const std::string& winner);
63
64         //Carga la informacion de fin del turno
65         static Buffer sendEndTurn();
66 };

```

jun 12, 18 14:03

ServerProtocol.h

Page 2/2

```

67
68 #endif

```

jun 25, 18 19:28

Game.cpp

Page 1/3

```

1  #include "Game.h"
2  #include "Girder.h"
3  #include "WeaponFactory.h"
4  #include "Server.h"
5  #include <map>
6  #include <string>
7  #include <vector>
8
9  #define TURN_STEP 100 //milliseconds
10
11 Game::Game(size_t players, const std::string& config_file,
12             const std::string& map, Server& server):
13     players(players), server(server),
14     parameters(config_file, map),
15     world(this->parameters){
16     this->running = true;
17 }
18
19 Game::~Game() {
20     this->world.stop();
21     this->world.join();
22     if (data_sender){
23         this->data_sender->stop();
24         this->data_sender->join();
25     }
26 }
27
28 bool Game::addPlayer(Player& player){
29     if (this->isFull()){
30         return false;
31     }
32     return this->turn.addPlayer(player);
33 }
34
35
36 bool Game::isFull(){
37     return this->players <= this->turn.getPlayersSize();
38 }
39
40 bool Game::playerCanJoin(const std::string& player_name){
41     if (this->isFull()){
42         return false;
43     }
44     return this->turn.playerCanJoin(player_name);
45 }
46
47 void Game::run(){
48     this->configure();
49     this->world.start();
50     this->data_sender->start();
51
52     std::this_thread::sleep_for(std::chrono::milliseconds(100));
53     this->waitToWorld();
54
55     while (!this->turn.gameEnded(this->world.getMutex())){
56         this->player_turn_active = true;
57         this->turn.beginTurn();
58         int worm_id = this->turn.getCurrentPlayer().getCurrentWorm().getId();
59         int player_id = this->turn.getCurrentPlayer().getId();
60         this->data_sender->sendStartTurn(worm_id, player_id, this->world.getWind
61     ));
62
63     size_t current_turn_time = 0;
64     size_t max_turn_time = this->parameters.getTurnTime() * 1000;
65     bool time_reduced = false;
66     while(current_turn_time < max_turn_time){

```

jun 25, 18 19:28

Game.cpp

Page 2/3

```

66         std::this_thread::sleep_for(std::chrono::milliseconds(TURN_STEP));
67         current_turn_time += TURN_STEP;
68         Worm& current_worm = this->turn.getCurrentPlayer().getCurrentWorm();
69         if (current_worm.damageReceived() || this->turn.gameEnded(world.getM
70     utex())){
71             current_turn_time = max_turn_time;
72             }else if (!time_reduced && current_worm.hasShot()){
73                 current_turn_time = max_turn_time - parameters.getTimeAfterShoot
74     () * 1000;
75                 time_reduced = true;
76             }
77         }
78         this->turn.endTurn();
79         this->data_sender->sendEndTurn();
80         this->waitToWorld();
81         this->world.update();
82     }
83     std::this_thread::sleep_for(std::chrono::milliseconds(50));
84     this->data_sender->sendEndGame(this->turn.getWinner());
85     this->world.stop();
86     this->data_sender->stop();
87     this->data_sender->join();
88     for (Player& player: this->turn.getPlayers()){
89         if (player.isConnected()){
90             this->server.addConnectedClient(std::move(player.getProtocol()));
91         }
92     }
93     this->running = false;
94 }
95
96 void Game::configure(){
97     DataSender* s = new DataSender(world, turn.getPlayers(), parameters);
98     this->data_sender.reset(s);
99     this->turn.startGame(*this->data_sender);
100
101     this->data_sender->sendStartGame();
102     this->data_sender->sendBackgroundImage(this->parameters.getBackgroundImage()
103 );
104     int turn_time = this->parameters.getTurnTime();
105     int time_after_shoot = this->parameters.getTimeAfterShoot();
106     this->data_sender->sendTurnData(turn_time, time_after_shoot);
107     this->data_sender->sendPlayersId();
108
109     //Asignacion de gusanos
110     std::vector<b2Vec2>& worms_list = this->parameters.getWorms();
111     size_t size = worms_list.size();
112     for (size_t i = 0; i < size; i++){
113         this->turn.addWorm(this->world, this->parameters, worms_list[i], i);
114     }
115     this->turn.distributeWorms(size, this->parameters.getWormsLifeToAdd());
116
117     //Creacion de vigas
118     int max_height = 0;
119     std::vector<GirderParams>& list = this->parameters.getGirders();
120     size = list.size();
121     for (size_t i = 0; i < size; i++){
122         Girder* g = new Girder(world, parameters, list[i].len, list[i].rotation)
123     ;
124         physical_object_ptr girder(g);
125         this->world.addObject(girder, b2Vec2(list[i].pos_x, list[i].pos_y));
126         if (list[i].pos_y > max_height){
127             max_height = list[i].pos_y;
128         }
129     }
130     this->parameters.setMaxHeight(max_height);

```

jun 25, 18 19:28

Game.cpp

Page 3/3

```

128     this->data_sender->sendGirders();
129
130     //Municion de las armas
131     std::map<std::string, unsigned int>& ammo = this->parameters.getWeaponsAmmo(
    );
132     this->data_sender->sendWeaponsAmmo(ammo);
133     for (Player& player: this->turn.getPlayers()){
134         player.setWeaponsAmmo(ammo);
135     }
136 }
137
138 void Game::endTurn(){
139     this->player_turn_active = false;
140 }
141
142 void Game::waitToWorld(){
143     while (this->world.isActive() || this->data_sender->isActive()){
144         int sleep = this->parameters.getGameWaitingWorldSleep();
145         std::this_thread::sleep_for(std::chrono::milliseconds(sleep));
146     }
147 }

```

jun 12, 18 14:03

Game.h

Page 1/1

```

1  #ifndef __GAME_H__
2  #define __GAME_H__
3
4  #include <vector>
5  #include <memory>
6  #include "Turn.h"
7  #include "GameParameters.h"
8  #include "Thread.h"
9  #include "Player.h"
10 #include "Worm.h"
11 #include "World.h"
12 #include "DataSender.h"
13 #include <string>
14
15 class Player;
16 class Server;
17
18 class Game: public Thread{
19     private:
20         size_t players;
21         Server& server;
22         GameParameters parameters;
23         World world;
24         Turn turn;
25         std::unique_ptr<DataSender> data_sender;
26         bool player_turn_active;
27
28         /* Realiza la configuracion inicial de la partida */
29         void configure();
30
31         /* Espera a que los objetos dejen de moverse */
32         void waitToWorld();
33
34         typedef const std::string string;
35
36     public:
37         /* Constructor */
38         Game(size_t players, string& config_file, string& map, Server& server);
39
40         /* Destructor */
41         ~Game();
42
43         /* Agrega un jugador a la partida */
44         bool addPlayer(Player& player);
45
46         /* Devuelve true si la partida esta llena */
47         bool isFull();
48
49         /* Devuelve true si el jugador puede unirse a la partida */
50         bool playerCanJoin(const std::string& player_name);
51
52         /* Comienza la partida */
53         void run() override;
54
55         /* Finaliza el turno */
56         void endTurn();
57     };
58
59 #endif

```

jun 25, 18 19:28

GameParameters.cpp

Page 1/4

```

1  #include "GameParameters.h"
2  #include "ConfigFields.h"
3  #include "Path.h"
4  #include <algorithm>
5  #include <random>
6  #include <map>
7  #include <string>
8  #include <vector>
9
10 #define WORLD_MAX_HEIGHT "world_max_height"
11
12 typedef std::vector<std::vector<float>> worms_vector;
13 typedef std::vector<std::vector<float>> girders_vector;
14
15 GameParameters::GameParameters(const std::string& config_file,
16                               const std::string& config_editor_file){
17     //Compruebo que existan todos los parametros necesarios
18     YAML::Node config(YAML::LoadFile(config_file));
19     YAML::Node config_editor(YAML::LoadFile(config_editor_file));
20
21     params[DATA_SENDER_SLEEP] = config[DATA_SENDER_SLEEP].as<float>();
22     params[GAME_WAIT_WORLD_SLEEP] = config[GAME_WAIT_WORLD_SLEEP].as<float>();
23     params[WORLD_SLEEP_AFTER_STEP] = config[WORLD_SLEEP_AFTER_STEP].as<float>();
24     params[WORLD_TIME_STEP] = config[WORLD_TIME_STEP].as<float>();
25     params[TURN_TIME] = config[TURN_TIME].as<float>();
26     params[TIME_AFTER_SHOOT] = config[TIME_AFTER_SHOOT].as<float>();
27
28     params[WORMS_LIFE] = config_editor[WORMS_LIFE].as<float>();
29     params[WORMS_LIFE_TO_ADD] = config[WORMS_LIFE_TO_ADD].as<float>();
30     params[WORM_VELOCITY] = config[WORM_VELOCITY].as<float>();
31     params[WORM_EXPLOSION_VELOCITY] = config[WORM_EXPLOSION_VELOCITY].as<float>();
32
33     params[WORM_JUMP_VELOCITY] = config[WORM_JUMP_VELOCITY].as<float>();
34     params[WORM_ROLLBACK_VELOCITY] = config[WORM_ROLLBACK_VELOCITY].as<float>();
35     params[WORM_JUMP_HEIGHT] = config[WORM_JUMP_HEIGHT].as<float>();
36     params[WORM_ROLLBACK_HEIGHT] = config[WORM_ROLLBACK_HEIGHT].as<float>();
37     params[WORM_HEIGHT_TO_DAMAGE] = config[WORM_HEIGHT_TO_DAMAGE].as<float>();
38     params[WORM_MAX_HEIGHT_DAMAGE] = config[WORM_MAX_HEIGHT_DAMAGE].as<float>();
39     params[WEAPONS_VELOCITY] = config[WEAPONS_VELOCITY].as<float>();
40     params[WIND_MIN_VELOCITY] = config[WIND_MIN_VELOCITY].as<float>();
41     params[WIND_MAX_VELOCITY] = config[WIND_MAX_VELOCITY].as<float>();
42     params[GRAVITY] = config[GRAVITY].as<float>();
43     params[AIR_MISSILES_SEPARATION] = config[AIR_MISSILES_SEPARATION].as<float>();
44
45     params[GIRDER_ANGLE_FRICTION] = config[GIRDER_ANGLE_FRICTION].as<float>();
46     params[WORLD_MAX_HEIGHT] = 99999;
47
48     weapon_radius = config[WEAPON_RADIUS].as<std::map<std::string, int>>();
49     weapon_ammo =
50         config_editor[WEAPON_AMMO].as<std::map<std::string, unsigned int>>();
51     weapon_damage = config[WEAPON_DAMAGE].as<std::map<std::string, int>>();
52     weapon_fragments = config[WEAPON_FRAGMENTS].as<std::map<std::string, int>>();
53
54     worms_vector worms_file = config_editor[WORMS_DATA].as<worms_vector>();
55     for (std::vector<float>& worm: worms_file){
56         this->worms.push_back(b2Vec2(worm[0], worm[1]));
57     }
58     girders_vector girders_file = config_editor[GIRDERS_DATA].as<girders_vector>();
59     for (std::vector<float>& girder: girders_file){
60         this->girders.push_back(GirderParams(girder[0], girder[1], girder[2], gi

```

jun 25, 18 19:28

GameParameters.cpp

Page 2/4

```

61
62     std::vector<int> bg = config_editor[BACKGROUND_IMAGE].as<std::vector<int>>();
63
64     ;
65     Buffer buffer(bg.size());
66     for (int byte: bg){
67         buffer.setNext(byte);
68     }
69     this->background_image = std::move(buffer);
70
71     GameParameters::~GameParameters() {}
72
73     int GameParameters::getWormLife(){
74         return this->params[WORMS_LIFE];
75     }
76
77     int GameParameters::getWormsLifeToAdd(){
78         return this->params[WORMS_LIFE_TO_ADD];
79     }
80
81     std::vector<b2Vec2>& GameParameters::getWorms(){
82         std::random_device rd;
83         std::mt19937 random(rd());
84
85         std::shuffle(this->worms.begin(), this->worms.end(), random);
86         return this->worms;
87     }
88
89     std::vector<GirderParams>& GameParameters::getGirders(){
90         return this->girders;
91     }
92
93     std::map<std::string, unsigned int>& GameParameters::getWeaponsAmmo(){
94         return this->weapon_ammo;
95     }
96
97     float GameParameters::getWormVelocity(){
98         return this->params[WORM_VELOCITY];
99     }
100
101     float GameParameters::getWormExplosionVelocity(){
102         return this->params[WORM_EXPLOSION_VELOCITY];
103     }
104
105     float GameParameters::getWormJumpVelocity(){
106         return this->params[WORM_JUMP_VELOCITY];
107     }
108
109     float GameParameters::getWormRollbackVelocity(){
110         return this->params[WORM_ROLLBACK_VELOCITY];
111     }
112
113     float GameParameters::getWormJumpHeight(){
114         return this->params[WORM_JUMP_HEIGHT];
115     }
116
117     float GameParameters::getWormRollbackHeight(){
118         return this->params[WORM_ROLLBACK_HEIGHT];
119     }
120
121     int GameParameters::getWormHeightToDamage(){
122         return this->params[WORM_HEIGHT_TO_DAMAGE];
123     }
124
125     int GameParameters::getWormMaxHeightDamage(){
126         return this->params[WORM_MAX_HEIGHT_DAMAGE];

```


jun 25, 18 19:28

GameParameters.cpp

Page 3/4

