```cpp
#include "ClientHandler.h"
#include "MapsList.h"
#include <iostream>

ClientHandler::ClientHandler(Socket&& client, GamesList& games, std::mutex& mutex_cout):
    client(std::move(ServerProtocol(std::move(client)))), games(games),
    connected(false), mutex_cout(mutex_cout){}

ClientHandler::~ClientHandler(){}

void ClientHandler::run(){
    try{
        while(!this->connected){
            char action = this->client.getProtocol().receiveChar();
            std::string player_name = this->client.getProtocol().receiveString();
            this->client.setName(player_name);
            if (action == CREATE_GAME_ACTION){
                this->createGame();
            } else if (action == JOIN_GAME_ACTION){
                this->joinGame();
            }
        }

    } catch(const std::exception& e){
        std::lock_guard<std::mutex> lock(this->mutex_cout);
        std::cout << "[ERROR] Error con un cliente: " << e.what() << std::endl;
    }
    this->running = false;
}

void ClientHandler::stop(){
    this->client.getProtocol().stop();
}

void ClientHandler::createGame(){
    maps_list_t maps_list = MapsList::getAllMaps();

    size_t size = maps_list.size();
    this->client.getProtocol().sendLength(size);

    for (size_t i = 0; i < size; i++){
        this->client.getProtocol().sendString(maps_list[i]);
    }

    if (size == 0){
        return;
    }

    std::string map = this->client.getProtocol().receiveString();
    std::string game_name = this->client.getProtocol().receiveString();
    int max_players = this->client.getProtocol().receiveLength();

    bool result = this->games.addGame(game_name, map, max_players, this->client);
    if (!result){
        this->client.getProtocol().sendChar(false);
    } else {
        this->connected = true;
    }

}

void ClientHandler::joinGame(){
    games_list_t games_list = this->games.getJoinableGames(this->client.getName(
```



```cpp
    ));

    size_t size = games_list.size();
    this->client.getProtocol().sendLength(size);

    for (size_t i = 0; i < size; i++){
        this->client.getProtocol().sendString(games_list[i]);
    }

    if (size == 0){
        return;
    }

    std::string game_name = this->client.getProtocol().receiveString();

    bool result = this->games.addPlayer(game_name, this->client);

    if (!result){
        this->client.getProtocol().sendChar(false);
    } else {
        this->connected = true;
    }
}
```

Jun 02, 18 12:59 **ClientHandler.h** Page 1/1

```
#ifndef __CLIENTHANDLER_H__
#define __CLIENTHANDLER_H__

#include "Socket.h"
#include "Server.h"
#include "Thread.h"
#include "Player.h"
#include "GamesList.h"
#include <mutex>

class ClientHandler: public Thread{
    private:
        Player client;
        GamesList& games;
        bool connected;
        std::mutex& mutex_cout;

        /* Crea una partida nueva */
        void createGame();

        /* Agrega un jugador a una partida */
        void joinGame();

    public:
        /* Constructor */
        ClientHandler(Socket&& client, GamesList& games, std::mutex& mutex_cout)
;

        /* Destructor */
        ~ClientHandler();

        /* Ejecuta el client handler */
        void run();

        /* Se desconecta abruptamente del cliente */
        void stop();
};

#endif
```

Jun 02, 18 13:44 **GamesList.cpp** Page 1/2

```
#include "GamesList.h"
#include "Path.h"
#include <iostream>

GamesList::GamesList(std::mutex& mutex_cout): mutex_cout(mutex_cout){}

GamesList::~GamesList(){
    for (auto it = this->games.begin(); it != this->games.end(); ++it){
        it->second->join();
        std::lock_guard<std::mutex> lock(this->mutex_cout);
        std::cout << "[INFO] Partida terminada: " << it->first << std::endl;
    }
}

bool GamesList::addGame(const std::string& game_name, const std::string& map, in
t max_players, Player& player){
    std::lock_guard<std::mutex> lock(this->mutex);
    auto it = this->games.find(game_name);
    if (it != this->games.end()){
        return false;
    }

    try{
        std::unique_ptr<Game> game(new Game(max_players, SERVER_CONFIG_FILE, MAP
S_PATH + map));
        this->games[game_name] = std::move(game);
        std::lock_guard<std::mutex> lock(this->mutex_cout);
        std::cout << "[INFO] Nueva partida creada: " << game_name << std::endl;
    } catch (const std::exception& e){
        std::lock_guard<std::mutex> lock(this->mutex_cout);
        std::cout << "[ERROR] Error al crear partida: " << game_name << "-> " << e.what()
<< std::endl;
        return false;
    }

    std::string player_name = player.getName();
    bool result = this->games[game_name]->addPlayer(player);
    if (result){
        std::lock_guard<std::mutex> lock(this->mutex_cout);
        std::cout << "[INFO] El jugador '" << player_name << "' se unio a la partida '" << game
_name << "'" << std::endl;
    }



    if (this->games[game_name]->isFull()){
        this->games[game_name]->start();
    }//////////////////////////////////////////////por ahora, mientras se permit
a un player

    return result;
}

games_list_t GamesList::getJoinableGames(const std::string& player_name){
    std::lock_guard<std::mutex> lock(this->mutex);
    games_list_t joinables;

    for (auto it = this->games.begin(); it != this->games.end(); ++it){
        if (it->second->playerCanJoin(player_name)){
            joinables.push_back(it->first);
        }
    }
    return std::move(joinables);
}

bool GamesList::addPlayer(const std::string& game_name, Player& player){
```

Jun 02, 18 13:44 **GamesList.cpp** Page 2/2

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

Jun 02, 18 12:56 **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 GamesList{
14      private:
15          std::unordered_map<std::string, std::unique_ptr<Game>> games;
16          std::mutex mutex;
17          std::mutex& mutex_cout;
18
19      public:
20          /* Constructor */
21          GamesList(std::mutex& mutex_cout);
22
23          /* Destructor */
24          ~GamesList();
25
26          /* Agrega una patida nueva a la lista */
27          bool addGame(const std::string& game_name, const std::string& map, int m
    ax_players, Player& player);
28
29          /* Devuelve una lista con las partidas a las cuales se puede
30           * unir el jugador */
31          games_list_t getJoinableGames(const std::string& player_name);
32
33          /* Agrega un jugador a la partida */
34          bool addPlayer(const std::string& game_name, Player& player);
35
36          /* Verifica las partidas que terminaron */
37          void checkGames();
38  };
39
40  #endif
```

May 29, 18 13:35 **MapsList.cpp** Page 1/1

```cpp
#include "MapsList.h"
#include "Path.h"

maps_list_t MapsList::getAllMaps(){
    maps_list_t maps_list;

    struct dirent *entry;
    DIR* dir = opendir(MAPS_PATH.c_str());
    if (!dir){
        std::move(maps_list);
    }

    while((entry = readdir(dir))){
        std::string file(entry->d_name);
        if (file.rfind(YAML_EXTENSION) != std::string::npos){
            maps_list.push_back(file);
        }
    }

    closedir(dir);
    return std::move(maps_list);
}
```

May 28, 18 18:21 **MapsList.h** Page 1/1

```cpp
#ifndef __MAPSLIST_H__
#define __MAPSLIST_H__

#include <dirent.h>
#include <vector>
#include <string>

typedef std::vector<std::string> maps_list_t;

class MapsList{
    public:
        /* Devuelve una lista con todos los mapas */
        static maps_list_t getAllMaps();
};

#endif
```

Jun 02, 18 13:01 **Server.cpp** Page 1/1

```cpp
#include <string>
#include <memory>
#include <iostream>
#include "Server.h"
#include "ClientHandler.h"

#define MAX_CLIENT_WAIT 100

Server::Server(const std::string& service, std::mutex& mutex_cout):
    socket(Socket::Server(service.c_str(), MAX_CLIENT_WAIT)), games_list(mutex_cout), mutex_cout(mutex_cout){}

Server::~Server(){
    for (auto it = this->clients.begin(); it != this->clients.end(); ++it){
        (*it)->stop();
        (*it)->join();
    }
}

void Server::run(){
    while (this->running){
        try{
            Socket client = this->socket.acceptClient();
            {
                std::lock_guard<std::mutex> lock(this->mutex_cout);
                std::cout << "[INFO] Nuevo cliente conectado." << std::endl;
            }
            std::unique_ptr<Thread> t(new ClientHandler(std::move(client), this->games_list, this->mutex_cout));
            t->start();
            this->clients.push_back(std::move(t));

            this->check();
        } catch(const std::exception& e){
            if (this->running){
                std::lock_guard<std::mutex> lock(this->mutex_cout);
                std::cout << "[ERROR] " << e.what() << std::endl;
            }
        }
    }
}

void Server::stop(){
    this->running = false;
    this->socket.stop();
}

void Server::check(){
    //Elimino threads que ya terminaron
    auto it = this->clients.begin();
    while (it != this->clients.end()){
        if (!(*it)->isRunning()){
            (*it)->join();
            it = this->clients.erase(it);
        } else {
            ++it;
        }
    }

    this->games_list.checkGames();
}
```

Jun 02, 18 12:53 **Server.h** Page 1/1

```cpp
#ifndef __SERVER_H__
#define __SERVER_H__

#include <string>
#include <list>
#include <memory>
#include <mutex>
#include "Socket.h"
#include "Thread.h"
#include "GamesList.h"

class Server: public Thread{
    private:
        Socket socket;
        std::list<std::unique_ptr<Thread>> clients;
        GamesList games_list;
        std::mutex& mutex_cout;

        /* Elimina los clientes que terminaron su comunicacion
         * de la lista */
        void check();

    public:
        /* Crea el server y lo asocia al puerto indicado */
        Server(const std::string& service, std::mutex& mutex_cout);

        /* Desconecta el server */
        ~Server();

        /* Ejecuta el server */
        void run();

        /* Avisa al server que debe dejar de ejecutarse */
        void stop();
};

#endif
```

May 26, 18 12:13 **CollisionData.cpp** Page 1/1

```cpp
#include "CollisionData.h"
#include "PhysicalObject.h"

CollisionData::CollisionData(std::string type, PhysicalObject* object):
    type(type), object(object){}

CollisionData::~CollisionData(){}

const std::string& CollisionData::getType(){
    return this->type;
}

PhysicalObject* CollisionData::getObject(){
    return this->object;
}
```

May 30, 18 20:03 **CollisionData.h** Page 1/1

```cpp
#ifndef __COLLISIONDATA_H__
#define __COLLISIONDATA_H__

#include <string>


class PhysicalObject;

//Datos de un objeto para determinar colisiones
class CollisionData{
    private:
        std::string type;
        PhysicalObject* object;

    public:
        CollisionData(std::string type, PhysicalObject* object);
        ~CollisionData();

        const std::string& getType();
        PhysicalObject* getObject();
};

#endif
```

Jun 06, 18 21:17 **CollisionListener.cpp** Page 1/2

```cpp
#include "CollisionListener.h"
#include "PhysicalObject.h"
#include "Worm.h"
#include "Girder.h"

CollisionListener::CollisionListener(){}

CollisionListener::~CollisionListener(){}

void CollisionListener::BeginContact(b2Contact* contact){
    CollisionData* dataA = (CollisionData*)contact->GetFixtureA()->GetBody()->GetUserData();
    CollisionData* dataB = (CollisionData*)contact->GetFixtureB()->GetBody()->GetUserData();

    if (dataA->getObject()->isDead() || dataB->getObject()->isDead()){
        return;
    }

    if (dataA->getType() == TYPE_WEAPON){
        if (dataB->getType() == TYPE_WORM){
            int shooter_id = ((Weapon*)dataA->getObject())->getShooterId();
            int worm_id = dataB->getObject()->getId();
            if (shooter_id == worm_id){
                return;
            }
        }
        dataA->getObject()->collideWithSomething(dataB);
    } else if (dataB->getType() == TYPE_WEAPON){
        if (dataA->getType() == TYPE_WORM){
            int shooter_id = ((Weapon*)dataB->getObject())->getShooterId();
            int worm_id = dataA->getObject()->getId();
            if (shooter_id == worm_id){
                return;
            }
        }
        dataB->getObject()->collideWithSomething(dataA);
    }

    if (dataA->getType() == TYPE_WORM && contact->GetFixtureA()->IsSensor() &&
            (dataB->getType() == TYPE_GIRDER || dataB->getType() == TYPE_BORDER)){
        dataA->getObject()->collideWithSomething(dataB);