```

126 }
127
128 float GameParameters::getWeaponsVelocity() {
129     return this->params[WEAPONS_VELOCITY];
130 }
131
132 int GameParameters::getWeaponDamage(const std::string& weapon) {
133     return this->weapon_damage[weapon];
134 }
135
136 int GameParameters::getWeaponRadius(const std::string& weapon) {
137     return this->weapon_radius[weapon];
138 }
139
140 int GameParameters::getWeaponFragments(const std::string& weapon) {
141     return this->weapon_fragments[weapon];
142 }
143
144 float GameParameters::getWindMinVelocity() {
145     return this->params[WIND_MIN_VELOCITY];
146 }
147
148 float GameParameters::getWindMaxVelocity() {
149     return this->params[WIND_MAX_VELOCITY];
150 }
151
152 float GameParameters::getGravity() {
153     return this->params[GRAVITY];
154 }
155
156 float GameParameters::getAirMissilesSeparation() {
157     return this->params[AIR_MISSILES_SEPARATION];
158 }
159
160 int GameParameters::getMaxGirderRotationToFriction() {
161     return this->params[GIRDER_ANGLE_FRICTION];
162 }
163
164 void GameParameters::setMaxHeight(int height) {
165     this->params[WORLD_MAX_HEIGHT] = height + 10;
166 }
167
168 int GameParameters::getMaxHeight() {
169     return this->params[WORLD_MAX_HEIGHT];
170 }
171
172 int GameParameters::getDataSenderSleep() {
173     return this->params[DATA_SENDER_SLEEP];
174 }
175
176 int GameParameters::getGameWaitingWorldSleep() {
177     return this->params[GAME_WAIT_WORLD_SLEEP];
178 }
179
180 int GameParameters::getWorldSleepAfterStep() {
181     return this->params[WORLD_SLEEP_AFTER_STEP];
182 }
183
184 float GameParameters::getWorldTimeStep() {
185     return this->params[WORLD_TIME_STEP];
186 }
187
188 int GameParameters::getTurnTime() {
189     return this->params[TURN_TIME];
190 }
191

```

jun 25, 18 19:28

GameParameters.cpp

Page 4/4

```

192 int GameParameters::getTimeAfterShoot() {
193     return this->params[TIME_AFTER_SHOOT];
194 }
195
196 Buffer& GameParameters::getBackgroundImage() {
197     return this->background_image;
198 }
199
200 GameParameters::GirderParams::GirderParams(size_t len, float pos_x,
201                                             float pos_y, int rotation):
202     len(len), pos_x(pos_x), pos_y(pos_y), rotation(rotation) {}

```

jun 19, 18 13:48	GameParameters.h	Page 1/3
<pre> 1 #ifndef __GAMEPARAMETERS_H__ 2 #define __GAMEPARAMETERS_H__ 3 4 #include <string> 5 #include <vector> 6 #include <map> 7 #include "b2Math.h" 8 #include "yaml.h" 9 #include "Buffer.h" 10 11 // Clase que lee los archivos de configuracion 12 // y devuelve los parametros obtenidos 13 class GameParameters{ 14 public: 15 class GirderParams; 16 17 private: 18 std::map<std::string, float> params; 19 std::map<std::string, int> weapon_radius; 20 std::map<std::string, unsigned int> weapon_amm; 21 std::map<std::string, int> weapon_damage; 22 std::map<std::string, int> weapon_fragments; 23 24 std::vector<b2Vec2> worms; 25 std::vector<GirderParams> girders; 26 Buffer background_image; 27 28 public: 29 //Inicializa todos los parametros necesarios para la partida 30 GameParameters(const std::string& config, const std::string& editor_file 31); 32 ~GameParameters(); 33 34 //Devuelve la vida del worm 35 int getWormLife(); 36 37 //Devuelve la vida a agregar de los worms 38 int getWormsLifeToAdd(); 39 40 //Devuelve los worms del mapa 41 std::vector<b2Vec2>& getWorms(); 42 43 //Devuelve la vigas del mapa 44 std::vector<GirderParams>& getGirders(); 45 46 //Devuelve la municion de las armas 47 std::map<std::string, unsigned int>& getWeaponsAmmo(); 48 49 //Devuelve la velocidad del worm 50 float getWormVelocity(); 51 52 //Devuelve la velocidad del worm debido a una explosion 53 float getWormExplosionVelocity(); 54 55 //Devuelve la velocidad de salto del worm 56 float getWormJumpVelocity(); 57 58 //Devuelve la velocidad del rollback del worm 59 float getWormRollbackVelocity(); 60 61 //Devuelve la altura de salto del worm 62 float getWormJumpHeight(); 63 64 //Devuelve la altura del rollback del worm 65 float getWormRollbackHeight(); 66 </pre>		

jun 19, 18 13:48	GameParameters.h	Page 2/3
<pre> 66 67 //Devuelve la altura en la cual el worm sufre daÑo 68 int getWormHeightToDamage(); 69 70 //Devuelve el daÑo maximo por caida 71 int getWormMaxHeightDamage(); 72 73 74 //Devuelve la velocidad del arma 75 float getWeaponsVelocity(); 76 77 //Devuelve el daÑo del arma 78 int getWeaponDamage(const std::string& weapon); 79 80 //Devuelve el radio de daÑo del arma 81 int getWeaponRadius(const std::string& weapon); 82 83 //Devuelve la cantidad de fragmentos del arma 84 int getWeaponFragments(const std::string& weapon); 85 86 //Devuelve la velocidad minima del viento 87 float getWindMinVelocity(); 88 89 //Devuelve la velocidad maxima del viento 90 float getWindMaxVelocity(); 91 92 //Devuelve la gravedad 93 float getGravity(); 94 95 //Devuelve la separacion de los misiles aereos 96 float getAirMissilesSeparation(); 97 98 //Devuelve la rotacion maxima para la cual 99 //el gusano no desliza 100 int getMaxGirderRotationToFriction(); 101 102 //Establece la altura maxima 103 void setMaxHeight(int height); 104 105 //Devuelve la altura maxima 106 int getMaxHeight(); 107 108 //Devuelve el tiempo de sleep del DataSender 109 int getDataSenderSleep(); 110 111 //Devuelve el tiempo de sleep del World 112 int getGameWaitingWorldSleep(); 113 114 //Devuelve el tiempo de sleep del step del World 115 int getWorldSleepAfterStep(); 116 117 //Devuelve el time step del World 118 float getWorldTimeStep(); 119 120 //Devuelve el tiempo del turno 121 int getTurnTime(); 122 123 //Devuelve el tiempo adicional luego de un disparo 124 int getTimeAfterShoot(); 125 126 //Devuelve la imagen de fondo 127 Buffer& getBackgroundImage(); 128 }; 129 130 class GameParameters::GirderParams{ 131 public: </pre>		

jun 19, 18 13:48

GameParameters.h

Page 3/3

```

132     size_t len;
133     float pos_x;
134     float pos_y;
135     int rotation;
136
137     GirderParams(size_t len, float pos_x, float pos_y, int rotation);
138 };
139
140 typedef GameParameters::GirderParams GirderParams;
141
142 #endif

```

jun 12, 18 14:03

Player.cpp

Page 1/2

```

1  #include "Player.h"
2  #include <map>
3  #include <string>
4
5  Player::Player(ServerProtocol&& protocol): protocol(std::move(protocol)),
6      id(-1), connected(true){}
7
8  Player::Player(Player&& other):
9      protocol(std::move(other.protocol)), name(std::move(other.name)),
10      worms(std::move(other.worms)), id(other.id), connected(other.connected){}
11
12  Player::~Player(){}
13
14  void Player::setId(int id){
15      this->id = id;
16  }
17
18  int Player::getId() const{
19      return this->id;
20  }
21
22  Worm& Player::getCurrentWorm(){
23      return this->worms.getCurrentWorm();
24  }
25
26  void Player::beginTurn(){
27      this->worms.beginTurn();
28  }
29
30  void Player::addWorm(World& world,
31                      GameParameters& params, const b2Vec2& position, int id){
32      physical_object_ptr worm(new Worm(world, params, id, this->id, this->weapons
33      ));
34      this->worms.add(worm);
35      world.addObject(worm, position);
36  }
37
38  void Player::distributeWorms(size_t max, int life_to_add){
39      this->worms.distribute(max, life_to_add);
40  }
41
42  bool Player::isDead(){
43      return this->worms.isEmpty();
44  }
45
46  ServerProtocol& Player::getProtocol(){
47      return this->protocol;
48  }
49
50  void Player::setName(const std::string& name){
51      this->name = name;
52  }
53
54  const std::string& Player::getName() const{
55      return this->name;
56  }
57
58  bool Player::isConnected() const{
59      return this->connected;
60  }
61
62  void Player::disconnect(){
63      this->connected = false;
64      this->worms.kill();
65  }

```

jun 12, 18 14:03

Player.cpp

Page 2/2

```

66 void Player::setWeaponsAmmo(const std::map<std::string, unsigned int>& ammo){
67     this->weapons.updateAmmo(ammo);
68 }
69
70 void Player::changeWeapon(const std::string& weapon){
71     this->weapons.changeWeapon(weapon);
72 }

```

jun 12, 18 14:03

Player.h

Page 1/2

```

1  #ifndef __PLAYER_H__
2  #define __PLAYER_H__
3
4  #include "WormsList.h"
5  #include "ServerProtocol.h"
6  #include "Worm.h"
7  #include "World.h"
8  #include "GameParameters.h"
9  #include "WeaponList.h"
10 #include <string>
11 #include <map>
12
13 class Player{
14     private:
15         ServerProtocol protocol;
16         std::string name;
17         WormsList worms;
18         WeaponList weapons;
19         int id;
20         bool connected;
21
22     public:
23         explicit Player(ServerProtocol&& protocol);
24
25         Player(Player&& other);
26
27         ~Player();
28
29         //Setea el id del jugador por el pasado
30         void setId(int id);
31
32         //Devuelve el id del jugador
33         int getId() const;
34
35         //Devuelve el gusano actual del jugador
36         Worm& getCurrentWorm();
37
38         //Empieza el turno del jugador
39         void beginTurn();
40
41         //Agrega un nuevo gusano al jugador
42         void addWorm(World& world, GameParameters& param, const b2Vec2& pos, int
id);
43
44         //Agrega vida a los gusanos del jugador
45         //en caso de que tenga menos gusanos que otros jugadores
46         void distributeWorms(size_t max, int life_to_add);
47
48         //Devuelve true si el jugador esta muerto
49         bool isDead();
50
51         //Devuelve true si el jugador esta desconectado
52         bool isConnected() const;
53
54         //Desconecta al jugador
55         void disconnect();
56
57         //Setea la municion de las armas
58         void setWeaponsAmmo(const std::map<std::string, unsigned int>& ammo);
59
60         //Cambia el arma actual del jugador
61         void changeWeapon(const std::string& weapon);
62
63         //Setea el nombre del jugador
64         void setName(const std::string& name);
65

```

jun 12, 18 14:03

Player.h

Page 2/2

```

66      //Devuelve el nombre del jugador
67      const std::string& getName() const;
68
69      //Devuelve el protocolo del jugador
70      ServerProtocol& getProtocol();
71  };
72
73  #endif

```

jun 25, 18 19:28

Turn.cpp

Page 1/2

```

1  #include "Turn.h"
2  #include <string>
3  #include <vector>
4
5  Turn::Turn(): current(0){}
6
7  Turn::~Turn(){
8      for (std::unique_ptr<PlayerDataReceiver>& receiver: this->receivers){
9          receiver->stop();
10         receiver->join();
11     }
12 }
13
14 bool Turn::addPlayer(Player& player){
15     if (!this->playerCanJoin(player.getName())){
16         return false;
17     }
18     player.setId(this->players.size());
19     player.getProtocol().sendChar(true);
20     this->players.push_back(std::move(player));
21     return true;
22 }
23
24 bool Turn::playerCanJoin(const std::string& player_name){
25     for (Player& player: this->players){
26         if (player.getName() == player_name){
27             return false;
28         }
29     }
30     return true;
31 }
32
33 size_t Turn::getPlayersSize() const{
34     return this->players.size();
35 }
36
37 Player& Turn::getCurrentPlayer(){
38     return this->players.at(this->current);
39 }
40
41 void Turn::startGame(DataSender& data_sender){
42     for (Player& player: this->players){
43         PlayerDataReceiver* r = new PlayerDataReceiver(player, data_sender);
44         std::unique_ptr<PlayerDataReceiver> receiver(r);
45         receiver->start();
46         this->receivers.push_back(std::move(receiver));
47     }
48 }
49
50 void Turn::beginTurn(){
51     do {
52         this->advanceCurrent();
53     } while (this->getCurrentPlayer().isDead());
54     this->getCurrentPlayer().beginTurn();
55     this->receivers[this->current]->beginTurn();
56 }
57
58 void Turn::endTurn(){
59     this->receivers[this->current]->endTurn();
60 }
61
62 std::vector<Player>& Turn::getPlayers(){
63     return this->players;
64 }
65
66 void Turn::advanceCurrent(){

```

jun 25, 18 19:28

Turn.cpp

Page 2/2

```

67     this->current++;
68     if (this->current >= this->players.size()){
69         this->current = 0;
70     }
71 }
72
73 void Turn::addWorm(World& world, GameParameters& params, b2Vec2 pos, int id){
74     this->players[this->current].addWorm(world, params, pos, id);
75     this->advanceCurrent();
76 }
77
78 void Turn::distributeWorms(size_t size, int life_to_add){
79     int quantity = (size / this->players.size());
80     if (size % this->players.size() != 0){
81         quantity += 1;
82     }
83
84     for (Player& player: this->players){
85         player.distributeWorms(quantity, life_to_add);
86     }
87 }
88
89 bool Turn::gameEnded(std::mutex& mutex){
90     std::lock_guard<std::mutex> lock(mutex);
91     this->winner.clear();
92     size_t players_alive = 0;
93     for (Player& player: this->players){
94         if (!player.isDead()){
95             players_alive++;
96             this->winner = player.getName();
97         }
98     }
99     return players_alive <= 1;
100 }
101
102 const std::string& Turn::getWinner(){
103     for (std::unique_ptr<PlayerDataReceiver>& receiver: this->receivers){
104         receiver->stop();
105     }
106     return this->winner;
107 }

```

jun 12, 18 14:03

Turn.h

Page 1/1

```

1  #ifndef __SERVERTURN_H__
2  #define __SERVERTURN_H__
3
4  #include "Player.h"
5  #include "PlayerDataReceiver.h"
6  #include "DataSender.h"
7  #include <vector>
8  #include <string>
9  #include <memory>
10
11 class Turn{
12     private:
13         std::vector<Player> players;
14         std::vector<std::unique_ptr<PlayerDataReceiver>> receivers;
15         std::string winner;
16         size_t current;
17
18         void advanceCurrent();
19
20     public:
21         Turn();
22         ~Turn();
23
24         //Agrega un nuevo jugador
25         bool addPlayer(Player& player);
26
27         //Devuelve true si el jugador se puede unir a la partida
28         bool playerCanJoin(const std::string& player_name);
29
30         //Devuelve la cantidad de jugadores
31         size_t getPlayersSize() const;
32
33         //Devuelve un vector con los jugadores
34         std::vector<Player>& getPlayers();
35
36         //Devuelve el jugador actual
37         Player& getCurrentPlayer();
38
39         //Realiza la configuracion inicial
40         void startGame(DataSender& data_sender);
41
42         //Empieza un nuevo turno, cambiando el jugador actual
43         void beginTurn();
44
45         //Termina el turno del jugador actual
46         void endTurn();
47
48         //Agrega un gusano al proximo jugador
49         void addWorm(World& world, GameParameters& params, b2Vec2 pos, int id);
50
51         //Agrega vida a los jugadores con menos gusanos
52         void distributeWorms(size_t size, int life_to_add);
53
54         //Devuelve true si queda uno o ningun jugador vivo
55         bool gameEnded(std::mutex& mutex);
56
57         //Devuelve el nombre del jugador ganador
58         const std::string& getWinner();
59 };
60
61 #endif

```

jun 12, 18 14:03

WeaponList.cpp

Page 1/1

```

1  #include "WeaponList.h"
2  #include "WeaponNames.h"
3  #include "WeaponFactory.h"
4  #include <map>
5  #include <string>
6
7  WeaponList::WeaponList(): current_weapon(DEFAULT_WEAPON){}
8
9  WeaponList::~WeaponList(){}
10
11 void WeaponList::updateAmmo(const std::map<std::string, unsigned int>& ammo){
12     this->ammo = ammo;
13 }
14
15 bool WeaponList::shoot(){
16     if (this->ammo[this->current_weapon] == 0){
17         return false;
18     }
19     this->ammo[this->current_weapon]--;
20     return true;
21 }
22
23 physical_object_ptr WeaponList::getCurrentWeapon(World& world,
24                                             GameParameters& parameters){
25     WeaponFactory factory(world, parameters);
26     return factory.getWeapon(this->current_weapon);
27 }
28
29 void WeaponList::changeWeapon(const std::string& weapon){
30     this->current_weapon = weapon;
31 }

```

jun 12, 18 14:03

WeaponList.h

Page 1/1

```

1  #ifndef __WEAPONLIST_H__
2  #define __WEAPONLIST_H__
3
4  #include <map>
5  #include <string>
6  #include "PhysicalObject.h"
7
8  class GameParameters;
9
10 class WeaponList{
11     private:
12         std::map<std::string, unsigned int> ammo;
13         std::string current_weapon;
14
15     public:
16         WeaponList();
17
18         ~WeaponList();
19
20         //Actualiza la municion de las armas
21         void updateAmmo(const std::map<std::string, unsigned int>& ammo);
22
23         //Devuelve si puede disparar el arma, y disminuye la municion
24         bool shoot();
25
26         //Devuelve el arma actual
27         physical_object_ptr getCurrentWeapon(World& world, GameParameters& param
28 s);
29
30         //Cambia el arma actual
31         void changeWeapon(const std::string& weapon);
32     };
33 #endif

```

jun 25, 18 19:28

WormsList.cpp

Page 1/1

```

1  #include "WormsList.h"
2
3  WormsList::WormsList(): current(0){}
4
5  WormsList::~WormsList(){}
6
7  Worm& WormsList::getCurrentWorm(){
8      Worm* worm = (Worm*)this->list[this->current].get();
9      return *worm;
10 }
11
12 void WormsList::beginTurn(){
13     do {
14         this->current++;
15         if (this->current >= this->list.size()){
16             this->current = 0;
17         }
18     } while (this->getCurrentWorm().isDead());
19     this->getCurrentWorm().beginTurn();
20 }
21
22 void WormsList::add(physical_object_ptr worm){
23     this->list.push_back(worm);
24 }
25
26 WormsList::WormsList(WormsList&& other):
27     list(std::move(other.list)), current(other.current){}
28
29 void WormsList::distribute(size_t max, int life_to_add){
30     if (this->list.size() < max){
31         for (physical_object_ptr& worm_ptr: this->list){
32             Worm* worm = (Worm*)worm_ptr.get();
33             worm->addLife(life_to_add);
34         }
35     }
36 }
37
38 bool WormsList::isEmpty(){
39     for (physical_object_ptr& worm: this->list){
40         if (!worm->isDead()){
41             return false;
42         }
43     }
44     return true;
45 }
46
47 void WormsList::kill(){
48     for (physical_object_ptr& worm: this->list){
49         if (!worm->isDead()){
50             worm->kill();
51         }
52     }
53 }

```

jun 09, 18 19:03

WormsList.h

Page 1/1

```

1  #ifndef __WORMSLIST_H__
2  #define __WORMSLIST_H__
3
4  #include <vector>
5  #include "Worm.h"
6
7  class WormsList{
8      private:
9          std::vector<physical_object_ptr> list;
10         size_t current;
11
12     public:
13         /* Constructor */
14         WormsList();
15
16         /* Destructor */
17         ~WormsList();
18
19         /* Devuelve el worm actual */
20         Worm& getCurrentWorm();
21
22         /* Comienza el turno, cambiando el gusano actual */
23         void beginTurn();
24
25         /* Agrega un worm a la lista */
26         void add(physical_object_ptr worm);
27
28         /* Constructor por movimiento */
29         WormsList(WormsList&& other);
30
31         /* Aumenta la vida de los worms si la cantidad de
32          * worms es menor que la de otros jugadores */
33         void distribute(size_t max, int life_to_add);
34
35         /* Devuelve true si todos los worms estan muertos */
36         bool isEmpty();
37
38         /* Mata a todos los worms */
39         void kill();
40     };
41
42 #endif

```


jun 12, 18 14:03

CollisionData.cpp

Page 1/1

```

1  #include "CollisionData.h"
2  #include "PhysicalObject.h"
3  #include <string>
4
5  CollisionData::CollisionData(std::string type, PhysicalObject* object):
6      type(type), object(object){}
7
8  CollisionData::~CollisionData(){}
9
10 const std::string& CollisionData::getType(){
11     return this->type;
12 }
13
14 PhysicalObject* CollisionData::getObject(){
15     return this->object;
16 }

```

jun 12, 18 14:03

CollisionData.h

Page 1/1

```

1  #ifndef __COLLISIONDATA_H__
2  #define __COLLISIONDATA_H__
3
4  #include <string>
5
6
7  class PhysicalObject;
8
9  //Datos de un objeto para determinar colisiones
10 class CollisionData{
11     private:
12         std::string type;
13         PhysicalObject* object;
14
15     public:
16         CollisionData(std::string type, PhysicalObject* object);
17         ~CollisionData();
18
19         //Devuelve el tipo del objeto fisico
20         const std::string& getType();
21
22         //Devuelve el objeto fisico
23         PhysicalObject* getObject();
24 };
25
26 #endif

```

jun 12, 18 14:03

CollisionListener.cpp

Page 1/2

```

1  #include "CollisionListener.h"
2  #include "PhysicalObject.h"
3  #include "Worm.h"
4  #include "Girder.h"
5
6  CollisionListener::CollisionListener() {}
7
8  CollisionListener::~CollisionListener() {}
9
10 void CollisionListener::BeginContact(b2Contact* contact){
11     CollisionData* dataA =
12         (CollisionData*) contact->GetFixtureA()->GetBody()->GetUserData()
13 ;
14     CollisionData* dataB =
15         (CollisionData*) contact->GetFixtureB()->GetBody()->GetUserData()
16 ;
17     if (dataA->getObject()->isDead() || dataB->getObject()->isDead()){
18         return;
19     }
20     if (dataA->getType() == TYPE_WEAPON){
21         if (dataB->getType() == TYPE_WORM){
22             int shooter_id = ((Weapon*)dataA->getObject()->getShooterId();
23             int worm_id = dataB->getObject()->getId();
24             if (shooter_id == worm_id){
25                 return;
26             }
27         }
28         dataA->getObject()->collideWithSomething(dataB);
29     } else if (dataB->getType() == TYPE_WEAPON){
30         if (dataA->getType() == TYPE_WORM){
31             int shooter_id = ((Weapon*)dataB->getObject()->getShooterId();
32             int worm_id = dataA->getObject()->getId();
33             if (shooter_id == worm_id){
34                 return;
35             }
36         }
37         dataB->getObject()->collideWithSomething(dataA);
38     }
39     if (dataA->getType() == TYPE_WORM && contact->GetFixtureA()->IsSensor() &&
40         (dataB->getType() == TYPE_GIRDER || dataB->getType() == TYPE_BORDER)
41 ) {
42     dataA->getObject()->collideWithSomething(dataB);
43 }
44 } else if (dataB->getType() == TYPE_WORM &&
45     contact->GetFixtureB()->IsSensor() &&
46     (dataA->getType() == TYPE_GIRDER || dataA->getType() == TYPE_BORDER)
47 ) {
48     dataB->getObject()->collideWithSomething(dataA);
49 }
50 }
51 void CollisionListener::EndContact(b2Contact* contact){
52     CollisionData* dataA =
53         (CollisionData*) contact->GetFixtureA()->GetBody()->GetUserData()
54 ;
55     CollisionData* dataB =
56         (CollisionData*) contact->GetFixtureB()->GetBody()->GetUserData()
57 ;
58     if (dataA->getType() == TYPE_WORM &&
59         contact->GetFixtureA()->IsSensor() && dataB->getType() == TYPE_GIRDE
60 R) {
61     bool friction = ((Girder *) dataB->getObject()->hasFriction();

```

jun 12, 18 14:03

CollisionListener.cpp

Page 2/2

```

60     ((Worm *) dataA->getObject()->endCollisionGirder(friction);
61 } else if (dataB->getType() == TYPE_WORM &&
62     contact->GetFixtureB()->IsSensor() && dataA->getType() == TYPE_G
63 IRDER) {
64     bool friction = ((Girder *) dataA->getObject()->hasFriction();
65     ((Worm *) dataB->getObject()->endCollisionGirder(friction);
66 }
67 if (dataA->getType() == TYPE_WEAPON){
68     ((Weapon*)dataA->getObject()->removeShooterId();
69 }
70 if (dataB->getType() == TYPE_WEAPON){
71     ((Weapon*)dataB->getObject()->removeShooterId();
72 }
73 }
74
75 bool CollisionListener::ShouldCollide(b2Fixture* fixtureA, b2Fixture* fixtureB){
76     CollisionData* dataA = (CollisionData*) fixtureA->GetBody()->GetUserData();
77     CollisionData* dataB = (CollisionData*) fixtureB->GetBody()->GetUserData();
78
79     if (dataA->getType() == TYPE_WORM && dataB->getType() == TYPE_WORM){
80         return false;
81     }
82     if (dataA->getType() == TYPE_WEAPON && dataB->getType() == TYPE_WEAPON){
83         return false;
84     }
85     return true;
86 }

```

jun 09, 18 19:03

CollisionListener.h

Page 1/1

```

1  #ifndef __COLLISIONLISTENER_H__
2  #define __COLLISIONLISTENER_H__
3
4  #include <string>
5  #include "CollisionData.h"
6  #include "b2WorldCallbacks.h"
7  #include "b2Contact.h"
8  #include <list>
9
10 class CollisionListener: public b2ContactListener, public b2ContactFilter{
11     public:
12         CollisionListener();
13         ~CollisionListener();
14
15         //Analiza la colision entre dos objetos
16         void BeginContact(b2Contact* contact) override;
17
18         //Analiza el fin de colision entre dos objetos
19         void EndContact(b2Contact* contact) override;
20
21         //Analiza si dos objetos deben colisionar o no
22         bool ShouldCollide(b2Fixture* fixtureA, b2Fixture* fixtureB) override;
23 };
24
25 #endif

```

jun 25, 18 19:28

RayCastWeaponExploded.cpp

Page 1/1

```

1  #include "RayCastWeaponExploded.h"
2  #include "Worm.h"
3
4  RayCastWeaponExploded::RayCastWeaponExploded(): closest(NULL){}
5
6  RayCastWeaponExploded::~RayCastWeaponExploded(){}
7
8  b2Body* RayCastWeaponExploded::getClosestWorm(){
9      if (!this->closest){
10         return NULL;
11     }
12     CollisionData* data = (CollisionData*)this->closest->GetUserData();
13     if (data->getType() != TYPE_WORM){
14         this->closest = NULL;
15         return NULL;
16     }
17
18     this->affected_worms.push_back(this->closest);
19     b2Body* closest_worm = this->closest;
20     this->closest = NULL;
21     return closest_worm;
22 }
23
24 float32 RayCastWeaponExploded::ReportFixture(b2Fixture* fixture,
25     const b2Vec2& point, const b2Vec2& normal, float32 fract
26 ion){
27     b2Body* closest_body = fixture->GetBody();
28     for (b2Body* affected_worm: this->affected_worms){
29         if (affected_worm == closest_body){
30             return -1;
31         }
32     }
33     this->closest = closest_body;
34     return fraction;
35 }

```

jun 12, 18 14:03

RayCastWeaponExploded.h

Page 1/1