    } else if (dataB->getType() == TYPE_WORM && contact->GetFixtureB()->IsSensor() &&
            (dataA->getType() == TYPE_GIRDER || dataA->getType() == TYPE_BORDER)){
        dataB->getObject()->collideWithSomething(dataA);
    }
}

void CollisionListener::EndContact(b2Contact* contact){
    CollisionData* dataA = (CollisionData*)contact->GetFixtureA()->GetBody()->GetUserData();
    CollisionData* dataB = (CollisionData*)contact->GetFixtureB()->GetBody()->GetUserData();

    if (dataA->getType() == TYPE_WORM && contact->GetFixtureA()->IsSensor() && dataB->getType() == TYPE_GIRDER){
        bool friction = ((Girder *) dataB->getObject())->hasFriction();
        ((Worm *) dataA->getObject())->endCollissionGirder(friction);
    } else if (dataB->getType() == TYPE_WORM && contact->GetFixtureB()->IsSensor() && dataA->getType() == TYPE_GIRDER){
        bool friction = ((Girder *) dataA->getObject())->hasFriction();
        ((Worm *) dataB->getObject())->endCollissionGirder(friction);
```

Jun 06, 18 21:17 **CollisionListener.cpp** Page 2/2

```cpp
    }

    if (dataA->getType() == TYPE_WEAPON){
        ((Weapon*)dataA->getObject())->removeShooterId();
    }
    if (dataB->getType() == TYPE_WEAPON){
        ((Weapon*)dataB->getObject())->removeShooterId();
    }
}

bool CollisionListener::ShouldCollide(b2Fixture* fixtureA, b2Fixture* fixtureB){
    CollisionData* dataA = (CollisionData*)fixtureA->GetBody()->GetUserData();
    CollisionData* dataB = (CollisionData*)fixtureB->GetBody()->GetUserData();

    if (dataA->getType() == TYPE_WORM && dataB->getType() == TYPE_WORM){
        return false;
    }
    if (dataA->getType() == TYPE_WEAPON && dataB->getType() == TYPE_WEAPON){
        return false;
    }
    return true;
}
```

Jun 06, 18 20:55 **CollisionListener.h** Page 1/1

```
#ifndef __COLLISIONLISTENER_H__
#define __COLLISIONLISTENER_H__

#include <string>
#include "CollisionData.h"
#include "b2WorldCallbacks.h"
#include "b2Contact.h"
#include <list>

class CollisionListener: public b2ContactListener, public b2ContactFilter{
    public:
        CollisionListener();
        ~CollisionListener();

        //Analiza la colision entre dos objetos
        void BeginContact(b2Contact* contact) override;

        //Analiza el fin de colision entre dos objetos
        void EndContact(b2Contact* contact) override;

        //Analiza si dos objetos deben colisionar o no
        bool ShouldCollide(b2Fixture* fixtureA, b2Fixture* fixtureB) override;
};

#endif
```

May 26, 18 12:13 **RayCastWeaponExploded.cpp** Page 1/1

```
#include "RayCastWeaponExploded.h"
#include "Worm.h"

RayCastWeaponExploded::RayCastWeaponExploded(): closest(NULL){}

RayCastWeaponExploded::~RayCastWeaponExploded(){}

b2Body* RayCastWeaponExploded::getClosestWorm(){
    if (!this->closest){
        return NULL;
    }
    CollisionData* data = (CollisionData*)this->closest->GetUserData();
    if (data->getType() != TYPE_WORM){
        this->closest = NULL;
        return NULL;
    }

    this->affected_worms.push_back(this->closest);
    b2Body* closest_worm = this->closest;
    this->closest = NULL;
    return closest_worm;
}

float32 RayCastWeaponExploded::ReportFixture(b2Fixture* fixture, const b2Vec2& p
oint, const b2Vec2& normal, float32 fraction){
    b2Body* closest_body = fixture->GetBody();
    for (auto it = this->affected_worms.begin(); it != this->affected_worms.end(
); ++it){
        if (*it == closest_body){
            return -1;
        }
    }
    this->closest = closest_body;
    return fraction;
}
```

May 30, 18 20:03 **RayCastWeaponExploded.h** Page 1/1

```
#ifndef __RAYCASTWEAPONEXPLODED_H__
#define __RAYCASTWEAPONEXPLODED_H__

#include "b2Body.h"
#include "b2Fixture.h"
#include "b2WorldCallbacks.h"
#include <vector>

class RayCastWeaponExploded: public b2RayCastCallback{
    private:
        std::vector<b2Body*> affected_worms;
        b2Body* closest;

    public:
        RayCastWeaponExploded();
        ~RayCastWeaponExploded();

        //Devuelve el gusano mas cercano a la explosion, si hay
        b2Body* getClosestWorm();

        //Busca al objeto mas cercano a la explosion
        float32 ReportFixture(b2Fixture* fixture, const b2Vec2& point, const b2Vec2& normal, float32 fraction) override;
};

#endif

```

Jun 05, 18 14:07 **Game.cpp** Page 1/2

```
#include "Game.h"
#include "Girder.h"
#include "WeaponFactory.h"


Game::Game(size_t players, const std::string& config_file, const std::string& map):
    players(players), parameters(config_file, map), world(this->parameters){
        this->running = true;
    }

Game::~Game(){
    this->world.stop();
    this->world.join();
    if (data_sender){
        this->data_sender->stop();
        this->data_sender->join();
    }
}

bool Game::addPlayer(Player& player){
    if (this->isFull()){
        return false;
    }

    return this->turn.addPlayer(player);
}

bool Game::isFull(){
    return this->players <= this->turn.getPlayersSize();
}

bool Game::playerCanJoin(const std::string& player_name){
    if (this->isFull()){
        return false;
    }
    return this->turn.playerCanJoin(player_name);
}

void Game::run(){
    this->configure();
    this->world.start();
    this->data_sender->start();

    std::this_thread::sleep_for(std::chrono::milliseconds(100));
    this->waitToWorld();

    while (!this->turn.gameEnded(this->world.getMutex())){
        this->player_turn_active = true;
        this->turn.beginTurn();
        int worm_id = this->turn.getCurrentPlayer().getCurrentWorm().getId();
        int player_id = this->turn.getCurrentPlayer().getId();
        this->data_sender->sendStartTurn(worm_id, player_id, this->world.getWind());

        while (this->player_turn_active){
            try{
                this->turn.getCurrentPlayer().getProtocol().receive(*this, *this->data_sender);
            } catch (const SocketException& e){
                this->player_turn_active = false;
                this->turn.getCurrentPlayer().disconnect();
            }
        }

        this->waitToWorld();
```

Jun 05, 18 14:07 **Game.cpp** Page 2/2

```
64          this->world.update();
65      }
66      std::this_thread::sleep_for(std::chrono::milliseconds(50));
67      this->data_sender->sendEndGame(this->turn.getWinner());
68      this->world.stop();
69      this->data_sender->stop();
70      this->running = false;
71  }
72
73  void Game::configure(){
74      this->data_sender.reset(new DataSender(this->world, this->turn.getPlayers(),
     this->parameters));
75
76      this->data_sender->sendStartGame();
77      this->data_sender->sendBackgroundImage(this->parameters.getBackgroundImage()
    );
78      this->data_sender->sendPlayersId();
79
80      //Asignacion de gusanos
81      std::vector<b2Vec2>& worms_list = this->parameters.getWorms();
82      size_t size = worms_list.size();
83      for (size_t i = 0; i < size; i++){
84          this->turn.addWorm(this->world, this->parameters, worms_list[i], i);
85      }
86      this->turn.distributeWorms(size, this->parameters.getWormsLifeToAdd());
87
88      //Creacion de vigas
89      int max_height = 0;
90      std::vector<GirderParams>& girders_list = this->parameters.getGirders();
91      size = girders_list.size();
92      for (size_t i = 0; i < size; i++){
93          physical_object_ptr girder(new Girder(this->world, this->parameters, gir
    ders_list[i].len, girders_list[i].rotation));
94          this->world.addObject(girder, b2Vec2(girders_list[i].pos_x, girders_list
    [i].pos_y));
95          if (girders_list[i].pos_y > max_height){
96              max_height = girders_list[i].pos_y;
97          }
98      }
99      this->parameters.setMaxHeight(max_height);
100     this->data_sender->sendGirders();
101
102     //Municion de las armas
103     std::map<std::string, int>& ammo = this->parameters.getWeaponsAmmo();
104     this->data_sender->sendWeaponsAmmo(ammo);
105 }
106
107 Worm& Game::getCurrentWorm(){
108     return this->turn.getCurrentPlayer().getCurrentWorm();
109 }
110
111 void Game::endTurn(){
112     this->player_turn_active = false;
113 }
114
115 void Game::waitToWorld(){
116     while (this->world.isActive() || this->data_sender->isActive()){
117         std::this_thread::sleep_for(std::chrono::milliseconds(this->parameters.g
    etGameWaitingWorldSleep()));
118     }
119 }
```

Jun 05, 18 15:24 **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
14  class Player;
15
16  class Game: public Thread{
17      private:
18          size_t players;
19          GameParameters parameters;
20          World world;
21          Turn turn;
22          std::unique_ptr<DataSender> data_sender;
23          bool player_turn_active;
24
25          /* Realiza la configuracion inicial de la partida */
26          void configure();
27
28          /* Espera a que los objetos dejen de moverse */
29          void waitToWorld();
30
31      public:
32          /* Constructor */
33          Game(size_t players, const std::string& config_file, const std::string&
    map);
34
35          /* Destructor */
36          ~Game();
37
38          /* Agrega un jugador a la partida */
39          bool addPlayer(Player& player);
40
41          /* Devuelve true si la partida esta llena */
42          bool isFull();
43
44          /* Devuelve true si el jugador puede unirse a la partida */
45          bool playerCanJoin(const std::string& player_name);
46
47          /* Comienza la partida */
48          void run() override;
49
50          /* Devuelve el worm actual */
51          Worm& getCurrentWorm();
52
53          /* Finaliza el turno */
54          void endTurn();
55  };
56
57  #endif
```

```cpp
#include "GameParameters.h"
#include "ConfigFields.h"
#include "Path.h"
#include <algorithm>
#include <random>

#define WORLD_MAX_HEIGHT "world_max_height"

GameParameters::GameParameters(const std::string& config_file, const std::string& config_editor_file){

    //Compruebo que existan todos los parametros necesarios
    YAML::Node config(YAML::LoadFile(config_file));
    YAML::Node config_editor(YAML::LoadFile(config_editor_file));

    this->float_parameters[DATA_SENDER_SLEEP] = config[DATA_SENDER_SLEEP].as<float>();
    this->float_parameters[GAME_WAITING_WORLD_SLEEP] = config[GAME_WAITING_WORLD_SLEEP].as<float>();
    this->float_parameters[WORLD_SLEEP_AFTER_STEP] = config[WORLD_SLEEP_AFTER_STEP].as<float>();
    this->float_parameters[WORLD_TIME_STEP] = config[WORLD_TIME_STEP].as<float>();

    this->float_parameters[WORMS_LIFE] = config_editor[WORMS_LIFE].as<float>();
    this->float_parameters[WORMS_LIFE_TO_ADD] = config[WORMS_LIFE_TO_ADD].as<float>();
    this->float_parameters[WORM_VELOCITY] = config[WORM_VELOCITY].as<float>();
    this->float_parameters[WORM_EXPLOSION_VELOCITY] = config[WORM_EXPLOSION_VELOCITY].as<float>();
    this->float_parameters[WORM_JUMP_VELOCITY] = config[WORM_JUMP_VELOCITY].as<float>();
    this->float_parameters[WORM_ROLLBACK_VELOCITY] = config[WORM_ROLLBACK_VELOCITY].as<float>();
    this->float_parameters[WORM_JUMP_HEIGHT] = config[WORM_JUMP_HEIGHT].as<float>();
    this->float_parameters[WORM_ROLLBACK_HEIGHT] = config[WORM_ROLLBACK_HEIGHT].as<float>();
    this->float_parameters[WORM_HEIGHT_TO_DAMAGE] = config[WORM_HEIGHT_TO_DAMAGE].as<float>();
    this->float_parameters[WORM_MAX_HEIGHT_DAMAGE] = config[WORM_MAX_HEIGHT_DAMAGE].as<float>();
    this->float_parameters[WEAPONS_VELOCITY] = config[WEAPONS_VELOCITY].as<float>();
    this->float_parameters[WIND_MIN_VELOCITY] = config[WIND_MIN_VELOCITY].as<float>();
    this->float_parameters[WIND_MAX_VELOCITY] = config[WIND_MAX_VELOCITY].as<float>();
    this->float_parameters[GRAVITY] = config[GRAVITY].as<float>();
    this->float_parameters[AIR_MISSILES_SEPARATION] = config[AIR_MISSILES_SEPARATION].as<float>();
    this->float_parameters[MAX_GIRDER_ROTATION_FRICTION] = config[MAX_GIRDER_ROTATION_FRICTION].as<float>();
    this->float_parameters[WORLD_MAX_HEIGHT] = 99999;

    this->weapon_radius = config[WEAPON_RADIUS].as<std::map<std::string, int>>();
    this->weapon_ammo = config_editor[WEAPON_AMMO].as<std::map<std::string, int>>();
    this->weapon_damage = config[WEAPON_DAMAGE].as<std::map<std::string, int>>();
    this->weapon_fragments = config[WEAPON_FRAGMENTS].as<std::map<std::string, int>>();

    std::vector<std::vector<float>> worms_file = config_editor[WORMS_DATA].as<std::vector<std::vector<float>>>();
```



```cpp
    for (auto it = worms_file.begin(); it != worms_file.end(); ++it){
        this->worms.push_back(b2Vec2((*it)[0], (*it)[1]));
    }

    std::vector<std::vector<float>> girders_file = config_editor[GIRDERS_DATA].as<std::vector<std::vector<float>>>();
    for (auto it = girders_file.begin(); it != girders_file.end(); ++it){
        this->girders.push_back(GirderParams((*it)[0], (*it)[1], (*it)[2], (*it)[3]));
    }

    std::string background = BACKGROUND_PATH + config_editor[BACKGROUND_IMAGE].as<std::string>();
    this->background_image = std::move(File(background, FILE_READ_MODE));
}

GameParameters::~GameParameters(){}

int GameParameters::getWormLife(){
    return this->float_parameters[WORMS_LIFE];
}

int GameParameters::getWormsLifeToAdd(){
    return this->float_parameters[WORMS_LIFE_TO_ADD];
}

std::vector<b2Vec2>& GameParameters::getWorms(){
    std::random_device rd;
    std::mt19937 random(rd());

    std::shuffle(this->worms.begin(), this->worms.end(), random);
    return this->worms;
}

std::vector<GirderParams>& GameParameters::getGirders(){
    return this->girders;
}

std::map<std::string, int>& GameParameters::getWeaponsAmmo(){
    return this->weapon_ammo;
}

float GameParameters::getWormVelocity(){
    return this->float_parameters[WORM_VELOCITY];
}

float GameParameters::getWormExplosionVelocity(){
    return this->float_parameters[WORM_EXPLOSION_VELOCITY];
}

float GameParameters::getWormJumpVelocity(){
    return this->float_parameters[WORM_JUMP_VELOCITY];
}

float GameParameters::getWormRollbackVelocity(){
    return this->float_parameters[WORM_ROLLBACK_VELOCITY];
}

float GameParameters::getWormJumpHeight(){
    return this->float_parameters[WORM_JUMP_HEIGHT];
}

float GameParameters::getWormRollbackHeight(){
    return this->float_parameters[WORM_ROLLBACK_HEIGHT];
}
```

```
107 int GameParameters::getWormHeightToDamage(){
108     return this->float_parameters[WORM_HEIGHT_TO_DAMAGE];
109 }
110
111 int GameParameters::getWormMaxHeightDamage(){
112     return this->float_parameters[WORM_MAX_HEIGHT_DAMAGE];
113 }
114
115 float GameParameters::getWeaponsVelocity(){
116     return this->float_parameters[WEAPONS_VELOCITY];
117 }
118
119 int GameParameters::getWeaponDamage(const std::string& weapon){
120     return this->weapon_damage[weapon];
121 }
122
123 int GameParameters::getWeaponRadius(const std::string& weapon){
124     return this->weapon_radius[weapon];
125 }
126
127 int GameParameters::getWeaponFragments(const std::string& weapon){
128     return this->weapon_fragments[weapon];
129 }
130
131 float GameParameters::getWindMinVelocity(){
132     return this->float_parameters[WIND_MIN_VELOCITY];
133 }
134
135 float GameParameters::getWindMaxVelocity(){
136     return this->float_parameters[WIND_MAX_VELOCITY];
137 }
138
139 float GameParameters::getGravity(){
140     return this->float_parameters[GRAVITY];
141 }
142
143 float GameParameters::getAirMissilesSeparation(){
144     return this->float_parameters[AIR_MISSILES_SEPARATION];
145 }
146
147 int GameParameters::getMaxGirderRotationToFriction(){
148     return this->float_parameters[MAX_GIRDER_ROTATION_FRICTION];
149 }
150
151 void GameParameters::setMaxHeight(int height){
152     this->float_parameters[WORLD_MAX_HEIGHT] = height + 15;
153 }
154
155 int GameParameters::getMaxHeight(){
156     return this->float_parameters[WORLD_MAX_HEIGHT];
157 }
158
159 int GameParameters::getDataSenderSleep(){
160     return this->float_parameters[DATA_SENDER_SLEEP];
161 }
162
163 int GameParameters::getGameWaitingWorldSleep(){
164     return this->float_parameters[GAME_WAITING_WORLD_SLEEP];
165 }
166
167 int GameParameters::getWorldSleepAfterStep(){
168     return this->float_parameters[WORLD_SLEEP_AFTER_STEP];
169 }
170
171 float GameParameters::getWorldTimeStep(){
172     return this->float_parameters[WORLD_TIME_STEP];
```



```
173 }
174
175 File& GameParameters::getBackgroundImage(){
176     return this->background_image;
177 }
178
179 GameParameters::GirderParams::GirderParams(size_t len, float pos_x, float pos_y,
     int rotation):
180     len(len), pos_x(pos_x), pos_y(pos_y), rotation(rotation){}
```