```

1  #ifndef __RAYCASTWEAPONEXPLODED_H__
2  #define __RAYCASTWEAPONEXPLODED_H__
3
4  #include "b2Body.h"
5  #include "b2Fixture.h"
6  #include "b2WorldCallbacks.h"
7  #include <vector>
8
9  class RayCastWeaponExploded: public b2RayCastCallback{
10     private:
11         std::vector<b2Body*> affected_worms;
12         b2Body* closest;
13
14     public:
15         RayCastWeaponExploded();
16         ~RayCastWeaponExploded();
17
18         //Devuelve el gusano mas cercano a la explosion, si hay
19         b2Body* getClosestWorm();
20
21         //Busca al objeto mas cercano a la explosion
22         float32 ReportFixture(b2Fixture* fixture, const b2Vec2& point,
23                               const b2Vec2& normal, float32 fraction) override;
24 };
25
26 #endif
27

```

jun 12, 18 14:03

BottomBorder.cpp

Page 1/1

```

1  #include "BottomBorder.h"
2  #include "b2PolygonShape.h"
3  #include "b2Fixture.h"
4
5  BottomBorder::BottomBorder(World& world):
6      PhysicalObject(world, 0, TYPE_BORDER){}
7
8  BottomBorder::~BottomBorder(){}
9
10 void BottomBorder::getBodyDef(b2BodyDef& body_def, const b2Vec2& pos){
11     body_def.type = b2_staticBody;
12     body_def.position.Set(pos.x, pos.y);
13 }
14
15 void BottomBorder::createFixtures(){
16     b2PolygonShape boxShape;
17     boxShape.SetAsBox(100000,1);
18
19     b2FixtureDef boxFixtureDef;
20     boxFixtureDef.shape = &boxShape;
21     boxFixtureDef.density = 1;
22     this->body->CreateFixture(&boxFixtureDef);
23 }

```

jun 12, 18 14:03

BottomBorder.h

Page 1/1

```

1  #ifndef __BOTTOMBORDER_H__
2  #define __BOTTOMBORDER_H__
3
4  #include "PhysicalObject.h"
5  #include <string>
6
7  //Determina el borde inferior del mundo
8  class BottomBorder: public PhysicalObject{
9      private:
10         std::string type;
11
12     protected:
13         void getBodyDef(b2BodyDef& body_def, const b2Vec2& pos) override;
14         void createFixtures() override;
15
16     public:
17         explicit BottomBorder(World& world);
18         ~BottomBorder();
19 };
20
21 #endif

```

jun 12, 18 14:03

Girder.cpp

Page 1/1

```

1  #include "Girder.h"
2  #include "b2PolygonShape.h"
3  #include "b2Fixture.h"
4  #include "Math.h"
5
6  Girder::Girder(World& world, GameParameters& params, size_t size, int angle):
7      PhysicalObject(world, 0, TYPE_GIRDER, size(size), rotation(angle),
8          max_rotation_to_friction(params.getMaxGirderRotationToFriction())){}
9
10 Girder::~Girder(){}
11
12 void Girder::getBodyDef(b2BodyDef& body_def, const b2Vec2& pos){
13     body_def.type = b2_staticBody;
14     body_def.position.Set(pos.x, pos.y);
15 }
16
17 void Girder::createFixtures(){
18     b2PolygonShape boxShape;
19     boxShape.SetAsBox(this->size / 2.0, girder_height / 2,
20         b2Vec2(0, 0), Math::degreesToRadians(this->rotation));
21
22     b2FixtureDef boxFixtureDef;
23     boxFixtureDef.shape = &boxShape;
24     boxFixtureDef.density = 1;
25     this->body->CreateFixture(&boxFixtureDef);
26 }
27
28 size_t Girder::getSize(){
29     return this->size;
30 }
31
32 int Girder::getRotation(){
33     return this->rotation;
34 }
35
36 bool Girder::hasFriction(){
37     int angle = this->getAngle();
38     return angle < this->max_rotation_to_friction || angle == 90;
39 }
40
41 int Girder::getAngle(){
42     int angle = this->rotation;
43     if (angle > 90){
44         angle = 180 - angle;
45     }
46     return angle;
47 }

```

jun 12, 18 14:03

Girder.h

Page 1/1

```

1  #ifndef __GIRDER_H__
2  #define __GIRDER_H__
3
4  #include "PhysicalObject.h"
5  #include "GameParameters.h"
6
7  class Girder: public PhysicalObject{
8  private:
9      size_t size;
10     int rotation;
11     int max_rotation_to_friction;
12
13     protected:
14         void getBodyDef(b2BodyDef& body_def, const b2Vec2& pos) override;
15         void createFixtures() override;
16
17     public:
18         Girder(World& world, GameParameters& parameters, size_t size, int rotation);
19         ~Girder();
20
21         //Devuelve la longitud de la viga
22         size_t getSize();
23
24         //Devuelve la rotacion de la viga
25         int getRotation();
26
27         //Devuelve true si la viga tiene friccion
28         bool hasFriction();
29
30         //Devuelve la rotacion normalizada
31         int getAngle();
32 };
33
34 #endif

```

jun 12, 18 14:03

PhysicalObject.cpp

Page 1/2

```

1  #include "PhysicalObject.h"
2  #include "World.h"
3  #include <string>
4
5  PhysicalObject::PhysicalObject(World& world, int id, const std::string& type):
6      world(world), body(NULL), is_dead(false), id(id),
7      type(type), last_position(-1, -1),
8      last_position_sent(false), data_updated(false),
9      collision_data(type, this){}
10
11  PhysicalObject::~PhysicalObject() {}
12
13  void PhysicalObject::initializeBody(b2Body* body){
14      this->body = body;
15      this->body->SetUserData(&this->collision_data);
16      this->createFixtures();
17      this->setInitialVelocity();
18  }
19
20  void PhysicalObject::destroyBody(){
21      this->body = NULL;
22      this->is_dead = true;
23  }
24
25  b2Vec2 PhysicalObject::getPosition(){
26      if (this->body){
27          return this->body->GetPosition();
28      }
29      return b2Vec2(-100, 0);
30  }
31
32  b2Body* PhysicalObject::getBody(){
33      return this->body;
34  }
35
36  bool PhysicalObject::isMoving(){
37      if (!this->body || this->is_dead){
38          return false;
39      }
40      b2Vec2 pos = this->body->GetPosition();
41      int last_x = this->last_position.x * UNIT_TO_SEND;
42      int last_y = this->last_position.y * UNIT_TO_SEND;
43      bool moved_x = (int)(pos.x * UNIT_TO_SEND) != last_x;
44      bool moved_y = (int)(pos.y * UNIT_TO_SEND) != last_y;
45      this->last_position = pos;
46      bool moved = moved_x || moved_y;
47      if (moved || this->data_updated){
48          this->last_position_sent = false;
49          this->data_updated = false;
50          return true;
51      }
52      if (!this->body->IsAwake() && !this->last_position_sent){
53          this->last_position_sent = true;
54          this->data_updated = false;
55          return true;
56      }
57      return false;
58  }
59
60  bool PhysicalObject::isActive(){
61      if (!this->body){
62          return false;
63      }
64      return this->body->IsAwake();
65  }
66

```

jun 12, 18 14:03

PhysicalObject.cpp

Page 2/2

```

67  bool PhysicalObject::isDead() {
68      return this->is_dead;
69  }
70
71  bool PhysicalObject::isWindAffected() {
72      return false;
73  }
74
75  void PhysicalObject::kill() {
76      this->is_dead = true;
77  }
78
79  int PhysicalObject::getId() {
80      return this->id;
81  }
82
83  const std::string& PhysicalObject::getType() {
84      return this->type;
85  }
86
87  void PhysicalObject::setInitialVelocity() {}
88
89  void PhysicalObject::collideWithSomething(CollisionData *other) {}

```

jun 12, 18 14:03

PhysicalObject.h

Page 1/2

```

1  #ifndef __PHYSICALOBJECT_H__
2  #define __PHYSICALOBJECT_H__
3
4  #include "b2Body.h"
5  #include "CollisionData.h"
6  #include "ObjectSizes.h"
7  #include "ObjectTypes.h"
8  #include <string>
9  #include <memory>
10
11  class World;
12
13  class PhysicalObject {
14  protected:
15      World& world;
16      b2Body* body;
17      bool is_dead;
18      int id;
19      const std::string& type;
20      b2Vec2 last_position;
21      bool last_position_sent;
22      bool data_updated;
23      CollisionData collision_data;
24
25      virtual void createFixtures() = 0;
26      virtual void setInitialVelocity();
27
28  public:
29      PhysicalObject(World& world, int id, const std::string& type);
30      virtual ~PhysicalObject();
31
32      //Inicializa el cuerpo del objeto
33      void initializeBody(b2Body* body);
34
35      //Destruye el cuerpo del objeto
36      void destroyBody();
37
38      //Devuelve la posicion del objeto
39      b2Vec2 getPosition();
40
41      //Devuelve el cuerpo del objeto
42      b2Body* getBody();
43
44      //Devuelve true si el objeto se esta moviendo
45      virtual bool isMoving();
46
47      //Devuelve true si el objeto esta activo
48      virtual bool isActive();
49
50      //Devuelve true si el objeto esta muerto
51      virtual bool isDead();
52
53      //Devuelve true si el objeto es afectado por el viento
54      virtual bool isWindAffected();
55
56      //Mata al objeto
57      void kill();
58
59      //Devuelve el id del objeto
60      int getId();
61
62      //Devuelve el tipo del objeto
63      const std::string& getType();
64
65      virtual void getBodyDef(b2BodyDef& body_def, const b2Vec2& pos) = 0;
66

```

jun 12, 18 14:03

PhysicalObject.h

Page 2/2

```

67         //Colisiona con otro objeto
68         virtual void collideWithSomething(CollisionData *other);
69     };
70
71     typedef std::shared_ptr<PhysicalObject> physical_object_ptr;
72
73 #endif

```

jun 12, 18 14:03

AirAttack.cpp

Page 1/1

```

1  #include "AirAttack.h"
2  #include "WeaponFactory.h"
3  #include "Worm.h"
4  #include <string>
5
6  AirAttack::AirAttack(World& world, GameParameters& parameters):
7      Weapon(world, parameters, 0),
8      missiles_separation(parameters.getAirMissilesSeparation()) {}
9
10 AirAttack::~AirAttack() {}
11
12 const std::string& AirAttack::getName() {
13     return AIR_ATTACK_NAME;
14 }
15
16 void AirAttack::shoot(char dir, int angle, int power, int time, int shooter){}
17
18 void AirAttack::shoot(Worm& shooter, b2Vec2 pos){
19     int missiles = this->parameters.getWeaponFragments(AIR_ATTACK_NAME);
20     float pos_x = pos.x - missiles * this->missiles_separation / 2;
21     float pos_y = this->parameters.getMaxHeight();
22     WeaponFactory factory(this->world, this->parameters);
23     for (int i = 0; i < missiles; i++, pos_x += this->missiles_separation){
24         physical_object_ptr missile = factory.getWeapon(AIR_ATTACK_MISSILE_NAME)
25     ;
26         this->world.addObject(missile, b2Vec2(pos_x, pos_y));
27     }

```


jun 12, 18 14:03

AirAttack.h

Page 1/1

```

1  #ifndef __SERVERAIRATTACK_H__
2  #define __SERVERAIRATTACK_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  class AirAttack: public Weapon{
8      private:
9          float missiles_separation;
10
11      public:
12          AirAttack(World& world, GameParameters& parameters);
13          ~AirAttack();
14
15          const std::string& getName() override;
16
17          void shoot(char dir, int angle, int power, int time, int shooter_id) override;
18
19          void shoot(Worm& shooter, b2Vec2 pos) override;
20 };
21
22 #endif

```

jun 12, 18 14:03

AirAttackMissile.cpp

Page 1/1

```

1  #include "AirAttackMissile.h"
2  #include <string>
3
4  AirAttackMissile::AirAttackMissile(World& world, GameParameters& parameters):
5      Weapon(world, parameters,
6          parameters.getWeaponDamage(AIR_ATTACK_MISSILE_NAME),
7          parameters.getWeaponRadius(AIR_ATTACK_MISSILE_NAME)) {}
8
9  AirAttackMissile::~AirAttackMissile() {}
10
11 const std::string& AirAttackMissile::getName() {
12     return AIR_ATTACK_MISSILE_NAME;
13 }
14
15 bool AirAttackMissile::isWindAffected() {
16     return true;
17 }

```

jun 12, 18 14:03

AirAttackMissile.h

Page 1/1

```

1  #ifndef __SERVERAIRATTACKMISSILE_H__
2  #define __SERVERAIRATTACKMISSILE_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  class AirAttackMissile: public Weapon{
8      public:
9          AirAttackMissile(World& world, GameParameters& parameters);
10         ~AirAttackMissile();
11
12         const std::string& getName() override;
13
14         bool isWindAffected() override;
15     };
16
17 #endif

```

jun 12, 18 14:03

Banana.cpp

Page 1/1

```

1  #include "Banana.h"
2  #include "b2Fixture.h"
3  #include "b2CircleShape.h"
4  #include <string>
5
6  Banana::Banana(World& world, GameParameters& parameters):
7      Weapon(world, parameters,
8          parameters.getWeaponDamage(BANANA_NAME),
9          parameters.getWeaponRadius(BANANA_NAME)) {}
10
11  Banana::~Banana() {}
12
13  const std::string& Banana::getName() {
14      return BANANA_NAME;
15  }
16
17  void Banana::createFixtures(){
18      b2CircleShape circleShape;
19      circleShape.m_p.Set(0, 0);
20      circleShape.m_radius = weapon_size / 2;
21
22      b2FixtureDef fixtureDef;
23      fixtureDef.shape = &circleShape;
24      fixtureDef.density = 4;
25      fixtureDef.restitution = 0.9; //rebotable
26      this->body->CreateFixture(&fixtureDef);
27  }

```

jun 12, 18 14:03

Banana.h

Page 1/1

```

1  #ifndef __SERVERBANANA_H__
2  #define __SERVERBANANA_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  class Banana: public Weapon{
8      protected:
9          void createFixtures() override;
10
11      public:
12          Banana(World& world, GameParameters& parameters);
13          ~Banana();
14
15          const std::string& getName() override;
16  };
17
18  #endif

```

jun 12, 18 14:03

Bat.cpp

Page 1/1

```

1  #include "Bat.h"
2  #include <string>
3
4  Bat::Bat(World& world, GameParameters& parameters):
5      Weapon(world, parameters, parameters.getWeaponDamage(BAT_NAME),
6              parameters.getWeaponRadius(BAT_NAME)) {}
7
8  Bat::~Bat() {}
9
10 const std::string& Bat::getName() {
11     return BAT_NAME;
12 }
13
14 void Bat::setInitialVelocity() {
15     this->explode();
16 }
17
18 void Bat::explode() {
19     b2Vec2 center = this->body->GetPosition();
20     this->attackWormExplosion(center, this->angle);
21
22     this->waiting_to_explode = false;
23     this->is_dead = true;
24 }

```

jun 12, 18 14:03

Bat.h

Page 1/1

```

1  #ifndef __SERVERBAT_H__
2  #define __SERVERBAT_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  class Bat: public Weapon{
8      public:
9          Bat(World& world, GameParameters& parameters);
10         ~Bat();
11
12         const std::string& getName() override;
13
14         void setInitialVelocity() override;
15
16         void explode() override;
17     };
18
19 #endif

```

jun 12, 18 14:03

Bazooka.cpp

Page 1/1

```

1  #include "Bazooka.h"
2  #include <string>
3
4  Bazooka::Bazooka(World& world, GameParameters& parameters):
5      Weapon(world, parameters,
6          parameters.getWeaponDamage(BAZOOKA_NAME),
7          parameters.getWeaponRadius(BAZOOKA_NAME)) {}
8
9  Bazooka::~Bazooka() {}
10
11  const std::string& Bazooka::getName() {
12      return BAZOOKA_NAME;
13  }
14
15  bool Bazooka::isWindAffected() {
16      return true;
17  }

```

jun 12, 18 14:03

Bazooka.h

Page 1/1

```
1  #ifndef __SERVERBAZOOKA_H__
2  #define __SERVERBAZOOKA_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  class Bazooka: public Weapon{
8      public:
9          Bazooka(World& world, GameParameters& parameters);
10         ~Bazooka();
11
12         const std::string& getName() override;
13
14         bool isWindAffected() override;
15     };
16
17 #endif
```

jun 12, 18 14:03

Dynamite.cpp

Page 1/1

```
1  #include "Dynamite.h"
2  #include <string>
3
4  Dynamite::Dynamite(World& world, GameParameters& parameters):
5      Weapon(world, parameters, parameters.getWeaponDamage(DYNAMITE_NAME),
6            parameters.getWeaponRadius(DYNAMITE_NAME)) {}
7
8  Dynamite::~Dynamite() {}
9
10 const std::string& Dynamite::getName() {
11     return DYNAMITE_NAME;
12 }
```

jun 12, 18 14:03

Dynamite.h

Page 1/1

```

1  #ifndef __SERVERDYNAMITE_H__
2  #define __SERVERDYNAMITE_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  class Dynamite: public Weapon{
8      public:
9          Dynamite(World& world, GameParameters& parameters);
10         ~Dynamite();
11
12         const std::string& getName() override;
13     };
14
15 #endif

```

jun 12, 18 14:03

FragmentableWeapon.cpp

Page 1/1

```

1  #include "FragmentableWeapon.h"
2  #include "WeaponFactory.h"
3  #include "Fragment.h"
4  #include "Math.h"
5  #include <string>
6
7  FragmentableWeapon::FragmentableWeapon(World& world, GameParameters& parameters,
8                                          int damage, int fragments, i
9                                          nt radius):
10      Weapon(world, parameters, damage, radius), fragments(fragments){}
11
12  FragmentableWeapon::~FragmentableWeapon(){}
13
14  void FragmentableWeapon::explode(){
15      WeaponFactory factory(this->world, this->parameters);
16      for (float angle = 0; angle < 360; angle+= (360 / fragments)){
17          physical_object_ptr fragment = factory.getWeapon(this->getName() + FRAGM
18          ENT);
19
20          b2Vec2 center = this->body->GetPosition() +
21                        0.3 * b2Vec2(Math::cosDegrees(angle), Math::sinDegrees(a
22          ngle));
23          ((Fragment *) fragment.get())->setShootPosition(center);
24          ((Fragment*) fragment.get())->shoot(angle);
25          this->world.addWeaponFragment(fragment);
26      }
27      Weapon::explode();
28  }

```

jun 12, 18 14:03

FragmentableWeapon.h

Page 1/1

```

1  #ifndef __FRAGMENTABLEWEAPON_H__
2  #define __FRAGMENTABLEWEAPON_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  class FragmentableWeapon: public Weapon{
8      protected:
9          int fragments;
10
11      public:
12          FragmentableWeapon(World& world, GameParameters& parameters,
13                               int damage, int fragments, int radius);
14          virtual ~FragmentableWeapon();
15
16          //Explota el arma y lanza fragmentos
17          void explode();
18      };
19
20 #endif

```

jun 12, 18 14:03

Fragment.cpp

Page 1/1

```

1  #include "Fragment.h"
2
3  Fragment::Fragment(World& world, GameParameters& params, int dam, int radius):
4      Weapon(world, params, dam, radius){}
5
6  Fragment::~~Fragment(){}
7
8  void Fragment::setShootPosition(b2Vec2 pos){
9      this->shoot_position = pos;
10 }
11
12 b2Vec2 Fragment::getShootPosition(){
13     return this->shoot_position;
14 }
15
16 void Fragment::shoot(int angle){
17     Weapon::shoot(1, angle, -1, -1, -1);
18 }

```

jun 12, 18 14:03

Fragment.h

Page 1/1

```

1  #ifndef __SERVERFRAGMENT_H__
2  #define __SERVERFRAGMENT_H__
3
4  #include "Weapon.h"
5
6  class Fragment: public Weapon{
7      private:
8          b2Vec2 shoot_position;
9
10     public:
11         Fragment(World& world, GameParameters& parameters, int damage, int radiu
12 s);
13         ~Fragment();
14
15         void setShootPosition(b2Vec2 pos);
16         b2Vec2 getShootPosition();
17
18         void shoot(int angle);
19     };
20 #endif

```

jun 12, 18 14:03

GreenGrenade.cpp

Page 1/1

```

1  #include "GreenGrenade.h"
2  #include <string>
3
4  GreenGrenade::GreenGrenade(World& world, GameParameters& parameters):
5      Weapon(world, parameters,
6          parameters.getWeaponDamage(GREEN_GRENADE_NAME),
7          parameters.getWeaponRadius(GREEN_GRENADE_NAME)) {}
8
9  GreenGrenade::~GreenGrenade() {}
10
11 const std::string& GreenGrenade::getName() {
12     return GREEN_GRENADE_NAME;
13 }

```


jun 12, 18 14:03

GreenGrenade.h

Page 1/1

```
1  #ifndef __SERVERGREENGRENADE_H__
2  #define __SERVERGREENGRENADE_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  class GreenGrenade: public Weapon{
8      public:
9          GreenGrenade(World& world, GameParameters& parameters);
10         ~GreenGrenade();
11
12         const std::string& getName() override;
13     };
14
15 #endif
```

jun 12, 18 14:03

HolyGrenade.cpp

Page 1/1

```
1  #include "HolyGrenade.h"
2  #include <string>
3
4  HolyGrenade::HolyGrenade(World& world, GameParameters& parameters):
5      Weapon(world, parameters,
6          parameters.getWeaponDamage(HOLY_GRENADE_NAME),
7          parameters.getWeaponRadius(HOLY_GRENADE_NAME)) {}
8
9  HolyGrenade::~HolyGrenade() {}
10
11 const std::string& HolyGrenade::getName() {
12     return HOLY_GRENADE_NAME;
13 }
```

jun 12, 18 14:03

HolyGrenade.h

Page 1/1

```

1  #ifndef __SERVERHOLYGRENADE_H__
2  #define __SERVERHOLYGRENADE_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  class HolyGrenade: public Weapon{
8      public:
9          HolyGrenade(World& world, GameParameters& parameters);
10         ~HolyGrenade();
11
12         const std::string& getName() override;
13     };
14
15 #endif

```

jun 12, 18 14:03

Mortar.cpp

Page 1/1

```

1  #include "Mortar.h"
2  #include <string>
3
4  Mortar::Mortar(World& world, GameParameters& parameters):
5      FragmentableWeapon(world, parameters,
6          parameters.getWeaponDamage(MORTAR_NAME),
7          parameters.getWeaponFragments(MORTAR_NAME),
8          parameters.getWeaponRadius(MORTAR_NAME)) {}
9
10 Mortar::~Mortar() {}
11
12 const std::string& Mortar::getName() {
13     return MORTAR_NAME;
14 }
15
16 bool Mortar::isWindAffected() {
17     return true;
18 }

```

jun 12, 18 14:03

MortarFragment.cpp

Page 1/1

```

1  #include "MortarFragment.h"
2  #include <string>
3
4  MortarFragment::MortarFragment(World& world, GameParameters& parameters):
5      Fragment(world, parameters,
6              parameters.getWeaponDamage(MORTAR_FRAGMENTS_NAME),
7              parameters.getWeaponRadius(MORTAR_FRAGMENTS_NAME)) {}
8
9  MortarFragment::~MortarFragment() {}
10
11  const std::string& MortarFragment::getName() {
12      return MORTAR_FRAGMENTS_NAME;
13  }
14
15  bool MortarFragment::isWindAffected() {
16      return true;
17  }

```

jun 12, 18 14:03

MortarFragment.h

Page 1/1

```

1  #ifndef __SERVERMORTARFRAGMENT_H__
2  #define __SERVERMORTARFRAGMENT_H__
3
4  #include "Fragment.h"
5  #include <string>
6
7  class MortarFragment: public Fragment{
8      public:
9          MortarFragment(World& world, GameParameters& parameters);
10         ~MortarFragment();
11
12         const std::string& getName() override;
13
14         bool isWindAffected() override;
15     };
16
17 #endif

```

jun 12, 18 14:03

Mortar.h

Page 1/1

```

1  #ifndef __SERVERMORTAR_H__
2  #define __SERVERMORTAR_H__
3
4  #include "FragmentableWeapon.h"
5  #include <string>
6
7  class Mortar: public FragmentableWeapon{
8      public:
9          Mortar(World& world, GameParameters& parameters);
10         ~Mortar();
11
12         const std::string& getName() override;
13
14         bool isWindAffected() override;
15     };
16
17 #endif

```

jun 12, 18 14:03

RedGrenade.cpp

Page 1/1

```

1  #include "RedGrenade.h"
2  #include <string>
3
4  RedGrenade::RedGrenade(World& world, GameParameters& parameters):
5      FragmentableWeapon(world, parameters,
6          parameters.getWeaponDamage(RED_GRENADE_NAME),
7          parameters.getWeaponFragments(RED_GRENADE_NAME),
8          parameters.getWeaponRadius(RED_GRENADE_NAME)) {}
9
10 RedGrenade::~RedGrenade() {}
11
12 const std::string& RedGrenade::getName() {
13     return RED_GRENADE_NAME;
14 }

```

jun 12, 18 14:03

RedGrenadeFragment.cpp

Page 1/1

```
1 #include "RedGrenadeFragment.h"
2 #include <string>
3
4 RedGrenadeFragment::RedGrenadeFragment(World& world, GameParameters& params):
5     Fragment(world, params,
6         params.getWeaponDamage(RED_GRENADE_FRAGMENTS_NAME),
7         params.getWeaponRadius(RED_GRENADE_FRAGMENTS_NAME)) {}
8
9 RedGrenadeFragment::~RedGrenadeFragment() {}
10
11 const std::string& RedGrenadeFragment::getName() {
12     return RED_GRENADE_FRAGMENTS_NAME;
13 }
```

jun 12, 18 14:03

RedGrenadeFragment.h

Page 1/1

```
1 #ifndef __SERVERREDGRENADEFRAGMENT_H__
2 #define __SERVERREDGRENADEFRAGMENT_H__
3
4 #include "Fragment.h"
5 #include <string>
6
7 class RedGrenadeFragment: public Fragment{
8     public:
9         RedGrenadeFragment(World& world, GameParameters& parameters);
10         ~RedGrenadeFragment();
11
12         const std::string& getName() override;
13 };
14
15 #endif
```

jun 12, 18 14:03

RedGrenade.h

Page 1/1

```

1  #ifndef __SERVERREDGRENADE_H__
2  #define __SERVERREDGRENADE_H__
3
4  #include "FragmentableWeapon.h"
5  #include <string>
6
7  class RedGrenade: public FragmentableWeapon{
8      public:
9          RedGrenade(World& world, GameParameters& parameters);
10         ~RedGrenade();
11
12         const std::string& getName() override;
13     };
14
15 #endif

```

jun 12, 18 14:03

Teleportation.cpp

Page 1/1