Jun 06, 18 20:13 **GameParameters.h** Page 1/2

```
#ifndef __GAMEPARAMETERS_H__
#define __GAMEPARAMETERS_H__

#include <string>
#include <vector>
#include <map>
#include "b2Math.h"
#include "yaml.h"
#include "File.h"

// Clase que lee los archivos de configuracion
// y devuelve los parametros obtenidos
class GameParameters{
    public:
        class GirderParams;

    private:
        std::map<std::string, float> float_parameters;
        std::map<std::string, int> weapon_radius;
        std::map<std::string, int> weapon_ammo;
        std::map<std::string, int> weapon_damage;
        std::map<std::string, int> weapon_fragments;

        std::vector<b2Vec2> worms;
        std::vector<GirderParams> girders;
        File background_image;

    public:
        //Inicializa todos los parametros necesarios para la partida
        GameParameters(const std::string& config_file, const std::string& config
_editor_file);
        ~GameParameters();

        int getWormLife();
        int getWormsLifeToAdd();

        std::vector<b2Vec2>& getWorms();
        std::vector<GirderParams>& getGirders();
        std::map<std::string, int>& getWeaponsAmmo();


        float getWormVelocity();
        float getWormExplosionVelocity();
        float getWormJumpVelocity();
        float getWormRollbackVelocity();
        float getWormJumpHeight();
        float getWormRollbackHeight();

        int getWormHeightToDamage();
        int getWormMaxHeightDamage();

        float getWeaponsVelocity();

        int getWeaponDamage(const std::string& weapon);
        int getWeaponRadius(const std::string& weapon);
        int getWeaponFragments(const std::string& weapon);

        float getWindMinVelocity();
        float getWindMaxVelocity();
        float getGravity();
        float getAirMissilesSeparation();

        int getMaxGirderRotationToFriction();
        void setMaxHeight(int height);
        int getMaxHeight();

```

Jun 06, 18 20:13 **GameParameters.h** Page 2/2

```
        int getDataSenderSleep();
        int getGameWaitingWorldSleep();
        int getWorldSleepAfterStep();
        float getWorldTimeStep();

        File& getBackgroundImage();
};

class GameParameters::GirderParams{
    public:
        size_t len;
        float pos_x;
        float pos_y;
        int rotation;

        GirderParams(size_t len, float pos_x, float pos_y, int rotation);
};

typedef GameParameters::GirderParams GirderParams;

#endif
```

Jun 05, 18 14:07 **Player.cpp** Page 1/1

```cpp
#include "Player.h"

Player::Player(ServerProtocol&& protocol): protocol(std::move(protocol)),
    id(-1), connected(true){}

Player::Player(Player&& other):
    protocol(std::move(other.protocol)), name(std::move(other.name)),
    worms(std::move(other.worms)), id(other.id), connected(other.connected){}

Player::~Player(){}

void Player::setId(int id){
    this->id = id;
}

int Player::getId() const{
    return this->id;
}

Worm& Player::getCurrentWorm(){
    return this->worms.getCurrentWorm();
}

void Player::beginTurn(){
    this->worms.beginTurn();
}

void Player::addWorm(World& world, GameParameters& parameters, const b2Vec2& pos
ition, int id){
    physical_object_ptr worm(new Worm(world, parameters, id, this->id));
    this->worms.add(worm);
    world.addObject(worm, position);
}

void Player::distributeWorms(size_t max, int life_to_add){
    this->worms.distribute(max, life_to_add);
}

bool Player::isDead(){
    return this->worms.isEmpty();
}

ServerProtocol& Player::getProtocol(){
    return this->protocol;
}

void Player::setName(const std::string& name){
    this->name = name;
}

const std::string& Player::getName() const{
    return this->name;
}

bool Player::isConnected() const{
    return this->connected;
}

void Player::disconnect(){
    this->connected = false;
    this->worms.kill();
}
```

Jun 05, 18 14:07 **Player.h** Page 1/1

```cpp
#ifndef __PLAYER_H__
#define __PLAYER_H__

#include "WormsList.h"
#include "ServerProtocol.h"
#include "Worm.h"
#include "World.h"
#include "GameParameters.h"
#include <string>

class Player{
    private:
        ServerProtocol protocol;
        std::string name;
        WormsList worms;
        int id;
        bool connected;

    public:
        Player(ServerProtocol&& protocol);

        Player(Player&& other);

        ~Player();

        void setId(int id);

        int getId() const;

        //Devuelve el gusano actual del jugador
        Worm& getCurrentWorm();

        //Empieza el turno del jugador
        void beginTurn();

        //Agrega un nuevo gusano al jugador
        void addWorm(World& world, GameParameters& parameters, const b2Vec2& pos
ition, int id);

        //Agrega vida a los gusanos del jugador
        //en caso de que tenga menos gusanos que otros jugadores
        void distributeWorms(size_t max, int life_to_add);

        //Devuelve true si el jugador esta muerto
        bool isDead();

        //Devuelve true si el jugador esta desconectado
        bool isConnected() const;

        //Desconecta al jugador
        void disconnect();

        void setName(const std::string& name);

        const std::string& getName() const;

        ServerProtocol& getProtocol();

};

#endif
```

Jun 05, 18 14:07 **Turn.cpp** Page 1/2

```cpp
#include "Turn.h"

Turn::Turn(): current(0){}

Turn::~Turn(){}

bool Turn::addPlayer(Player& player){
    if (!this->playerCanJoin(player.getName())){
        return false;
    }
    player.setId(this->players.size());
    player.getProtocol().sendChar(true);
    this->players.push_back(std::move(player));
    return true;
}

bool Turn::playerCanJoin(const std::string& player_name){
    for (auto it = this->players.begin(); it != this->players.end(); ++it){
        if (it->getName() == player_name){
            return false;
        }
    }
    return true;
}

size_t Turn::getPlayersSize() const{
    return this->players.size();
}

Player& Turn::getCurrentPlayer(){
    return this->players.at(this->current);
}

void Turn::beginTurn(){
    do {
        this->advanceCurrent();
    } while (this->getCurrentPlayer().isDead());
    this->getCurrentPlayer().beginTurn();
}

std::vector<Player>& Turn::getPlayers(){
    return this->players;
}

void Turn::advanceCurrent(){
    this->current++;
    if (this->current >= this->players.size()){
        this->current = 0;
    }
}

void Turn::addWorm(World& world, GameParameters& parameters, b2Vec2 position, int id){
    this->players[this->current].addWorm(world, parameters, position, id);
    this->advanceCurrent();
}

void Turn::distributeWorms(size_t size, int life_to_add){
    int quantity = (size / this->players.size());
    if (size % this->players.size() != 0){
        quantity += 1;
    }

    for (auto it = this->players.begin(); it != this->players.end(); ++it){
        it->distributeWorms(quantity, life_to_add);
    }
```

Jun 05, 18 14:07 **Turn.cpp** Page 2/2

```cpp
}

bool Turn::gameEnded(std::mutex& mutex){
    std::lock_guard<std::mutex> lock(mutex);
    this->winner.clear();
    size_t players_alive = 0;
    for (auto it = this->players.begin(); it != this->players.end(); ++it){
        if (!it->isDead()){
            players_alive++;
            this->winner = it->getName();
        }
    }
    ////////////////////////////////////////////////return players_alive <= 1;
    if (this->players.size() == 1){
    return players_alive == 0;
    }else{return players_alive<=1;}////por ahora con un solo jugador
}

const std::string& Turn::getWinner(){
    return this->winner;
}
```