```

1  #include "Teleportation.h"
2  #include "Worm.h"
3  #include <mutex>
4  #include <string>
5
6  Teleportation::Teleportation(World& world, GameParameters& parameters):
7      Weapon(world, parameters, 0){}
8
9  Teleportation::~Teleportation(){}
10
11 const std::string& Teleportation::getName(){
12     return TELEPORT_NAME;
13 }
14
15 void Teleportation::shoot(char dir, int angle, int power,
16                           int time, int shooter_id){}
17
18 void Teleportation::shoot(Worm& shooter, b2Vec2 pos){
19     pos.x += (worm_size / 2);
20     pos.y += (worm_size / 2);
21     std::lock_guard<std::mutex> lock(this->world.getMutex());
22     b2Body* body = shooter.getBody();
23     if (body){
24         shooter.getBody()->SetTransform(pos, 0);
25         shooter.getBody()->SetAwake(true);
26     }
27 }

```

jun 12, 18 14:03

Teleportation.h

Page 1/1

```

1  #ifndef __SERVERTELEPORTATION_H__
2  #define __SERVERTELEPORTATION_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  class Teleportation: public Weapon{
8      public:
9          Teleportation(World& world, GameParameters& parameters);
10         ~Teleportation();
11
12         const std::string& getName() override;
13
14         void shoot(char dir, int angle, int power, int time, int shooter_id) over
15         rride;
16
17         //Teletransporta al gusano
18         void shoot(Worm& shooter, b2Vec2 pos) override;
19
20 #endif

```

jun 12, 18 14:03

Weapon.cpp

Page 1/2

```

1  #include "Weapon.h"
2  #include "b2Fixture.h"
3  #include "b2CircleShape.h"
4  #include "CollisionData.h"
5  #include "Worm.h"
6  #include "Math.h"
7
8  int Weapon::weapon_id = 1;
9
10 Weapon::Weapon(World& world, GameParameters& params, int damage, int radius):
11     PhysicalObject(world, Weapon::weapon_id++, TYPE_WEAPON),
12     parameters(params),
13     damage(damage), radius(radius),
14     waiting_to_explode(false), time_to_explode(-1),
15     angle(MAX_WEAPON_ANGLE + 1), power(-1),
16     shooter_id(-1), explode_time(world, *this){}
17
18 Weapon::~Weapon(){
19     this->explode_time.join();
20 }
21
22 bool Weapon::isActive(){
23     return this->waiting_to_explode || PhysicalObject::isActive();
24 }
25
26 void Weapon::shoot(char dir, int angle, int power, int time, int shooter_id){
27     if (dir == -1 && angle <= MAX_WEAPON_ANGLE){
28         angle = 180 - angle;
29     }
30     this->time_to_explode = time;
31     this->angle = angle;
32     this->power = power;
33     this->shooter_id = shooter_id;
34 }
35
36 void Weapon::shoot(Worm& shooter, b2Vec2 pos){}
37
38 void Weapon::getBodyDef(b2BodyDef& body_def, const b2Vec2& pos){
39     body_def.type = b2_dynamicBody;
40     body_def.position.Set(pos.x, pos.y);
41     body_def.fixedRotation = true;
42     body_def.bullet = true;
43 }
44
45 void Weapon::createFixtures(){
46     b2CircleShape circleShape;
47     circleShape.m_p.Set(0, 0);
48     circleShape.m_radius = weapon_size / 2;
49
50     b2FixtureDef fixtureDef;
51     fixtureDef.shape = &circleShape;
52     fixtureDef.density = 4;
53     this->body->CreateFixture(&fixtureDef);
54 }
55
56 void Weapon::setInitialVelocity(){
57     if (this->angle <= 360){
58         int velocity = this->parameters.getWeaponsVelocity();
59         if (this->power != -1){
60             velocity *= this->power / 1000;
61         }
62         b2Vec2 linear_velocity(velocity * Math::cosDegrees(this->angle),
63                                velocity * Math::sinDegrees(this->angle));
64         this->body->SetLinearVelocity(linear_velocity);
65     }
66     this->waiting_to_explode = true;

```

jun 12, 18 14:03

Weapon.cpp

Page 2/2

```

67     this->explode_time.setTime(this->time_to_explode);
68     this->explode_time.start();
69 }
70
71
72 void Weapon::explode(){
73     b2Vec2 center = this->body->GetPosition();
74     for (float bullet_angle = 0; bullet_angle < 360; bullet_angle+= 5){
75         this->attackWormExplosion(center, bullet_angle);
76     }
77
78     this->explode_time.stop();
79     this->waiting_to_explode = false;
80     this->is_dead = true;
81 }
82
83 void Weapon::attackWormExplosion(const b2Vec2& center, int angle){
84     b2Vec2 end = center + this->radius *
85         b2Vec2(Math::cosDegrees(angle), Math::sinDegrees(ang
86 le));
87     b2Body* closest = this->world.getClosestObject(&explosion, center, end);
88     if (closest){
89         Worm* worm = (Worm*)((CollisionData*)closest->GetUserData())->getObject(
90 );
91         float distance = b2Distance(center, worm->getPosition());
92         //Justo en el borde hace la mitad de danio
93         int worm_damage = this->damage * (1 - distance / (2 * this->radius));
94
95         worm->receiveWeaponDamage(worm_damage, center);
96     }
97 }
98
99 void Weapon::collideWithSomething(CollisionData *other){
100     if (this->time_to_explode == -1){
101         this->explode_time.stop();
102         this->explode();
103     } else if (other->getType() == TYPE_BORDER){
104         this->explode_time.stop();
105         this->is_dead = true;
106     }
107 }
108
109 int Weapon::getShooterId() const{
110     return this->shooter_id;
111 }
112
113 void Weapon::removeShooterId(){
114     this->shooter_id = -1;
115 }

```

jun 12, 18 14:03

WeaponExplodeTime.cpp

Page 1/1

```

1  #include "WeaponExplodeTime.h"
2  #include "Weapon.h"
3  #include "World.h"
4
5  WeaponExplodeTime::WeaponExplodeTime(World& world, Weapon& weapon):
6      weapon(weapon), world(world), time(-1){}
7
8  WeaponExplodeTime::~WeaponExplodeTime(){}
9
10 void WeaponExplodeTime::setTime(int time){
11     this->time = time;
12 }
13
14 void WeaponExplodeTime::run(){
15     if (this->time > 0){
16         int passed = 0;
17         while (this->running && passed < this->time){
18             std::this_thread::sleep_for(std::chrono::seconds(1));
19             passed++;
20         }
21         if (this->running){
22             std::lock_guard<std::mutex> lock(this->world.getMutex());
23             if (!this->weapon.isDead()){
24                 this->weapon.explode();
25                 this->world.removeTimedWeapon(this->weapon);
26             }
27         }
28     }
29 }

```


jun 12, 18 14:03

WeaponExplodeTime.h

Page 1/1

```

1  #ifndef __WEAPONEXPLODETIME_H__
2  #define __WEAPONEXPLODETIME_H__
3
4  #include "Thread.h"
5  #include <mutex>
6
7  class Weapon;
8  class World;
9
10 class WeaponExplodeTime: public Thread{
11     private:
12         Weapon& weapon;
13         World& world;
14         int time;
15
16     public:
17         WeaponExplodeTime(World& world, Weapon& weapon);
18         ~WeaponExplodeTime();
19
20         //Setea el tiempo de explosion
21         void setTime(int time);
22
23         //Cuenta el tiempo que falta para que el arma explote
24         void run() override;
25 };
26
27 #endif

```

jun 12, 18 14:03

WeaponFactory.cpp

Page 1/1

```

1  #include "WeaponFactory.h"
2  #include "WeaponNames.h"
3  #include <string>
4
5  #include "Bazooka.h"
6  #include "Dynamite.h"
7  #include "RedGrenade.h"
8  #include "RedGrenadeFragment.h"
9  #include "GreenGrenade.h"
10 #include "HolyGrenade.h"
11 #include "Banana.h"
12 #include "Teleportation.h"
13 #include "AirAttack.h"
14 #include "AirAttackMissile.h"
15 #include "Mortar.h"
16 #include "MortarFragment.h"
17 #include "Bat.h"
18
19 WeaponFactory::WeaponFactory(World& world, GameParameters& parameters):
20     world(world), parameters(parameters){}
21
22 WeaponFactory::~WeaponFactory(){}
23
24 physical_object_ptr WeaponFactory::getWeapon(const std::string& name){
25     if (name == BAZOOKA_NAME){
26         return physical_object_ptr(new Bazooka(world, parameters));
27     } else if (name == DYNAMITE_NAME){
28         return physical_object_ptr(new Dynamite(world, parameters));
29     } else if (name == RED_GRENADE_NAME){
30         return physical_object_ptr(new RedGrenade(world, parameters));
31     } else if (name == RED_GRENADE_FRAGMENTS_NAME){
32         return physical_object_ptr(new RedGrenadeFragment(world, parameters));
33     } else if (name == GREEN_GRENADE_NAME){
34         return physical_object_ptr(new GreenGrenade(world, parameters));
35     } else if (name == HOLY_GRENADE_NAME){
36         return physical_object_ptr(new HolyGrenade(world, parameters));
37     } else if (name == MORTAR_NAME){
38         return physical_object_ptr(new Mortar(world, parameters));
39     } else if (name == MORTAR_FRAGMENTS_NAME){
40         return physical_object_ptr(new MortarFragment(world, parameters));
41     } else if (name == BANANA_NAME){
42         return physical_object_ptr(new Banana(world, parameters));
43     } else if (name == BAT_NAME){
44         return physical_object_ptr(new Bat(world, parameters));
45     } else if (name == TELEPORT_NAME){
46         return physical_object_ptr(new Teleportation(world, parameters));
47     } else if (name == AIR_ATTACK_NAME){
48         return physical_object_ptr(new AirAttack(world, parameters));
49     } else if (name == AIR_ATTACK_MISSILE_NAME){
50         return physical_object_ptr(new AirAttackMissile(world, parameters));
51     }
52
53     throw std::runtime_error(name + ": El arma no existe.");
54 }

```

jun 12, 18 14:03

WeaponFactory.h

Page 1/1

```

1  #ifndef __WEAPONFACTORY_H__
2  #define __WEAPONFACTORY_H__
3
4  #include "World.h"
5  #include "GameParameters.h"
6  #include <string>
7
8  class WeaponFactory{
9      private:
10         World& world;
11         GameParameters& parameters;
12
13     public:
14         WeaponFactory(World& world, GameParameters& parameters);
15         ~WeaponFactory();
16
17         //Devuelve el arma pedida
18         physical_object_ptr getWeapon(const std::string& name);
19     };
20
21 #endif

```

jun 12, 18 14:03

Weapon.h

Page 1/2

```

1  #ifndef __WEAPON_H__
2  #define __WEAPON_H__
3
4  #include "PhysicalObject.h"
5  #include "GameParameters.h"
6  #include "World.h"
7  #include "WeaponExplodeTime.h"
8  #include <string>
9  #include "WeaponNames.h"
10 #include "RayCastWeaponExploded.h"
11
12 class Worm;
13
14 class Weapon: public PhysicalObject{
15     protected:
16         GameParameters& parameters;
17         int damage;
18         int radius;
19         bool waiting_to_explode;
20         int time_to_explode;
21         float angle;
22         float power;
23         int shooter_id;
24         WeaponExplodeTime explode_time;
25         RayCastWeaponExploded explosion;
26
27         virtual void createFixtures() override;
28         virtual void setInitialVelocity() override;
29
30         //Ataca a los gusanos en el radio de explosion
31         void attackWormExplosion(const b2Vec2& center, int angle);
32
33     public:
34         static int weapon_id;
35
36         Weapon(World& world, GameParameters& parameters, int damage, int radius
37 = 0);
38         virtual ~Weapon();
39
40         //Devuelve true si el arma esta en movimiento o esperando para explotar
41         bool isActive() override;
42
43         //Carga los datos para disparar el arma
44         virtual void shoot(char dir, int angle, int power, int time, int shooter
45 _id);
46
47         //Dispara un arma teledirigida
48         virtual void shoot(Worm& shooter, b2Vec2 pos);
49
50         //Explota el arma
51         virtual void explode();
52
53         //Establece la accion a realizar cuando el arma colisiona
54         virtual void collideWithSomething(CollisionData *other) override;
55
56         void getBodyDef(b2BodyDef& body_def, const b2Vec2& pos) override;
57
58         //Devuelve el nombre del arma
59         virtual const std::string& getName() = 0;
60
61         //Devuelve el id del tirador
62         int getShooterId() const;
63
64         //Remueve el id del tirador
65         void removeShooterId();
66     };

```

jun 12, 18 14:03

Weapon.h

Page 2/2

```

65
66 #endif

```

jun 12, 18 14:03

Worm.cpp

Page 1/4

```

1  #include "Worm.h"
2  #include "b2CircleShape.h"
3  #include "b2PolygonShape.h"
4  #include "b2Fixture.h"
5  #include "Protocol.h"
6  #include "WeaponFactory.h"
7  #include "Girder.h"
8  #include "Math.h"
9  #include <algorithm>
10 #include <string>
11
12 Worm::Worm(World& world, GameParameters& parameters,
13           int id, int player_id, WeaponList& weapons):
14     PhysicalObject(world, id, TYPE_WORM), player_id(player_id),
15     life(parameters.getWormLife()),
16     dir(1), parameters(parameters), weapons(weapons), max_height(0),
17     colliding_with_girder(0), friction(0),
18     movement_allowed(false), angle(0), has_shot(false), damage_received(false){}
19
20 Worm::~Worm() {}
21
22 void Worm::getBodyDef(b2BodyDef& body_def, const b2Vec2& pos){
23     body_def.type = b2_dynamicBody;
24     body_def.position.Set(pos.x, pos.y);
25 }
26
27 void Worm::createFixtures(){
28     b2CircleShape circleShape;
29     circleShape.m_p.Set(0, 0);
30     circleShape.m_radius = worm_size / 2;
31
32     b2FixtureDef fixtureDef;
33     fixtureDef.shape = &circleShape;
34     fixtureDef.density = 10;
35     this->body->CreateFixture(&fixtureDef);
36     this->body->SetFixedRotation(true);
37
38     //Sensor para colisiones
39     b2PolygonShape sensorShape;
40     sensorShape.SetAsBox(worm_size * 0.5 * 0.7, worm_size / 5,
41                          b2Vec2(0, -1 * worm_size / 2), 0);
42
43     b2FixtureDef sensorFixtureDef;
44     sensorFixtureDef.shape = &sensorShape;
45     sensorFixtureDef.isSensor = true;
46     this->body->CreateFixture(&sensorFixtureDef);
47 }
48
49 int Worm::getPlayerId() const{
50     return this->player_id;
51 }
52
53 int Worm::getLife() const{
54     return this->life;
55 }
56
57 char Worm::getDir() const{
58     return this->dir;
59 }
60
61 bool Worm::isColliding() const{
62     return this->colliding_with_girder && !this->movement_allowed;
63 }
64
65 const std::string& Worm::getCurrentWeapon() const{
66     physical_object_ptr weapon = weapons.getCurrentWeapon(world, parameters);

```

jun 12, 18 14:03

Worm.cpp

Page 2/4