Jun 05, 18 14:07 **Turn.h** Page 1/1

```
#ifndef __SERVERTURN_H__
#define __SERVERTURN_H__

#include "Player.h"
#include <vector>
#include <string>

class Turn{
    private:
        std::vector<Player> players;
        std::string winner;
        size_t current;

        void advanceCurrent();

    public:
        Turn();
        ~Turn();

        //Agrega un nuevo jugador
        bool addPlayer(Player& player);

        //Devuelve true si el jugador se puede unir a la partida
        bool playerCanJoin(const std::string& player_name);

        //Devuelve la cantidad de jugadores
        size_t getPlayersSize() const;

        //Devuelve un vector con los jugadores
        std::vector<Player>& getPlayers();

        //Devuelve el jugador actual
        Player& getCurrentPlayer();

        //Empieza un nuevo turno, cambiando el jugador actual
        void beginTurn();

        //Agrega un gusano al proximo jugador
        void addWorm(World& world, GameParameters& parameters, b2Vec2 position,
int id);

        //Agrega vida a los jugadores con menos gusanos
        void distributeWorms(size_t size, int life_to_add);

        //Devuelve true si queda uno o ningun jugador vivo
        bool gameEnded(std::mutex& mutex);

        //Devuelve el nombre del jugador ganador
        const std::string& getWinner();

};

#endif
```

Jun 05, 18 14:07 **WormsList.cpp** Page 1/1

```
#include "WormsList.h"

WormsList::WormsList(): current(0){}

WormsList::~WormsList(){}

Worm& WormsList::getCurrentWorm(){
    Worm* worm = (Worm*)this->list[this->current].get();
    return *worm;
}

void WormsList::beginTurn(){
    do {
        this->current++;
        if (this->current >= this->list.size()){
            this->current = 0;
        }
    } while (this->getCurrentWorm().isDead());
}

void WormsList::add(physical_object_ptr worm){
    this->list.push_back(worm);
}

WormsList::WormsList(WormsList&& other): list(std::move(other.list)), current(ot
her.current){}

void WormsList::distribute(size_t max, int life_to_add){
    if (this->list.size() < max){
        for (auto it = this->list.begin(); it != this->list.end(); ++it){
            Worm* worm = (Worm*)it->get();
            worm->addLife(life_to_add);
        }
    }
}

bool WormsList::isEmpty(){
    for (auto it = this->list.begin(); it != this->list.end(); ++it){
        if (!(*it)->isDead()){
            return false;
        }
    }
    return true;
}

void WormsList::kill(){
    for (auto it = this->list.begin(); it != this->list.end(); ++it){
        if (!(*it)->isDead()){
            (*it)->kill();
        }
    }
}
```

Jun 05, 18 15:24 **WormsList.h** Page 1/1

```
#ifndef __WORMSLIST_H__
#define __WORMSLIST_H__

#include <vector>
#include "Worm.h"

class WormsList{
    private:
        std::vector<physical_object_ptr> list;
        size_t current;

    public:
        /* Constructor */
        WormsList();

        /* Destructor */
        ~WormsList();

        /* Devuelve el worm actual */
        Worm& getCurrentWorm();

        /* Comienza el turno, cambiando el gusano actual */
        void beginTurn();

        /* Agrega un worm a la lista */
        void add(physical_object_ptr worm);

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

        /* Aumenta la vida de los worms si la cantidad de
         * worms es menor que la de otros jugadores */
        void distribute(size_t max, int life_to_add);

        /* Devuelve true si todos los worms estan muertos */
        bool isEmpty();

        /* Mata a todos los worms */
        void kill();
};

#endif
```

Jun 02, 18 13:00 **main.cpp** Page 1/1

```
#include "Server.h"
#include "yaml.h"
#include "ConfigFields.h"
#include "Path.h"
#include <iostream>
#include <mutex>

#define EXIT_CHAR 'q'

int main(int argc, const char* argv[]){
    std::mutex mutex_cout;
    try{
        YAML::Node config(YAML::LoadFile(SERVER_CONFIG_FILE));
        Server server(config[SERVER_PORT].as<std::string>(), mutex_cout);
        std::cout << "[LOG] Server iniciado." << std::endl;
        server.start();
        while (std::cin.get() != EXIT_CHAR){}
        {
            std::lock_guard<std::mutex> lock(mutex_cout);
            std::cout << "[LOG] Comenzando el cierre del servidor." << std::endl;
        }
        server.stop();
        server.join();
    } catch(const std::exception& e){
        std::lock_guard<std::mutex> lock(mutex_cout);
        std::cout << "[ERROR] " << e.what() << std::endl;
    }
    return 0;
}
```

May 26, 18 12:13 **BottomBorder.cpp** Page 1/1

```cpp
#include "BottomBorder.h"
#include "b2PolygonShape.h"
#include "b2Fixture.h"

BottomBorder::BottomBorder(World& world): PhysicalObject(world, 0, TYPE_BORDER){
}

BottomBorder::~BottomBorder(){}

void BottomBorder::getBodyDef(b2BodyDef& body_def, const b2Vec2& pos){
    body_def.type = b2_staticBody;
    body_def.position.Set(pos.x, pos.y);
}

void BottomBorder::createFixtures(){
    b2PolygonShape boxShape;
    boxShape.SetAsBox(100000,1);

    b2FixtureDef boxFixtureDef;
    boxFixtureDef.shape = &boxShape;
    boxFixtureDef.density = 1;
    this->body->CreateFixture(&boxFixtureDef);
}
```

May 30, 18 20:03 **BottomBorder.h** Page 1/1

```cpp
#ifndef __BOTTOMBORDER_H__
#define __BOTTOMBORDER_H__

#include "PhysicalObject.h"

//Determina el borde inferior del mundo
class BottomBorder: public PhysicalObject{
    private:
        std::string type;

    protected:
        void getBodyDef(b2BodyDef& body_def, const b2Vec2& pos) override;
        void createFixtures() override;

    public:
        BottomBorder(World& world);
        ~BottomBorder();

};

#endif
```

Jun 05, 18 14:07 **Girder.cpp** Page 1/1

```cpp
#include "Girder.h"
#include "b2PolygonShape.h"
#include "b2Fixture.h"
#include "Math.h"

Girder::Girder(World& world, GameParameters& parameters, size_t size, int rotation):
    PhysicalObject(world, 0, TYPE_GIRDER), size(size), rotation(rotation),
    max_rotation_to_friction(parameters.getMaxGirderRotationToFriction()){}

Girder::~Girder(){}

void Girder::getBodyDef(b2BodyDef& body_def, const b2Vec2& pos){
    body_def.type = b2_staticBody;
    body_def.position.Set(pos.x, pos.y);
}

void Girder::createFixtures(){
    b2PolygonShape boxShape;
    boxShape.SetAsBox(this->size / 2.0, girder_height / 2, b2Vec2(0, 0), Math::degreesToRadians(this->rotation));

    b2FixtureDef boxFixtureDef;
    boxFixtureDef.shape = &boxShape;
    boxFixtureDef.density = 1;
    this->body->CreateFixture(&boxFixtureDef);
}

size_t Girder::getSize(){
    return this->size;
}

int Girder::getRotation(){
    return this->rotation;
}

bool Girder::hasFriction(){
    return this->getAngle() < this->max_rotation_to_friction;
}

int Girder::getAngle(){
    int angle = this->rotation;
    if (angle > 90){
        angle = 180 - angle;
    }
    return angle;
}
```

Jun 05, 18 14:07 **Girder.h** Page 1/1

```cpp
#ifndef __GIRDER_H__
#define __GIRDER_H__

#include "PhysicalObject.h"
#include "GameParameters.h"

class Girder: public PhysicalObject{
    private:
        size_t size;
        int rotation;
        int max_rotation_to_friction;

    protected:
        void getBodyDef(b2BodyDef& body_def, const b2Vec2& pos) override;
        void createFixtures() override;

    public:
        Girder(World& world, GameParameters& parameters, size_t size, int rotation);
        ~Girder();

        //Devuelve la longitud de la viga
        size_t getSize();

        //Devuelve la rotacion de la viga
        int getRotation();

        //Devuelve true si la viga tiene friccion
        bool hasFriction();

        //Devuelve la rotacion normalizada
        int getAngle();

};

#endif
```

```cpp
#include "PhysicalObject.h"
#include "World.h"

PhysicalObject::PhysicalObject(World& world, int id, const std::string& type):
    world(world), body(NULL), is_dead(false), id(id), type(type), last_position(-1, -1),
    last_position_sent(false), data_updated(false), collision_data(type, this){}

PhysicalObject::~PhysicalObject(){}

void PhysicalObject::initializeBody(b2Body* body){
    this->body = body;
    this->body->SetUserData(&this->collision_data);
    this->createFixtures();
    this->setInitialVelocity();
}

void PhysicalObject::destroyBody(){
    this->body = NULL;
    this->is_dead = true;
}

b2Vec2 PhysicalObject::getPosition(){
    if (this->body){
        return this->body->GetPosition();
    }
    return b2Vec2(-100, 0);
}

b2Body* PhysicalObject::getBody(){
    return this->body;
}

bool PhysicalObject::isMoving(){
    if (!this->body || this->is_dead){
        return false;
    }
    b2Vec2 pos = this->body->GetPosition();
    bool moved_x = (int)(pos.x * UNIT_TO_SEND) != (int)(this->last_position.x * UNIT_TO_SEND);
    bool moved_y = (int)(pos.y * UNIT_TO_SEND) != (int)(this->last_position.y * UNIT_TO_SEND);
    this->last_position = pos;
    bool moved = moved_x || moved_y;
    if (moved || this->data_updated){
        this->last_position_sent = false;
        this->data_updated = false;
        return true;
    }
    if (!this->body->IsAwake() && !this->last_position_sent){
        this->last_position_sent = true;
        this->data_updated = false;
        return true;
    }
    return false;
}

bool PhysicalObject::isActive(){
    if (!this->body){
        return false;
    }
    return this->body->IsAwake();
}

bool PhysicalObject::isDead(){
    return this->is_dead;
```



```cpp
}

bool PhysicalObject::isWindAffected(){
    return false;
}

void PhysicalObject::kill(){
    this->is_dead = true;
}

int PhysicalObject::getId(){
    return this->id;
}

const std::string& PhysicalObject::getType(){
    return this->type;
}

void PhysicalObject::setInitialVelocity(){}

void PhysicalObject::collideWithSomething(CollisionData *other){}
```

Jun 06, 18 21:33 **PhysicalObject.h** Page 1/2

```
#ifndef __PHYSICALOBJECT_H__
#define __PHYSICALOBJECT_H__

#include "b2Body.h"
#include "CollisionData.h"
#include "ObjectSizes.h"
#include "ObjectTypes.h"
#include <string>
#include <memory>

class World;

class PhysicalObject{
    protected:
        World& world;
        b2Body* body;
        bool is_dead;
        int id;
        const std::string& type;
        b2Vec2 last_position;
        bool last_position_sent;
        bool data_updated;
        CollisionData collision_data;

        virtual void createFixtures() = 0;
        virtual void setInitialVelocity();

    public:
        PhysicalObject(World& world, int id, const std::string& type);
        virtual ~PhysicalObject();

        //Inicializa el cuerpo del objeto
        void initializeBody(b2Body* body);

        //Destruye el cuerpo del objeto
        void destroyBody();

        //Devuelve la posicion del objeto
        b2Vec2 getPosition();

        //Devuelve el cuerpo del objeto
        b2Body* getBody();

        //Devuelve true si el objeto se esta moviendo
        virtual bool isMoving();

        //Devuelve true si el objeto esta activo
        virtual bool isActive();

        //Devuelve true si el objeto esta muerto
        virtual bool isDead();

        //Devuelve true si el objeto es afectado por el viento
        virtual bool isWindAffected();

        //Mata al objeto
        void kill();

        int getId();

        //Devuelve el tipo del objeto
        const std::string& getType();

        virtual void getBodyDef(b2BodyDef& body_def, const b2Vec2& pos) = 0;

        //Colisiona con otro objeto
        virtual void collideWithSomething(CollisionData *other);

};

typedef std::shared_ptr<PhysicalObject> physical_object_ptr;

#endif
```

Jun 02, 18 13:11 **AirAttack.cpp** Page 1/1

```cpp
#include "AirAttack.h"
#include "WeaponFactory.h"
#include "Worm.h"

AirAttack::AirAttack(World& world, GameParameters& parameters):
    Weapon(world, parameters, 0), missiles_separation(parameters.getAirMissilesSeparation()){}

AirAttack::~AirAttack(){}

const std::string& AirAttack::getName(){
    return AIR_ATTACK_NAME;
}

void AirAttack::shoot(char dir, int angle, int power, int time, int shooter_id){
}

void AirAttack::shoot(Worm& shooter, b2Vec2 pos){
    int missiles = this->parameters.getWeaponFragments(AIR_ATTACK_NAME);
    float pos_x = pos.x - missiles * this->missiles_separation / 2;
    float pos_y = this->parameters.getMaxHeight();
    WeaponFactory factory(this->world, this->parameters);
    for (int i = 0; i < missiles; i++, pos_x += this->missiles_separation){
        physical_object_ptr missile = factory.getWeapon(AIR_ATTACK_MISSILE_NAME);
        this->world.addObject(missile, b2Vec2(pos_x, pos_y));
    };
}
```