```

67     return ((Weapon*)weapon.get())->getName();
68 }
69
70 void Worm::addLife(int life){
71     this->life += life;
72 }
73
74 void Worm::reduceLife(size_t damage){
75     this->life -= damage;
76     this->damage_received = true;
77     this->data_updated = true;
78     if (this->life <= 0){
79         this->life = 0;
80         this->is_dead = true;
81     }
82 }
83
84 bool Worm::move(char action){
85     if (!this->colliding_with_girder || this->movement_allowed){
86         return false;
87     }
88     this->movement_allowed = false;
89     if (action == MOVE_RIGHT){
90         this->dir = action;
91         b2Vec2 velocity(parameters.getWormVelocity(), 0);
92         this->world.setLinearVelocity(*this, velocity);
93     } else if (action == MOVE_LEFT){
94         this->dir = action;
95         b2Vec2 velocity(-1 * parameters.getWormVelocity(), 0);
96         this->world.setLinearVelocity(*this, velocity);
97     } else {
98         this->movement_allowed = true;
99         if (action == JUMP){
100             b2Vec2 velocity(parameters.getWormJumpVelocity(),
101                             parameters.getWormJumpHeight());
102             velocity.x *= this->dir;
103             this->world.setLinearVelocity(*this, velocity);
104         } else if (action == ROLLBACK){
105             b2Vec2 velocity(parameters.getWormRollbackVelocity(),
106                             parameters.getWormRollbackHeight());
107             velocity.x *= -1 * this->dir;
108             this->world.setLinearVelocity(*this, velocity);
109         }
110     }
111     return true;
112 }
113
114 void Worm::shoot(int angle, int power, int time){
115     if (!this->weapons.shoot()){
116         return;
117     }
118     b2Vec2 pos = this->getPosition();
119     int shooter_id = this->id;
120     float x_add = (worm_size * this->dir);
121     float y_add = worm_size;
122     if (angle > MAX_WEAPON_ANGLE){
123         shooter_id = -1;
124         x_add *= Math::cosDegrees(this->angle);
125         y_add *= Math::sinDegrees(this->angle);
126     } else {
127         float factor = (this->getCurrentWeapon() == BAT_NAME ? 0.2 : 0.7);
128         x_add *= Math::cosDegrees(angle) * factor;
129         y_add *= Math::sinDegrees(angle) * factor;
130     }
131 }
132 pos.x += x_add;

```

jun 12, 18 14:03

Worm.cpp

Page 3/4

```

133     pos.y += y_add;
134
135     physical_object_ptr weapon = weapons.getCurrentWeapon(world, parameters);
136     ((Weapon*)weapon.get())->shoot(this->dir, angle, power, time, shooter_id);
137     this->world.addObject(weapon, pos);
138     this->has_shot = true;
139 }
140
141 void Worm::shoot(b2Vec2 pos){
142     if (!this->weapons.shoot()){
143         return;
144     }
145     physical_object_ptr weapon = weapons.getCurrentWeapon(world, parameters);
146     ((Weapon*)weapon.get())->shoot(*this, pos);
147     this->has_shot = true;
148 }
149
150 void Worm::receiveWeaponDamage(int damage, const b2Vec2 &epicenter){
151     this->reduceLife(damage);
152     b2Vec2 direction = this->body->GetPosition() - epicenter;
153     direction.Normalize();
154     this->body->SetGravityScale(1);
155     this->movement_allowed = true;
156     this->body->SetLinearVelocity(
157         damage * parameters.getWormExplosionVelocity() * direction);
158 }
159
160 void Worm::collideWithSomething(CollisionData *other){
161     if (other->getType() == TYPE_BORDER){
162         this->kill();
163     } else if (other->getType() == TYPE_GIRDER){
164         int min_height = parameters.getWormHeightToDamage();
165         float current_height = this->body->GetPosition().y;
166         this->max_height -= current_height;
167
168         if (this->max_height >= min_height){
169             this->reduceLife(std::min((int) this->max_height - min_height + 1,
170                                     parameters.getWormMaxHeightDamage()));
171         }
172         this->max_height = 0;
173         this->colliding_with_girder++;
174         Girder* girder = (Girder*)other->getObject();
175         if (girder->hasFriction()){
176             this->friction++;
177             this->movement_allowed = false;
178             this->angle = girder->getAngle();
179         }
180     }
181 }
182
183 void Worm::endCollissionGirder(char has_friction){
184     this->friction -= has_friction;
185     this->colliding_with_girder--;
186     if (this->friction <= 0){
187         this->friction = 0;
188         this->body->SetGravityScale(1);
189         this->angle = 0;
190     }
191 }
192
193 bool Worm::isActive(){
194     if (!this->colliding_with_girder){
195         float height = this->body->GetPosition().y;
196         this->max_height = std::max(this->max_height, height);
197     } else if (this->friction && !this->movement_allowed){

```

jun 12, 18 14:03

Worm.cpp

Page 4/4

```

198         this->body->SetGravityScale(0);
199         this->body->SetLinearVelocity(b2Vec2(0, 0));
200     }
201     if (!this->body->IsAwake()){
202         this->movement_allowed = false;
203     } else if (!this->friction){
204         this->movement_allowed = true;
205     }
206     return PhysicalObject::isActive();
207 }
208
209 bool Worm::hasShot() const{
210     return this->has_shot;
211 }
212
213 bool Worm::damageReceived() const{
214     return this->damage_received || this->is_dead;
215 }
216
217 void Worm::beginTurn(){
218     this->has_shot = false;
219     this->damage_received = false;
220 }

```

jun 12, 18 14:03

Worm.h

Page 1/2

```

1  #ifndef __WORM_H__
2  #define __WORM_H__
3
4  #include "PhysicalObject.h"
5  #include "GameParameters.h"
6  #include "Weapon.h"
7  #include "WeaponList.h"
8  #include <string>
9
10 class Worm: public PhysicalObject{
11     private:
12         int player_id;
13         int life;
14         char dir;
15         GameParameters& parameters;
16         WeaponList& weapons;
17         float max_height;
18         int colliding_with_girder;
19         int friction;
20         bool movement_allowed;
21         int angle;
22
23         bool has_shot;
24         bool damage_received;
25
26     protected:
27         void getBodyDef(b2BodyDef& body_def, const b2Vec2& pos) override;
28         void createFixtures() override;
29
30     public:
31         Worm(World& world, GameParameters& parameters,
32             int id, int player_id, WeaponList& weapons);
33         ~Worm();
34
35         //Devuelve el id del jugador
36         int getPlayerId() const;
37
38         //Devuelve la vida del worm
39         int getLife() const;
40
41         //Devuelve la direccion del worm
42         char getDir() const;
43
44         //Devuelve true si esta colisionando
45         bool isColliding() const;
46
47         //Devuelve el arma actual
48         const std::string& getCurrentWeapon() const;
49
50         //Aumenta la vida del gusano
51         void addLife(int life);
52
53         //Reduce la vida del gusano
54         void reduceLife(size_t damage);
55
56         //Ejecuta una accion de movimiento del gusano
57         bool move(char action);
58
59         //Dispara un arma no teledirigida
60         void shoot(int angle, int power, int time);
61
62         //Dispara un arma teledirigida
63         void shoot(b2Vec2 pos);
64
65         //Analiza la colision con el objeto
66         void collideWithSomething(CollisionData *other) override;

```

jun 12, 18 14:03

Worm.h

Page 2/2

```

67
68 //Analiza el fin del contacto con una viga
69 void endCollissionGirder(char friction);
70
71 //Recibe danio de un arma o una explosion
72 void receiveWeaponDamage(int damage, const b2Vec2 &epicenter);
73
74 //Devuelve true si el gusano esta en movimiento
75 bool isActive() override;
76
77 //Devuelve true si el gusano disparo
78 bool hasShot() const;
79
80 //Devuelve true si el gusano recibio danio
81 bool damageReceived() const;
82
83 //Empieza el turno del gusano
84 void beginTurn();
85 };
86
87 #endif

```

jun 12, 18 14:03

Wind.cpp

Page 1/1

```

1  #include "Wind.h"
2  #include <random>
3
4  Wind::Wind(GameParameters& parameters):
5      min_velocity(parameters.getWindMinVelocity()),
6      max_velocity(parameters.getWindMaxVelocity()){
7      this->update();
8  }
9
10 Wind::~Wind(){}
11
12 float Wind::getVelocity() const{
13     return this->velocity;
14 }
15
16 void Wind::update(){
17     std::mt19937 rng;
18     rng.seed(std::random_device()());
19     std::uniform_real_distribution<float> dist(min_velocity, max_velocity);
20     std::uniform_int_distribution<int> direction(-1, 1); //Acepto velocidad 0
21
22     this->velocity = dist(rng);
23     this->velocity *= direction(rng);
24 }

```

jun 12, 18 14:03

Wind.h

Page 1/1

```

1  #ifndef __WIND_H__
2  #define __WIND_H__
3
4  #include "GameParameters.h"
5
6  class Wind{
7      private:
8          float min_velocity;
9          float max_velocity;
10         float velocity;
11
12     public:
13         explicit Wind(GameParameters& parameters);
14         ~Wind();
15
16         //Devuelve la velocidad del viento
17         float getVelocity() const;
18
19         //Actualiza la velocidad del viento
20         void update();
21 };
22
23
24 #endif

```

jun 25, 18 19:28

World.cpp

Page 1/3

```

1  #include "World.h"
2  #include "Weapon.h"
3  #include "BottomBorder.h"
4  #include "b2WorldCallbacks.h"
5  #include "Fragment.h"
6  #include <list>
7
8  World::World(GameParameters& parameters):
9      world(b2Vec2(0, parameters.getGravity())),
10      wind(parameters), is_active(true),
11      sleep_time(parameters.getWorldSleepAfterStep()),
12      time_step(parameters.getWorldTimeStep()){
13      this->world.SetAllowSleeping(true);
14      this->world.SetContinuousPhysics(true);
15      this->world.SetContactListener(&this->collision_listener);
16      this->world.SetContactFilter(&this->collision_listener);
17      this->initialize();
18  }
19
20  World::~World(){}
21
22  void World::run(){
23      int32 velocityIterations = 8;    //how strongly to correct velocity
24      int32 positionIterations = 3;    //how strongly to correct position
25
26      while(this->running){
27          std::this_thread::sleep_for(std::chrono::milliseconds(this->sleep_time))
28          ;
29
30          this->addAllFragments();
31
32          std::lock_guard<std::mutex> lock(this->mutex);
33
34          this->world.Step(this->time_step, velocityIterations, positionIterations
35          );
36
37          this->is_active = false;
38          for (physical_object_ptr& object: this->objects){
39              if (object->isDead()){
40                  this->removeObject(object);
41              } else if (object->isActive()){
42                  this->is_active = true;
43                  b2Body* body = object->getBody();
44                  if (body && object->isWindAffected()){
45                      body->ApplyForceToCenter(b2Vec2(this->wind.getVelocity(), 0)
46                      , false);
47                  }
48              }
49          }
50
51          void World::addAllFragments(){
52              std::lock_guard<std::mutex> lock(this->mutex);
53              for (physical_object_ptr& fragment: this->fragments_to_add){
54                  b2BodyDef body_def;
55                  b2Vec2 pos = ((Fragment *) fragment.get())->getShootPosition();
56                  fragment->getBodyDef(body_def, pos);
57                  this->initializeObject(fragment, &body_def);
58              }
59              this->fragments_to_add.clear();
60          }
61
62          bool World::isActive(){
63              std::lock_guard<std::mutex> lock(this->mutex);
64              return this->is_active;
65          }

```

jun 25, 18 19:28

World.cpp

Page 2/3

```

64 }
65
66 void World::update(){
67     std::lock_guard<std::mutex> lock(this->mutex);
68     this->wind.update();
69 }
70
71 void World::addObject(physical_object_ptr object, const b2Vec2& pos){
72     b2BodyDef body_def;
73     object->getBodyDef(body_def, pos);
74
75     std::lock_guard<std::mutex> lock(this->mutex);
76     this->initializeObject(object, &body_def);
77 }
78
79 void World::initializeObject(physical_object_ptr object, b2BodyDef* body_def){
80     object->initializeBody(this->world.CreateBody(body_def));
81     if (body_def->type != b2_staticBody){
82         this->objects.push_back(object);
83     } else {
84         this->girders.push_back(object);
85     }
86 }
87
88 void World::addWeaponFragment(physical_object_ptr fragment){
89     this->fragments_to_add.push_back(fragment);
90 }
91
92 void World::removeTimedWeapon(Weapon& weapon){
93     b2Body* body = weapon.getBody();
94     if (body){
95         this->world.DestroyBody(body);
96         weapon.destroyBody();
97     }
98 }
99
100 void World::removeObject(physical_object_ptr object){
101     b2Body* body = object->getBody();
102     if (body){
103         this->world.DestroyBody(body);
104         object->destroyBody();
105     }
106 }
107
108 void World::initialize(){
109     physical_object_ptr bottom_border(new BottomBorder(*this));
110     this->addObject(bottom_border, b2Vec2(0, 0));
111 }
112
113 void World::setLinearVelocity(PhysicalObject& object, b2Vec2& velocity){
114     std::lock_guard<std::mutex> lock(this->mutex);
115     b2Body* body = object.getBody();
116     if (body){
117         body->SetGravityScale(1);
118         body->SetLinearVelocity(velocity);
119     }
120 }
121
122 b2Body* World::getClosestObject(RayCastWeaponExploded* callback,
123                                 b2Vec2 center, b2Vec2 end){
124     this->world.RayCast(callback, center, end);
125     return callback->getClosestWorm();
126 }
127
128 float World::getWind() const{
129     return this->wind.getVelocity();

```

jun 25, 18 19:28

World.cpp

Page 3/3