Jun 02, 18 13:11 **AirAttack.h** Page 1/1

```cpp
#ifndef __SERVERAIRATTACK_H__
#define __SERVERAIRATTACK_H__

#include "Weapon.h"

class AirAttack: public Weapon{
    private:
        float missiles_separation;

    public:

        AirAttack(World& world, GameParameters& parameters);
        ~AirAttack();

        const std::string& getName() override;

        void shoot(char dir, int angle, int power, int time, int shooter_id) override;

        void shoot(Worm& shooter, b2Vec2 pos) override;

};

#endif
```

May 29, 18 14:09 **AirAttackMissile.cpp** Page 1/1

```cpp
#include "AirAttackMissile.h"

AirAttackMissile::AirAttackMissile(World& world, GameParameters& parameters):
    Weapon(world, parameters, parameters.getWeaponDamage(AIR_ATTACK_MISSILE_NAME
), parameters.getWeaponRadius(AIR_ATTACK_MISSILE_NAME)){}

AirAttackMissile::~AirAttackMissile(){}

const std::string& AirAttackMissile::getName(){
    return AIR_ATTACK_MISSILE_NAME;
}

bool AirAttackMissile::isWindAffected(){
    return true;
}
```

May 26, 18 12:13 **AirAttackMissile.h** Page 1/1

```cpp
#ifndef __SERVERAIRATTACKMISSILE_H__
#define __SERVERAIRATTACKMISSILE_H__

#include "Weapon.h"

class AirAttackMissile: public Weapon{
    public:

        AirAttackMissile(World& world, GameParameters& parameters);
        ~AirAttackMissile();

        const std::string& getName() override;

        bool isWindAffected() override;

};

#endif
```

Jun 05, 18 14:08 **Banana.cpp** Page 1/1

```cpp
#include "Banana.h"
#include "b2Fixture.h"
#include "b2CircleShape.h"

Banana::Banana(World& world, GameParameters& parameters):
    Weapon(world, parameters, parameters.getWeaponDamage(BANANA_NAME), parameters.getWeaponRadius(BANANA_NAME)){}

Banana::~Banana(){}

const std::string& Banana::getName(){
    return BANANA_NAME;
}

void Banana::createFixtures(){
    b2CircleShape circleShape;
    circleShape.m_p.Set(0, 0);
    circleShape.m_radius = weapon_size / 2;

    b2FixtureDef fixtureDef;
    fixtureDef.shape = &circleShape;
    fixtureDef.density = 4;
    fixtureDef.restitution = 0.9; //rebotable
    this->body->CreateFixture(&fixtureDef);
}
```

May 26, 18 12:13 **Banana.h** Page 1/1

```cpp
#ifndef __SERVERBANANA_H__
#define __SERVERBANANA_H__

#include "Weapon.h"

class Banana: public Weapon{
    protected:
        void createFixtures() override;

    public:

        Banana(World& world, GameParameters& parameters);
        ~Banana();

        const std::string& getName() override;

};

#endif
```

May 26, 18 12:13 **Bat.cpp** Page 1/1

```cpp
#include "Bat.h"

Bat::Bat(World& world, GameParameters& parameters):
    Weapon(world, parameters, parameters.getWeaponDamage(BAT_NAME), parameters.getWeaponRadius(BAT_NAME)){}

Bat::~Bat(){}

const std::string& Bat::getName(){
    return BAT_NAME;
}

void Bat::setInitialVelocity(){
    this->explode();
}

void Bat::explode(){
    b2Vec2 center = this->body->GetPosition();
    this->attackWormExplosion(center, this->angle);

    this->waiting_to_explode = false;
    this->is_dead = true;
}
```

May 26, 18 12:13 **Bat.h** Page 1/1

```cpp
#ifndef __SERVERBAT_H__
#define __SERVERBAT_H__

#include "Weapon.h"

class Bat: public Weapon{
    public:
        Bat(World& world, GameParameters& parameters);
        ~Bat();

        const std::string& getName() override;

        void setInitialVelocity() override;

        void explode() override;
};

#endif
```

May 26, 18 12:13 **Bazooka.cpp** Page 1/1

```cpp
#include "Bazooka.h"

Bazooka::Bazooka(World& world, GameParameters& parameters):
    Weapon(world, parameters, parameters.getWeaponDamage(BAZOOKA_NAME), parameters.getWeaponRadius(BAZOOKA_NAME)){}

Bazooka::~Bazooka(){}

const std::string& Bazooka::getName(){
    return BAZOOKA_NAME;
}

bool Bazooka::isWindAffected(){
    return true;
}
```

May 26, 18 12:13 **Bazooka.h** Page 1/1

```cpp
#ifndef __SERVERBAZOOKA_H__
#define __SERVERBAZOOKA_H__

#include "Weapon.h"

class Bazooka: public Weapon{
    public:

        Bazooka(World& world, GameParameters& parameters);
        ~Bazooka();

        const std::string& getName() override;
        bool isWindAffected() override;

};

#endif
```

Jun 05, 18 14:09 **Dynamite.cpp** Page 1/1

```cpp
#include "Dynamite.h"

Dynamite::Dynamite(World& world, GameParameters& parameters):
    Weapon(world, parameters, parameters.getWeaponDamage(DYNAMITE_NAME), parameters.getWeaponRadius(DYNAMITE_NAME)){}

Dynamite::~Dynamite(){}

const std::string& Dynamite::getName(){
    return DYNAMITE_NAME;
}
```

Jun 05, 18 14:09 **Dynamite.h** Page 1/1

```cpp
#ifndef __SERVERDYNAMITE_H__
#define __SERVERDYNAMITE_H__

#include "Weapon.h"

class Dynamite: public Weapon{

    public:
        Dynamite(World& world, GameParameters& parameters);
        ~Dynamite();

        const std::string& getName() override;
};

#endif
```

Jun 06, 18 20:08 **FragmentableWeapon.cpp** Page 1/1

```cpp
#include "FragmentableWeapon.h"
#include "WeaponFactory.h"
#include "Fragment.h"
#include "Math.h"

FragmentableWeapon::FragmentableWeapon(World& world, GameParameters& parameters,
 int damage, int fragments, int radius):
    Weapon(world, parameters, damage, radius), fragments(fragments){}

FragmentableWeapon::~FragmentableWeapon(){}

void FragmentableWeapon::explode(){
    WeaponFactory factory(this->world, this->parameters);
    for (float fragment_angle = 0; fragment_angle < 360; fragment_angle+= (360 /
 this->fragments)){
        physical_object_ptr fragment = factory.getWeapon(this->getName() + FRAGM
ENT);

        b2Vec2 center = this->body->GetPosition() + 0.3 * b2Vec2(Math::cosDegree
s(fragment_angle),
                                                                 Math::sinDegrees(frag
ment_angle));
         ((Fragment *) fragment.get())->setShootPosition(center);
         ((Fragment*)fragment.get())->shoot(fragment_angle);
        this->world.addWeaponFragment(fragment);
    }
    Weapon::explode();
}
```

May 30, 18 20:03 **FragmentableWeapon.h** Page 1/1

```cpp
#ifndef __FRAGMENTABLEWEAPON_H__
#define __FRAGMENTABLEWEAPON_H__

#include "Weapon.h"

class FragmentableWeapon: public Weapon{
    protected:
        int fragments;

    public:

        FragmentableWeapon(World& world, GameParameters& parameters, int damage,
 int fragments, int radius);
        virtual ~FragmentableWeapon();

        //Explota el arma y lanza fragmentos
        void explode();
};

#endif
```

Jun 06, 18 20:08 **Fragment.cpp** Page 1/1

```cpp
#include "Fragment.h"

Fragment::Fragment(World& world, GameParameters& parameters, int damage, int radius):
    Weapon(world, parameters, damage, radius){}

Fragment::~Fragment(){}

void Fragment::setShootPosition(b2Vec2 pos){
    this->shoot_position = pos;
}

b2Vec2 Fragment::getShootPosition(){
    return this->shoot_position;
}

void Fragment::shoot(int angle){
    Weapon::shoot(1, angle, -1, -1, -1);
}
```

Jun 06, 18 20:08 **Fragment.h** Page 1/1

```cpp
#ifndef __SERVERFRAGMENT_H__
#define __SERVERFRAGMENT_H__

#include "Weapon.h"

class Fragment: public Weapon{
    private:
        b2Vec2 shoot_position;

    public:

        Fragment(World& world, GameParameters& parameters, int damage, int radius);
        ~Fragment();

        void setShootPosition(b2Vec2 pos);
        b2Vec2 getShootPosition();

        void shoot(int angle);

};

#endif
```

May 26, 18 12:13 **GreenGrenade.cpp** Page 1/1

```cpp
#include "GreenGrenade.h"

GreenGrenade::GreenGrenade(World& world, GameParameters& parameters):
    Weapon(world, parameters, parameters.getWeaponDamage(GREEN_GRENADE_NAME), parameters.getWeaponRadius(GREEN_GRENADE_NAME)){}

GreenGrenade::~GreenGrenade(){}

const std::string& GreenGrenade::getName(){
    return GREEN_GRENADE_NAME;
}
```

May 26, 18 12:13 **GreenGrenade.h** Page 1/1

```cpp
#ifndef __SERVERGREENGRENADE_H__
#define __SERVERGREENGRENADE_H__

#include "Weapon.h"

class GreenGrenade: public Weapon{
    public:

        GreenGrenade(World& world, GameParameters& parameters);
        ~GreenGrenade();

        const std::string& getName() override;
};

#endif
```

May 26, 18 12:13 **HolyGrenade.cpp** Page 1/1

```cpp
#include "HolyGrenade.h"

HolyGrenade::HolyGrenade(World& world, GameParameters& parameters):
    Weapon(world, parameters, parameters.getWeaponDamage(HOLY_GRENADE_NAME), parameters.getWeaponRadius(HOLY_GRENADE_NAME)){}

HolyGrenade::~HolyGrenade(){}

const std::string& HolyGrenade::getName(){
    return HOLY_GRENADE_NAME;
}
```

May 26, 18 12:13 **HolyGrenade.h** Page 1/1

```cpp
#ifndef __SERVERHOLYGRENADE_H__
#define __SERVERHOLYGRENADE_H__

#include "Weapon.h"

class HolyGrenade: public Weapon{
    public:

        HolyGrenade(World& world, GameParameters& parameters);
        ~HolyGrenade();

        const std::string& getName() override;
};

#endif
```

May 26, 18 12:13 **Mortar.cpp** Page 1/1

```cpp
#include "Mortar.h"

Mortar::Mortar(World& world, GameParameters& parameters):
    FragmentableWeapon(world, parameters, parameters.getWeaponDamage(MORTAR_NAME), parameters.getWeaponFragments(MORTAR_NAME), parameters.getWeaponRadius(MORTAR_NAME)){}

Mortar::~Mortar(){}

const std::string& Mortar::getName(){
    return MORTAR_NAME;
}

bool Mortar::isWindAffected(){
    return true;
}
```

May 26, 18 12:13 **MortarFragment.cpp** Page 1/1

```cpp
#include "MortarFragment.h"

MortarFragment::MortarFragment(World& world, GameParameters& parameters):
    Fragment(world, parameters, parameters.getWeaponDamage(MORTAR_FRAGMENTS_NAME), parameters.getWeaponRadius(MORTAR_FRAGMENTS_NAME)){}

MortarFragment::~MortarFragment(){}

const std::string& MortarFragment::getName(){
    return MORTAR_FRAGMENTS_NAME;
}

bool MortarFragment::isWindAffected(){
    return true;
}
```

May 26, 18 12:13 **MortarFragment.h** Page 1/1

```
#ifndef __SERVERMORTARFRAGMENT_H__
#define __SERVERMORTARFRAGMENT_H__

#include "Fragment.h"

class MortarFragment: public Fragment{
    public:

        MortarFragment(World& world, GameParameters& parameters);
        ~MortarFragment();

        const std::string& getName() override;

        bool isWindAffected() override;

};

#endif
```

May 26, 18 12:13 **Mortar.h** Page 1/1

```
#ifndef __SERVERMORTAR_H__
#define __SERVERMORTAR_H__

#include "FragmentableWeapon.h"

class Mortar: public FragmentableWeapon{
    public:

        Mortar(World& world, GameParameters& parameters);
        ~Mortar();

        const std::string& getName() override;

        bool isWindAffected() override;
};

#endif
```

May 26, 18 12:13 **RedGrenade.cpp** Page 1/1

```cpp
#include "RedGrenade.h"

RedGrenade::RedGrenade(World& world, GameParameters& parameters):
    FragmentableWeapon(world, parameters, parameters.getWeaponDamage(RED_GRENADE_NAME), parameters.getWeaponFragments(RED_GRENADE_NAME), parameters.getWeaponRadius(RED_GRENADE_NAME)){}

RedGrenade::~RedGrenade(){}

const std::string& RedGrenade::getName(){
    return RED_GRENADE_NAME;
}
```

May 26, 18 12:13 **RedGrenadeFragment.cpp** Page 1/1

```cpp
#include "RedGrenadeFragment.h"

RedGrenadeFragment::RedGrenadeFragment(World& world, GameParameters& parameters)
:
    Fragment(world, parameters, parameters.getWeaponDamage(RED_GRENADE_FRAGMENTS_NAME), parameters.getWeaponRadius(RED_GRENADE_FRAGMENTS_NAME)){}

RedGrenadeFragment::~RedGrenadeFragment(){}

const std::string& RedGrenadeFragment::getName(){
    return RED_GRENADE_FRAGMENTS_NAME;
}
```

May 26, 18 12:13 **RedGrenadeFragment.h** Page 1/1

```
#ifndef __SERVERREDGRENADEFRAGMENT_H__
#define __SERVERREDGRENADEFRAGMENT_H__

#include "Fragment.h"

class RedGrenadeFragment: public Fragment{
    public:

        RedGrenadeFragment(World& world, GameParameters& parameters);
        ~RedGrenadeFragment();

        const std::string& getName() override;

};

#endif
```

May 26, 18 12:13 **RedGrenade.h** Page 1/1

```
#ifndef __SERVERREDGRENADE_H__
#define __SERVERREDGRENADE_H__

#include "FragmentableWeapon.h"

class RedGrenade: public FragmentableWeapon{
    public:

        RedGrenade(World& world, GameParameters& parameters);
        ~RedGrenade();

        const std::string& getName() override;
};

#endif
```

Jun 03, 18 21:28 **Teleportation.cpp** Page 1/1

```cpp
#include "Teleportation.h"
#include "Worm.h"
#include <mutex>

Teleportation::Teleportation(World& world, GameParameters& parameters):
    Weapon(world, parameters, 0){}

Teleportation::~Teleportation(){}

const std::string& Teleportation::getName(){
    return TELEPORT_NAME;
}

void Teleportation::shoot(char dir, int angle, int power, int time, int shooter_id){}

void Teleportation::shoot(Worm& shooter, b2Vec2 pos){
    pos.x += (worm_size / 2);
    pos.y += (worm_size / 2);
    std::lock_guard<std::mutex> lock(this->world.getMutex());
    b2Body* body = shooter.getBody();
    if (body){
        shooter.getBody()->SetTransform(pos, 0);
        shooter.getBody()->SetAwake(true);
    }
}
```

Jun 02, 18 13:11 **Teleportation.h** Page 1/1

```cpp
#ifndef __SERVERTELEPORTATION_H__
#define __SERVERTELEPORTATION_H__

#include "Weapon.h"

class Teleportation: public Weapon{
    public:

        Teleportation(World& world, GameParameters& parameters);
        ~Teleportation();

        const std::string& getName() override;

        void shoot(char dir, int angle, int power, int time, int shooter_id) override;

        //Teletransporta al gusano
        void shoot(Worm& shooter, b2Vec2 pos) override;

};

#endif
```

```cpp
#include "Weapon.h"
#include "b2Fixture.h"
#include "b2CircleShape.h"
#include "CollisionData.h"
#include "Worm.h"
#include "Math.h"

int Weapon::weapon_id = 1;

Weapon::Weapon(World& world, GameParameters& parameters, int damage, int radius)
:
    PhysicalObject(world, Weapon::weapon_id++, TYPE_WEAPON), parameters(parameters),
    damage(damage), radius(radius),
    waiting_to_explode(false), time_to_explode(-1), angle(MAX_WEAPON_ANGLE + 1), power(-1),
    shooter_id(-1), explode_time(world, *this){}

Weapon::~Weapon(){
    this->explode_time.join();
}

bool Weapon::isActive(){
    return this->waiting_to_explode || PhysicalObject::isActive();
}

void Weapon::shoot(char dir, int angle, int power, int time, int shooter_id){
    if (dir == -1 && angle <= MAX_WEAPON_ANGLE){
        angle = 180 - angle;
    }
    this->time_to_explode = time;
    this->angle = angle;
    this->power = power;
    this->shooter_id = shooter_id;
}

void Weapon::shoot(Worm& shooter, b2Vec2 pos){}

void Weapon::getBodyDef(b2BodyDef& body_def, const b2Vec2& pos){
    body_def.type = b2_dynamicBody;
    body_def.position.Set(pos.x, pos.y);
    body_def.fixedRotation = true;
    body_def.bullet = true;
}

void Weapon::createFixtures(){
    b2CircleShape circleShape;
    circleShape.m_p.Set(0, 0);
    circleShape.m_radius = weapon_size / 2;

    b2FixtureDef fixtureDef;
    fixtureDef.shape = &circleShape;
    fixtureDef.density = 4;
    this->body->CreateFixture(&fixtureDef);
}

void Weapon::setInitialVelocity(){
    if (this->angle <= 360){
        int velocity = this->parameters.getWeaponsVelocity();
        if (this->power != -1){
            velocity *= this->power / 1000;
        }
        b2Vec2 linear_velocity(velocity * Math::cosDegrees(this->angle), velocity * Math::sinDegrees(this->angle));
        this->body->SetLinearVelocity(linear_velocity);
    }
```



```cpp
    this->waiting_to_explode = true;
    this->explode_time.setTime(this->time_to_explode);
    this->explode_time.start();
}


void Weapon::explode(){
    b2Vec2 center = this->body->GetPosition();
    for (float bullet_angle = 0; bullet_angle < 360; bullet_angle+= 5){
        this->attackWormExplosion(center, bullet_angle);
    }

    this->explode_time.stop();
    this->waiting_to_explode = false;
    this->is_dead = true;
}

void Weapon::attackWormExplosion(const b2Vec2& center, int angle){
    b2Vec2 end = center + this->radius * b2Vec2(Math::cosDegrees(angle), Math::sinDegrees(angle));
    b2Body* closest_body = this->world.getClosestObject(&this->explosion, center, end);
    if (closest_body){
        Worm* worm = (Worm*)((CollisionData*)closest_body->GetUserData())->getObject();
        float distance = b2Distance(center, worm->getPosition());
        int worm_damage = this->damage * (1 - distance / (2 * this->radius)); // Justo en el borde hace la mitad de danio
        worm->receiveWeaponDamage(worm_damage, center);
    }
}

void Weapon::collideWithSomething(CollisionData *other){
    if (this->time_to_explode == -1){
        this->explode_time.stop();
        this->explode();
    } else if (other->getType() == TYPE_BORDER){
        this->explode_time.stop();
        this->is_dead = true;
    }
}

int Weapon::getShooterId() const{
    return this->shooter_id;
}

void Weapon::removeShooterId(){
    this->shooter_id = -1;
}
```

May 26, 18 12:13 **WeaponExplodeTime.cpp** Page 1/1

```cpp
#include "WeaponExplodeTime.h"
#include "Weapon.h"
#include "World.h"

WeaponExplodeTime::WeaponExplodeTime(World& world, Weapon& weapon):
    weapon(weapon), world(world), time(-1){}

WeaponExplodeTime::~WeaponExplodeTime(){}

void WeaponExplodeTime::setTime(int time){
    this->time = time;
}

void WeaponExplodeTime::run(){
    if (this->time > 0){
        int passed = 0;
        while (this->running && passed < this->time){
            std::this_thread::sleep_for(std::chrono::seconds(1));
            passed++;
        }
        if (this->running){
            std::lock_guard<std::mutex> lock(this->world.getMutex());
            if (!this->weapon.isDead()){
                this->weapon.explode();
                this->world.removeTimedWeapon(this->weapon);
            }
        }
    }

}
```

May 30, 18 20:03 **WeaponExplodeTime.h** Page 1/1

```cpp
#ifndef __WEAPONEXPLODETIME_H__
#define __WEAPONEXPLODETIME_H__

#include "Thread.h"
#include <mutex>

class Weapon;
class World;

class WeaponExplodeTime: public Thread{
    private:
        Weapon& weapon;
        World& world;
        int time;

    public:
        WeaponExplodeTime(World& world, Weapon& weapon);
        ~WeaponExplodeTime();

        void setTime(int time);

        //Cuenta el tiempo que falta para que el arma explote
        void run() override;

};

#endif
```

May 26, 18 12:13 **WeaponFactory.cpp** Page 1/1

```cpp
#include "WeaponFactory.h"
#include "WeaponNames.h"

#include "Bazooka.h"
#include "Dynamite.h"
#include "RedGrenade.h"
#include "RedGrenadeFragment.h"
#include "GreenGrenade.h"
#include "HolyGrenade.h"
#include "Banana.h"
#include "Teleportation.h"
#include "AirAttack.h"
#include "AirAttackMissile.h"
#include "Mortar.h"
#include "MortarFragment.h"
#include "Bat.h"

WeaponFactory::WeaponFactory(World& world, GameParameters& parameters):
    world(world), parameters(parameters){}

WeaponFactory::~WeaponFactory(){}

physical_object_ptr WeaponFactory::getWeapon(const std::string& name){
    if (name == BAZOOKA_NAME){
        return physical_object_ptr(new Bazooka(this->world, this->parameters));
    } else if (name == DYNAMITE_NAME){
        return physical_object_ptr(new Dynamite(this->world, this->parameters));
    } else if (name == RED_GRENADE_NAME){
        return physical_object_ptr(new RedGrenade(this->world, this->parameters)
);
    } else if (name == RED_GRENADE_FRAGMENTS_NAME){
        return physical_object_ptr(new RedGrenadeFragment(this->world, this->par
ameters));
    } else if (name == GREEN_GRENADE_NAME){
        return physical_object_ptr(new GreenGrenade(this->world, this->parameter
s));
    } else if (name == HOLY_GRENADE_NAME){
        return physical_object_ptr(new HolyGrenade(this->world, this->parameters
));
    } else if (name == MORTAR_NAME){
        return physical_object_ptr(new Mortar(this->world, this->parameters));
    } else if (name == MORTAR_FRAGMENTS_NAME){
        return physical_object_ptr(new MortarFragment(this->world, this->paramet
ers));
    } else if (name == BANANA_NAME){
        return physical_object_ptr(new Banana(this->world, this->parameters));
    } else if (name == BAT_NAME){
        return physical_object_ptr(new Bat(this->world, this->parameters));
    } else if (name == TELEPORT_NAME){
        return physical_object_ptr(new Teleportation(this->world, this->paramete
rs));
    } else if (name == AIR_ATTACK_NAME){
        return physical_object_ptr(new AirAttack(this->world, this->parameters))
;
    } else if (name == AIR_ATTACK_MISSILE_NAME){
        return physical_object_ptr(new AirAttackMissile(this->world, this->param
eters));
    }

    throw std::runtime_error(name + ": El arma no existe.");
}
```

May 30, 18 20:03 **WeaponFactory.h** Page 1/1

```cpp
#ifndef __WEAPONFACTORY_H__
#define __WEAPONFACTORY_H__

#include "World.h"
#include "GameParameters.h"

class WeaponFactory{
    private:
        World& world;
        GameParameters& parameters;

    public:
        WeaponFactory(World& world, GameParameters& parameters);
        ~WeaponFactory();

        //Devuelve el arma pedida
        physical_object_ptr getWeapon(const std::string& name);

};

#endif
```

Jun 06, 18 21:17 **Weapon.h** Page 1/1

```
#ifndef __WEAPON_H__
#define __WEAPON_H__

#include "PhysicalObject.h"
#include "GameParameters.h"
#include "World.h"
#include "WeaponExplodeTime.h"
#include <string>
#include "WeaponNames.h"
#include "RayCastWeaponExploded.h"

class Worm;

class Weapon: public PhysicalObject{
    protected:
        GameParameters& parameters;
        int damage;
        int radius;
        bool waiting_to_explode;
        int time_to_explode;
        float angle;
        float power;
        int shooter_id;
        WeaponExplodeTime explode_time;
        RayCastWeaponExploded explosion;

        virtual void createFixtures() override;
        virtual void setInitialVelocity() override;

        //Ataca a los gusanos en el radio de explosion
        void attackWormExplosion(const b2Vec2& center, int angle);

    public:
        static int weapon_id;

        Weapon(World& world, GameParameters& parameters, int damage, int radius = 0);
        virtual ~Weapon();

        //Devuelve true si el arma esta en movimiento o esperando para explotar
        bool isActive() override;

        //Carga los datos para disparar el arma
        virtual void shoot(char dir, int angle, int power, int time, int shooter_id);

        //Dispara un arma teledirigida
        virtual void shoot(Worm& shooter, b2Vec2 pos);

        //Explota el arma
        virtual void explode();

        virtual void collideWithSomething(CollisionData *other) override;

        void getBodyDef(b2BodyDef& body_def, const b2Vec2& pos) override;

        virtual const std::string& getName() = 0;

        int getShooterId() const;

        void removeShooterId();

};

#endif
```