```

130 }
131
132 std::list<physical_object_ptr>& World::getObjectsList(){
133     return this->objects;
134 }
135
136 std::list<physical_object_ptr>& World::getGirdersList(){
137     return this->girders;
138 }
139
140 std::mutex& World::getMutex(){
141     return this->mutex;
142 }

```


jun 12, 18 14:03	World.h	Page 1/2
<pre> 1 #ifndef __WORLD_H__ 2 #define __WORLD_H__ 3 4 #include "Thread.h" 5 #include "b2World.h" 6 #include "b2Body.h" 7 #include "PhysicalObject.h" 8 #include "CollisionListener.h" 9 #include "RayCastWeaponExploded.h" 10 #include "Wind.h" 11 #include <mutex> 12 #include <list> 13 14 class Weapon; 15 16 class World: public Thread{ 17 private: 18 b2World world; 19 Wind wind; 20 std::mutex mutex; 21 CollisionListener collision_listener; 22 std::list<physical_object_ptr> objects; 23 std::list<physical_object_ptr> girders; 24 std::list<physical_object_ptr> fragments_to_add; 25 bool is_active; 26 int sleep_time; 27 float time_step; 28 29 //Inicializa el mundo 30 void initialize(); 31 32 //Remueve un objeto del mundo 33 void removeObject(physical_object_ptr object); 34 35 //Inicializa un objeto recién agregado al mundo 36 void initializeObject(physical_object_ptr object, b2BodyDef* body_def); 37 38 //Agrega todos los fragmentos de armas al mundo 39 void addAllFragments(); 40 41 public: 42 explicit World(GameParameters& parameters); 43 ~World(); 44 45 void run() override; 46 47 //Agrega el objeto al mundo en la posición indicada 48 void addObject(physical_object_ptr object, const b2Vec2& pos); 49 50 //Agrega un fragmento de arma 51 void addWeaponFragment(physical_object_ptr fragment); 52 53 //Elimina una arma del mundo 54 void removeTimedWeapon(Weapon& weapon); 55 56 //Setea la velocidad de un objeto 57 void setLinearVelocity(PhysicalObject& object, b2Vec2& velocity); 58 59 //Devuelve true si alguno de los objetos está en movimiento 60 bool isActive(); 61 62 //Actualiza el mundo 63 void update(); 64 65 //Devuelve la velocidad del viento 66 float getWind() const; </pre>		

jun 12, 18 14:03	World.h	Page 2/2
<pre> 67 68 //Devuelve el objeto más cercano entre al centro en la dirección end - c 69 enter 70 b2Body* getClosestObject(RayCastWeaponExploded* callback, 71 b2Vec2 center, b2Vec2 end); 72 73 //Devuelve la lista de objetos 74 std::list<physical_object_ptr>& getObjectsList(); 75 76 //Devuelve la lista de vigas 77 std::list<physical_object_ptr>& getGirdersList(); 78 79 //Devuelve el mutex 80 std::mutex& getMutex(); 81 }; 82 83 #endif </pre>		

jun 12, 18 14:03

main.cpp

Page 1/1

```

1  #include "Server.h"
2  #include "yaml.h"
3  #include "ConfigFields.h"
4  #include "Path.h"
5  #include <iostream>
6  #include <string>
7  #include <mutex>
8
9  #define EXIT_CHAR 'q'
10
11 int main(int argc, const char* argv[]){
12     std::mutex mutex_cout;
13     try{
14         YAML::Node config(YAML::LoadFile(SERVER_CONFIG_FILE));
15         Server server(config[SERVER_PORT].as<std::string>(), mutex_cout);
16         std::cout << "[LOG] Server iniciado." << std::endl;
17         server.start();
18         while (std::cin.get() != EXIT_CHAR){
19             {
20                 std::lock_guard<std::mutex> lock(mutex_cout);
21                 std::cout << "[LOG] Comenzando el cierre del servidor." << std::endl;
22             }
23             server.stop();
24             server.join();
25         } catch(const std::exception& e){
26             std::lock_guard<std::mutex> lock(mutex_cout);
27             std::cout << "[ERROR]" << e.what() << std::endl;
28         }
29         return 0;
30     }

```

jun 26, 18 13:59

Table of Content

Page 1/2

Table of Contents				
1	1	ClientHandler.cpp...	sheets 1 to 1	(1) pages 1- 2 97 lines
2	2	ClientHandler.h.....	sheets 2 to 2	(1) pages 3- 3 39 lines
3	3	GamesList.cpp.....	sheets 2 to 3	(2) pages 4- 5 94 lines
4	4	GamesList.h.....	sheets 3 to 3	(1) pages 6- 6 46 lines
5	5	MapsList.cpp.....	sheets 4 to 4	(1) pages 7- 7 24 lines
6	6	MapsList.h.....	sheets 4 to 4	(1) pages 8- 8 17 lines
7	7	Server.cpp.....	sheets 5 to 5	(1) pages 9- 10 68 lines
8	8	Server.h.....	sheets 6 to 6	(1) pages 11- 11 42 lines
9	9	DataSender.cpp.....	sheets 6 to 7	(2) pages 12- 14 167 lines
10	10	DataSender.h.....	sheets 8 to 8	(1) pages 15- 16 84 lines
11	11	PlayerDataReceiver.cpp	sheets 9 to 9	(1) pages 17- 18 66 lines
12	12	PlayerDataReceiver.h	sheets 10 to 10	(1) pages 19- 19 39 lines
13	13	PlayerDataSender.cpp	sheets 10 to 10	(1) pages 20- 20 39 lines
14	14	PlayerDataSender.h..	sheets 11 to 11	(1) pages 21- 21 38 lines
15	15	ServerProtocol.cpp..	sheets 11 to 12	(2) pages 22- 24 166 lines
16	16	ServerProtocol.h....	sheets 13 to 13	(1) pages 25- 26 69 lines
17	17	Game.cpp.....	sheets 14 to 15	(2) pages 27- 29 148 lines
18	18	Game.h.....	sheets 15 to 15	(1) pages 30- 30 60 lines
19	19	GameParameters.cpp..	sheets 16 to 17	(2) pages 31- 34 203 lines
20	20	GameParameters.h....	sheets 18 to 19	(2) pages 35- 37 143 lines
21	21	Player.cpp.....	sheets 19 to 20	(2) pages 38- 39 73 lines
22	22	Player.h.....	sheets 20 to 21	(2) pages 40- 41 74 lines
23	23	Turn.cpp.....	sheets 21 to 22	(2) pages 42- 43 108 lines
24	24	Turn.h.....	sheets 22 to 22	(1) pages 44- 44 62 lines
25	25	WeaponList.cpp.....	sheets 23 to 23	(1) pages 45- 45 32 lines
26	26	WeaponList.h.....	sheets 23 to 23	(1) pages 46- 46 34 lines
27	27	WormsList.cpp.....	sheets 24 to 24	(1) pages 47- 47 54 lines
28	28	WormsList.h.....	sheets 24 to 24	(1) pages 48- 48 43 lines
29	29	CollisionData.cpp...	sheets 25 to 25	(1) pages 49- 49 17 lines
30	30	CollisionData.h....	sheets 25 to 25	(1) pages 50- 50 27 lines
31	31	CollisionListener.cpp	sheets 26 to 26	(1) pages 51- 52 87 lines
32	32	CollisionListener.h.	sheets 27 to 27	(1) pages 53- 53 26 lines
33	33	RayCastWeaponExploded.cpp	sheets 27 to 27	(1) pages 54- 54 35 lines
34	34	RayCastWeaponExploded.h	sheets 28 to 28	(1) pages 55- 55 28 lines
35	35	BottomBorder.cpp....	sheets 28 to 28	(1) pages 56- 56 24 lines
36	36	BottomBorder.h.....	sheets 29 to 29	(1) pages 57- 57 22 lines
37	37	Girder.cpp.....	sheets 29 to 29	(1) pages 58- 58 48 lines
38	38	Girder.h.....	sheets 30 to 30	(1) pages 59- 59 35 lines
39	39	PhysicalObject.cpp..	sheets 30 to 31	(2) pages 60- 61 90 lines
40	40	PhysicalObject.h....	sheets 31 to 32	(2) pages 62- 63 74 lines
41	41	AirAttack.cpp.....	sheets 32 to 32	(1) pages 64- 64 28 lines
42	42	AirAttack.h.....	sheets 33 to 33	(1) pages 65- 65 23 lines
43	43	AirAttackMissile.cpp	sheets 33 to 33	(1) pages 66- 66 18 lines
44	44	AirAttackMissile.h..	sheets 34 to 34	(1) pages 67- 67 18 lines
45	45	Banana.cpp.....	sheets 34 to 34	(1) pages 68- 68 28 lines
46	46	Banana.h.....	sheets 35 to 35	(1) pages 69- 69 19 lines
47	47	Bat.cpp.....	sheets 35 to 35	(1) pages 70- 70 25 lines
48	48	Bat.h.....	sheets 36 to 36	(1) pages 71- 71 20 lines
49	49	Bazooka.cpp.....	sheets 36 to 36	(1) pages 72- 72 18 lines
50	50	Bazooka.h.....	sheets 37 to 37	(1) pages 73- 73 18 lines
51	51	Dynamite.cpp.....	sheets 37 to 37	(1) pages 74- 74 13 lines
52	52	Dynamite.h.....	sheets 38 to 38	(1) pages 75- 75 16 lines
53	53	FragmentableWeapon.cpp	sheets 38 to 38	(1) pages 76- 76 26 lines
54	54	FragmentableWeapon.h	sheets 39 to 39	(1) pages 77- 77 21 lines
55	55	Fragment.cpp.....	sheets 39 to 39	(1) pages 78- 78 19 lines
56	56	Fragment.h.....	sheets 40 to 40	(1) pages 79- 79 21 lines
57	57	GreenGrenade.cpp....	sheets 40 to 40	(1) pages 80- 80 14 lines
58	58	GreenGrenade.h.....	sheets 41 to 41	(1) pages 81- 81 16 lines
59	59	HolyGrenade.cpp.....	sheets 41 to 41	(1) pages 82- 82 14 lines
60	60	HolyGrenade.h.....	sheets 42 to 42	(1) pages 83- 83 16 lines
61	61	Mortar.cpp.....	sheets 42 to 42	(1) pages 84- 84 19 lines
62	62	MortarFragment.cpp..	sheets 43 to 43	(1) pages 85- 85 18 lines
63	63	MortarFragment.h....	sheets 43 to 43	(1) pages 86- 86 18 lines
64	64	Mortar.h.....	sheets 44 to 44	(1) pages 87- 87 18 lines
65	65	RedGrenade.cpp.....	sheets 44 to 44	(1) pages 88- 88 15 lines

jun 26, 18 13:59

Table of Content

Page 2/2

67	66	<i>RedGrenadeFragment.cpp</i>	sheets	45 to	45 (1)	pages	89- 89	14 lines
68	67	<i>RedGrenadeFragment.h</i>	sheets	45 to	45 (1)	pages	90- 90	16 lines
69	68	<i>RedGrenade.h.....</i>	sheets	46 to	46 (1)	pages	91- 91	16 lines
70	69	<i>Teleportation.cpp...</i>	sheets	46 to	46 (1)	pages	92- 92	28 lines
71	70	<i>Teleportation.h.....</i>	sheets	47 to	47 (1)	pages	93- 93	21 lines
72	71	<i>Weapon.cpp.....</i>	sheets	47 to	48 (2)	pages	94- 95	115 lines
73	72	<i>WeaponExplodeTime.cpp</i>	sheets	48 to	48 (1)	pages	96- 96	30 lines
74	73	<i>WeaponExplodeTime.h.</i>	sheets	49 to	49 (1)	pages	97- 97	28 lines
75	74	<i>WeaponFactory.cpp...</i>	sheets	49 to	49 (1)	pages	98- 98	55 lines
76	75	<i>WeaponFactory.h.....</i>	sheets	50 to	50 (1)	pages	99- 99	22 lines
77	76	<i>Weapon.h.....</i>	sheets	50 to	51 (2)	pages	100-101	67 lines
78	77	<i>Worm.cpp.....</i>	sheets	51 to	53 (3)	pages	102-105	221 lines
79	78	<i>Worm.h.....</i>	sheets	53 to	54 (2)	pages	106-107	88 lines
80	79	<i>Wind.cpp.....</i>	sheets	54 to	54 (1)	pages	108-108	25 lines
81	80	<i>Wind.h.....</i>	sheets	55 to	55 (1)	pages	109-109	25 lines
82	81	<i>World.cpp.....</i>	sheets	55 to	56 (2)	pages	110-112	143 lines
83	82	<i>World.h.....</i>	sheets	57 to	57 (1)	pages	113-114	84 lines
84	83	<i>main.cpp.....</i>	sheets	58 to	58 (1)	pages	115-115	31 lines