Jun 06, 18 21:34 **Worm.cpp** Page 1/4

```
#include "Worm.h"
#include "b2CircleShape.h"
#include "b2PolygonShape.h"
#include "b2Fixture.h"
#include "Protocol.h"
#include "WeaponFactory.h"
#include "Girder.h"
#include "Math.h"
#include <algorithm>

Worm::Worm(World& world, GameParameters& parameters, int id, int player_id):
    PhysicalObject(world, id, TYPE_WORM), player_id(player_id), life(parameters.getWormLife()),
    dir(1), parameters(parameters), max_height(0), colliding_with_girder(0), friction(0),
    movement_allowed(false), angle(0){
        this->changeWeapon(DEFAULT_WEAPON);
    }

Worm::~Worm(){}

void Worm::getBodyDef(b2BodyDef& body_def, const b2Vec2& pos){
    body_def.type = b2_dynamicBody;
    body_def.position.Set(pos.x, pos.y);
}

void Worm::createFixtures(){
    b2CircleShape circleShape;
    circleShape.m_p.Set(0, 0);
    circleShape.m_radius = worm_size / 2;

    b2FixtureDef fixtureDef;
    fixtureDef.shape = &circleShape;
    fixtureDef.density = 10;
    this->body->CreateFixture(&fixtureDef);
    this->body->SetFixedRotation(true);

    //Sensor para colisiones
    b2PolygonShape sensorShape;
    sensorShape.SetAsBox(worm_size * 0.5 * 0.7, worm_size / 5, b2Vec2(0, -1 * worm_size / 2), 0);

    b2FixtureDef sensorFixtureDef;
    sensorFixtureDef.shape = &sensorShape;
    sensorFixtureDef.isSensor = true;
    this->body->CreateFixture(&sensorFixtureDef);
}

int Worm::getPlayerId() const{
    return this->player_id;
}

int Worm::getLife() const{
    return this->life;
}

char Worm::getDir() const{
    return this->dir;
}

bool Worm::isColliding() const{
    return this->colliding_with_girder && !this->movement_allowed;
}

const std::string& Worm::getCurrentWeapon() const{
    return ((Weapon*)weapon.get())->getName();
```

```
64  }
65
66  void Worm::addLife(int life){
67      this->life += life;
68  }
69
70  void Worm::reduceLife(int damage){
71      this->life -= damage;
72      this->data_updated = true;
73      if (this->life <= 0){
74          this->life = 0;
75          this->is_dead = true;
76      }
77  }
78
79  bool Worm::move(char action){
80      if (!this->friction){
81          return false;
82      }
83      if (action == MOVE_RIGHT){
84          this->dir = action;
85          b2Vec2 velocity(parameters.getWormVelocity(), 0);
86          this->world.setLinearVelocity(*this, velocity);
87      } else if (action == MOVE_LEFT){
88          this->dir = action;
89          b2Vec2 velocity(-1 * parameters.getWormVelocity(), 0);
90          this->world.setLinearVelocity(*this, velocity);
91      } else {
92
93          this->movement_allowed = true;
94          if (action == JUMP){
95              b2Vec2 velocity(parameters.getWormJumpVelocity(), parameters.getWorm
    JumpHeight());
96              velocity.x *= this->dir;
97              this->world.setLinearVelocity(*this, velocity);
98          } else if (action == ROLLBACK){
99              b2Vec2 velocity(parameters.getWormRollbackVelocity(), parameters.get
    WormRollbackHeight());
100             velocity.x *= -1 * this->dir;
101             this->world.setLinearVelocity(*this, velocity);
102         }
103     }
104     return true;
105 }
106
107 void Worm::changeWeapon(const std::string& weapon){
108     this->weapon.reset();
109     WeaponFactory factory(this->world, this->parameters);
110     this->weapon = factory.getWeapon(weapon);
111 }
112
113 void Worm::shoot(int angle, int power, int time){
114     b2Vec2 pos = this->getPosition();
115     int shooter_id = this->id;
116     float x_add = (worm_size * this->dir);;
117     float y_add = worm_size;
118     if (angle > MAX_WEAPON_ANGLE){
119         shooter_id = -1;
120         x_add *= Math::cosDegrees(this->angle);
121         y_add *= Math::sinDegrees(this->angle);
122     } else {
123         float factor = (this->getCurrentWeapon() == BAT_NAME ? 0.2 : 0.7);
124         x_add *= Math::cosDegrees(angle) * factor;
125         y_add *= Math::sinDegrees(angle) * factor;
126     }
127
```



```
128     pos.x += x_add;
129     pos.y += y_add;
130
131     ((Weapon*)this->weapon.get())->shoot(this->dir, angle, power, time, shooter_
    id);
132     this->world.addObject(this->weapon, pos);
133 }
134
135 void Worm::shoot(b2Vec2 pos){
136     ((Weapon*)this->weapon.get())->shoot(*this, pos);
137 }
138
139 void Worm::receiveWeaponDamage(int damage, const b2Vec2 &epicenter){
140     this->reduceLife(damage);
141     b2Vec2 direction = this->body->GetPosition() - epicenter;
142     direction.Normalize();
143     this->body->SetGravityScale(1);
144     this->movement_allowed = true;
145     this->body->SetLinearVelocity(damage * parameters.getWormExplosionVelocity()
     * direction);
146 }
147
148 void Worm::collideWithSomething(CollisionData *other){
149     if (other->getType() == TYPE_BORDER){
150         this->kill();
151     } else if(other->getType() == TYPE_GIRDER){
152         int min_height = parameters.getWormHeightToDamage();
153         float current_height = this->body->GetPosition().y;
154         this->max_height -= current_height;
155
156         if (this->max_height >= min_height){
157             this->reduceLife(std::min((int) this->max_height - min_height, param
    eters.getWormMaxHeightDamage()));
158         }
159         this->max_height = 0;
160         this->colliding_with_girder ++;
161         Girder* girder = (Girder*)other->getObject();
162         if (girder->hasFriction()){
163             this->friction++;
164             this->movement_allowed = false;
165             this->angle = girder->getAngle();
166         }
167     }
168 }
169
170 void Worm::endCollissionGirder(char has_friction){
171     this->friction -= has_friction;
172     this->colliding_with_girder --;
173     if (this->friction <= 0){
174         this->friction = 0;
175         this->body->SetGravityScale(1);
176         this->angle = 0;
177     }
178 }
179
180 bool Worm::isActive(){
181     if (!this->colliding_with_girder){
182         float height = this->body->GetPosition().y;
183         this->max_height = std::max(this->max_height, height);
184     } else if (this->friction && !this->movement_allowed){
185         this->body->SetGravityScale(0);
186         this->body->SetLinearVelocity(b2Vec2(0, 0));
187     }
188     if (!this->body->IsAwake()){
189         this->movement_allowed = false;
190     }
```

Jun 06, 18 21:34 **Worm.cpp** Page 4/4

```
191     return PhysicalObject::isActive();
192 }
```

Jun 06, 18 21:00 **Worm.h** Page 1/1

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

```cpp
#include "DataSender.h"

DataSender::DataSender(World& world, std::vector<Player>& players, GameParameters& parameters):
    objects(world.getObjectsList()), girders(world.getGirdersList()),
    players(players), mutex(world.getMutex()), active(false), sleep_time(parameters.getDataSenderSleep()){

        for (size_t i = 0; i < this->players.size(); i++){
            std::unique_ptr<PlayerDataSender> sender(new PlayerDataSender(this->players[i]));
            this->players_data_senders.push_back(std::move(sender));
            this->players_data_senders[i]->start();
        }
    }

DataSender::~DataSender(){
    for (size_t i = 0; i < this->players.size(); i++){
        this->players_data_senders[i]->stop();
        this->players_data_senders[i]->join();
    }
}

void DataSender::run(){
    while(this->running){
        std::this_thread::sleep_for(std::chrono::milliseconds(this->sleep_time));
        std::lock_guard<std::mutex> lock(this->mutex);
        this->active = false;
        auto it = this->objects.begin();

        while(it != this->objects.end()){
            if ((*it)->isDead() && !(*it)->getBody()){
                Buffer data = ServerProtocol::sendDeadObject(*it);

                this->sendBuffer(data);
                it = this->objects.erase(it);
                continue;
            }

            if ((*it)->isMoving()){
                Buffer data = ServerProtocol::sendObject(*it);
                this->sendBuffer(data);
                this->active = true;
            }
            ++it;
        }

        if (!this->active){
            //Chequeo que no se hayan desconectado
            Buffer empty_data;
            empty_data.setNext(NO_SEND_DATA);
            this->sendBuffer(empty_data);
        }

        this->notifyAll();
        this->checkPlayers();
    }
}

void DataSender::sendBackgroundImage(File& image){
    Buffer data = ServerProtocol::sendFile(image);
    this->sendBuffer(data);
    this->notifyAll();
}
```



```cpp
void DataSender::sendStartGame(){
    Buffer data = ServerProtocol::sendStartGame();
    this->sendBuffer(data);
    this->notifyAll();
}

void DataSender::sendPlayersId(){
    Buffer length = ServerProtocol::sendLengthBuffer(this->players.size());
    this->sendBuffer(length);
    for (auto it = this->players.begin(); it != this->players.end(); ++it){
        Buffer data = ServerProtocol::sendPlayerId(*it);
        this->sendBuffer(data);
    }
    this->notifyAll();
}

void DataSender::sendGirders(){
    Buffer length = ServerProtocol::sendLengthBuffer(this->girders.size());
    this->sendBuffer(length);
    for (auto it = this->girders.begin(); it != this->girders.end(); ++it){
        Buffer data = ServerProtocol::sendGirder(*it);
        this->sendBuffer(data);
    }
    this->notifyAll();
}

void DataSender::sendWeaponsAmmo(std::map<std::string, int>& weapons){
    Buffer length = ServerProtocol::sendLengthBuffer(weapons.size());
    this->sendBuffer(length);
    for (auto it = weapons.begin(); it != weapons.end(); ++it){
        Buffer data = ServerProtocol::sendWeaponAmmo(it->first, it->second);
        this->sendBuffer(data);
    }
    this->notifyAll();
}

void DataSender::sendStartTurn(int worm_id, int player_id, float wind){
    Buffer data = ServerProtocol::sendStartTurn(worm_id, player_id, wind);
    this->sendBuffer(data);
    this->notifyAll();
}

void DataSender::sendWeaponChanged(const std::string &weapon){
    Buffer data = ServerProtocol::sendWeaponChanged(weapon);
    this->sendBuffer(data);
    this->notifyAll();
}

void DataSender::sendWeaponShot(const std::string& weapon){
    Buffer data = ServerProtocol::sendWeaponShot(weapon);
    this->sendBuffer(data);
    this->notifyAll();
}

void DataSender::sendMoveAction(char action){
    if (action == MOVE_RIGHT || action == MOVE_LEFT){
        return;
    }
    Buffer data = ServerProtocol::sendMoveAction(action);
    this->sendBuffer(data);
    this->notifyAll();
}

void DataSender::sendUpdateScope(int angle) {
    Buffer data = ServerProtocol::sendUpdateScope(angle);
    this->sendBuffer(data);
```

Jun 06, 18 20:14 **DataSender.cpp** Page 3/3

```
129     this->notifyAll();
130 }
131
132 void DataSender::sendEndGame(const std::string& winner){
133     Buffer data = ServerProtocol::sendEndGame(winner);
134     this->sendBuffer(data);
135     this->notifyAll();
136 }
137
138 bool DataSender::isActive(){
139     std::lock_guard<std::mutex> lock(this->mutex);
140     return this->active;
141 }
142
143 void DataSender::sendBuffer(const Buffer& buffer){
144     for (size_t i = 0; i < this->players.size(); i++){
145         if (this->players[i].isConnected()){
146             this->players_data_senders[i]->sendData(buffer);
147         }
148     }
149 }
150
151 void DataSender::notifyAll(){
152     for (size_t i = 0; i < this->players.size(); i++){
153         if (this->players[i].isConnected()){
154             this->players_data_senders[i]->notify();
155         }
156     }
157 }
158
159 void DataSender::checkPlayers(){
160     size_t players_connected = 0;
161     for (size_t i = 0; i < this->players.size(); i++){
162         if (this->players[i].isConnected()){
163             players_connected++;
164         }
165     }
166     if (players_connected <= 1 && this->players.size() > 1){ ////////////////////
    ///////////Eliminar playersize>1 por ahora que hay un solo jugador
167         Buffer data = this->players[0].getProtocol().sendEndTurn();
168         this->sendBuffer(data);
169         this->notifyAll();
170     }
171 }
```

Jun 06, 18 20:14 **DataSender.h** Page 1/2

```
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 <list>
11 #include <memory>
12
13 class DataSender: public Thread{
14     private:
15         std::list<physical_object_ptr>& objects;
16         std::list<physical_object_ptr>& girders;
17         std::vector<Player>& players;
18         std::vector<std::unique_ptr<PlayerDataSender>> players_data_senders;
19         std::mutex& mutex;
20         bool active;
21         int sleep_time;
22
23         void sendBuffer(const Buffer& buffer);
24         void notifyAll();
25
26     public:
27         DataSender(World& world, std::vector<Player>& players, GameParameters& p
    arameters);
28         ~DataSender();
29
30         //Envia constantemente los datos de los objetos
31         void run() override;
32
33         //Envia la imagen de fondo
34         void sendBackgroundImage(File& image);
35
36         //Envia los datos de los jugadores
37         void sendPlayersId();
38
39         //Envia los datos de las vigas
40         void sendGirders();
41
42         //Envia las municiones de las armas
43         void sendWeaponsAmmo(std::map<std::string, int>& weapons);
44
45         //Envia que el jugador cambio de arma
46         void sendWeaponChanged(const std::string &weapon);
47
48         //Envia que el gusano actual salto
49         void sendMoveAction(char action);
50
51         //Envia que el jugador cambio el angulo de la mira
52         void sendUpdateScope(int angle);
53
54         //Envia que el jugador disparo un arma
55         void sendWeaponShot(const std::string& weapon);
56
57         //Envia la senial de comienzo del juego
58         void sendStartGame();
59
60         //Envia la senial de que inicia un nuevo turno
61         void sendStartTurn(int worm_id, int player_id, float wind);
62
63         //Envia la senial de que el juego termino
64         void sendEndGame(const std::string& winner);
65
```

Jun 06, 18 20:14 **DataSender.h** Page 2/2

```
66          //Devuelve true si sigue enviando datos
67          bool isActive();
68
69          //Chequea que haya jugadores conectados
70          void checkPlayers();
71  };
72
73
74  #endif
```

May 28, 18 19:58 **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  }
```

May 30, 18 20:03 **PlayerDataSender.h** Page 1/1

```
#ifndef __PLAYERDATASENDER_H__
#define __PLAYERDATASENDER_H__

#include "Thread.h"
#include "Player.h"
#include "Buffer.h"
#include <mutex>
#include <condition_variable>
#include <queue>

//Cola bloqueante para enviar datos a un jugador
class PlayerDataSender: public Thread{
    private:
        std::mutex mutex;
        std::condition_variable condition_variable;
        Player& player;
        std::queue<Buffer> queue;

    public:
        PlayerDataSender(Player& player);

        ~PlayerDataSender();

        //Envia datos al jugador
        void run() override;

        //Agrega un nuevo dato a la cola
        void sendData(Buffer buffer);

        //Notifica que hay nuevos datos
        void notify();

        //Termina el envio de datos
        void stop() override;

};

#endif
```

Jun 06, 18 20:15 **ServerProtocol.cpp** Page 1/4

```
#include "ServerProtocol.h"
#include "Game.h"
#include "Weapon.h"
#include "Girder.h"
#include "ObjectSizes.h"
#include "Player.h"
#include "DataSender.h"
#include <string>

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

ServerProtocol::ServerProtocol(ServerProtocol&& other): Protocol(std::move(other
)) {}

ServerProtocol::~ServerProtocol(){}

Buffer ServerProtocol::sendObject(physical_object_ptr& object){
    Buffer buffer;
    buffer.setNext(MOVING_OBJECT);

    const std::string& type = object->getType();
    if (type == TYPE_WORM){
        ServerProtocol::send_worm(object, buffer);
    } else if (type == TYPE_WEAPON){
        ServerProtocol::send_weapon(object, buffer);
    }
    return std::move(buffer);
}

Buffer ServerProtocol::sendDeadObject(physical_object_ptr& object){
    Buffer buffer;
    buffer.setNext(DEAD_OBJECT);

    const std::string& type = object->getType();
    if (type == TYPE_WORM){
        buffer.setNext(WORM_TYPE);
    } else if (type == TYPE_WEAPON){
        buffer.setNext(WEAPON_TYPE);
    }

    uint32_t id = object->getId();
    ServerProtocol::sendIntBuffer(buffer, id);

    return std::move(buffer);
}

void ServerProtocol::send_worm(physical_object_ptr& object, Buffer& buffer){
    Worm* worm = (Worm*)object.get();
    buffer.setNext(WORM_TYPE);
    int32_t id = worm->getId();

    b2Vec2 position = worm->getPosition();

    ServerProtocol::sendIntBuffer(buffer, id);
    ServerProtocol::sendIntBuffer(buffer, worm->getPlayerId());
    ServerProtocol::sendIntBuffer(buffer, position.x * UNIT_TO_SEND);
    ServerProtocol::sendIntBuffer(buffer, position.y * UNIT_TO_SEND);
    ServerProtocol::sendIntBuffer(buffer, worm->getLife());
    buffer.setNext(worm->getDir());
    buffer.setNext(worm->isColliding());
}

void ServerProtocol::send_weapon(physical_object_ptr& object, Buffer& buffer){
    buffer.setNext(WEAPON_TYPE);
    ServerProtocol::sendIntBuffer(buffer, object->getId());

```

```
66
67      b2Vec2 position = object->getPosition();
68      Weapon* weapon = (Weapon*)object.get();
69      std::string name = weapon->getName();
70
71      ServerProtocol::sendStringBuffer(buffer, name);
72      ServerProtocol::sendIntBuffer(buffer, position.x * UNIT_TO_SEND);
73      ServerProtocol::sendIntBuffer(buffer, position.y * UNIT_TO_SEND);
74  }
75
76  Buffer ServerProtocol::sendStartGame(){
77      Buffer buffer;
78      buffer.setNext(START_GAME_ACTION);
79      return buffer;
80  }
81
82  Buffer ServerProtocol::sendEndTurn(){
83      Buffer buffer;
84      buffer.setNext(END_TURN);
85      return buffer;
86  }
87
88  Buffer ServerProtocol::sendStartTurn(int32_t current_worm_id, int32_t current_pl
    ayer_id, float wind){
89      Buffer buffer;
90      buffer.setNext(START_TURN);
91
92      ServerProtocol::sendIntBuffer(buffer, current_worm_id);
93      ServerProtocol::sendIntBuffer(buffer, current_player_id);
94      ServerProtocol::sendIntBuffer(buffer, wind * UNIT_TO_SEND);
95
96      return buffer;
97  }
98
99  void ServerProtocol::receive(Game& game, DataSender& data_sender) {
100
101     Buffer buffer = std::move(this->receiveBuffer());
102
103     char action = buffer.getNext();
104
105     if (action == END_TURN) {
106         game.endTurn();
107     } else if (action == ACTION) {
108         char worm_action = buffer.getNext();
109         if (worm_action == MOVE_ACTION){
110             char move = buffer.getNext();
111             if (game.getCurrentWorm().move(move)){
112                 data_sender.sendMoveAction(move);
113             }
114         } else if (worm_action == CHANGE_WEAPON_ACTION) {
115             std::string weapon(this->receiveStringBuffer(buffer));
116             data_sender.sendWeaponChanged(weapon);
117             game.getCurrentWorm().changeWeapon(weapon);
118         } else if(worm_action == MOVE_SCOPE) {
119             int32_t angle = this->receiveIntBuffer(buffer);
120             data_sender.sendUpdateScope(angle);
121         } else if (worm_action == SHOOT_WEAPON) {
122             int angle = this->receiveIntBuffer(buffer);
123             int power = this->receiveIntBuffer(buffer);
124             int time = this->receiveIntBuffer(buffer);
125             data_sender.sendWeaponShot(game.getCurrentWorm().getCurrentWeapon())
    ;
126             game.getCurrentWorm().shoot(angle, power, time);
127         } else if(worm_action == SHOOT_SELF_DIRECTED) {
128             int pos_x = this->receiveIntBuffer(buffer) / UNIT_TO_SEND;
129             int pos_y = this->receiveIntBuffer(buffer) / UNIT_TO_SEND;
```



```
130             data_sender.sendWeaponShot(game.getCurrentWorm().getCurrentWeapon())
    ;
131             game.getCurrentWorm().shoot(b2Vec2(pos_x, pos_y));
132         }
133     }
134 }
135
136 Buffer ServerProtocol::sendPlayerId(const Player& player){
137     Buffer buffer;
138     ServerProtocol::sendIntBuffer(buffer, player.getId());
139     ServerProtocol::sendStringBuffer(buffer, player.getName());
140     return buffer;
141 }
142
143 Buffer ServerProtocol::sendGirder(physical_object_ptr& object){
144     Girder* girder = (Girder*)object.get();
145
146     Buffer buffer;
147     ServerProtocol::sendIntBuffer(buffer, girder->getSize());
148
149     b2Vec2 position = object->getPosition();
150     ServerProtocol::sendIntBuffer(buffer, position.x * UNIT_TO_SEND);
151     ServerProtocol::sendIntBuffer(buffer, position.y * UNIT_TO_SEND);
152     ServerProtocol::sendIntBuffer(buffer, girder->getRotation());
153     return buffer;
154 }
155
156 Buffer ServerProtocol::sendWeaponAmmo(const std::string& weapon_name, int ammo){
157     Buffer buffer;
158     ServerProtocol::sendStringBuffer(buffer, weapon_name);
159     ServerProtocol::sendIntBuffer(buffer, ammo);
160     return buffer;
161 }
162
163 Buffer ServerProtocol::sendWeaponChanged(const std::string &weapon){
164     Buffer buffer;
165     buffer.setNext(CHANGE_WEAPON_ACTION);
166     ServerProtocol::sendStringBuffer(buffer, weapon);
167     return buffer;
168 }
169
170 Buffer ServerProtocol::sendWeaponShot(const std::string &weapon){
171     Buffer buffer;
172     buffer.setNext(SHOOT_WEAPON_ACTION);
173     ServerProtocol::sendStringBuffer(buffer, weapon);
174     return buffer;
175 }
176
177 Buffer ServerProtocol::sendMoveAction(char action){
178     Buffer buffer;
179     buffer.setNext(MOVE_ACTION);
180     buffer.setNext(action);
181     return buffer;
182 }
183
184 Buffer ServerProtocol::sendUpdateScope(int angle) {
185     Buffer buffer;
186     buffer.setNext(MOVE_SCOPE);
187     ServerProtocol::sendIntBuffer(buffer, angle);
188     return buffer;
189 }
190
191 Buffer ServerProtocol::sendEndGame(const std::string& winner){
192     Buffer buffer;
193     buffer.setNext(END_GAME);
194     ServerProtocol::sendStringBuffer(buffer, winner);
```

Jun 06, 18 20:15 **ServerProtocol.cpp** Page 4/4

```
195         return buffer;
196     }
```

Jun 06, 18 20:15 **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
9   class Game;
10  class Player;
11  class DataSender;
12
13  class ServerProtocol : public Protocol{
14      private:
15          //Carga los datos del gusano en el buffer
16          static void send_worm(physical_object_ptr& object, Buffer& buffer);
17
18          //Carga los datos del arma en el buffer
19          static void send_weapon(physical_object_ptr& weapon, Buffer& buffer);
20
21      public:
22          ServerProtocol(Socket&& socket);
23          ServerProtocol(ServerProtocol&& other);
24          ~ServerProtocol();
25
26          //Carga un nuevo objeto en el buffer
27          static Buffer sendObject(physical_object_ptr& object);
28
29          //Carga la informacion de un objeto muerto en el buffer
30          static Buffer sendDeadObject(physical_object_ptr& object);
31
32          //Recibe datos de un cliente
33          void receive(Game& game, DataSender& data_sender);
34
35          //Carga la informacion de comienzo de juego
36          static Buffer sendStartGame();
37
38          //Carga la informacion de nuevo turno en el buffer
39          static Buffer sendStartTurn(int32_t current_worm_id, int32_t current_pla
    yer_id, float wind);
40
41          //Carga la informacion de un nuevo jugador en el buffer
42          static Buffer sendPlayerId(const Player& player);
43
44          //Carga la informacion de una viga en el buffer
45          static Buffer sendGirder(physical_object_ptr& girder);
46
47          //Carga la informacion de un arma en el buffer
48          static Buffer sendWeaponAmmo(const std::string& weapon_name, int ammo);
49
50          //Carga la informacion de cambio de arma en el buffer
51          static Buffer sendWeaponChanged(const std::string &weapon);
52
53          //Carga la informacion de arma disparada en el buffer
54          static Buffer sendWeaponShot(const std::string &weapon);
55
56          //Carga la informacion de que el gusano salto
57          static Buffer sendMoveAction(char action);
58
59          //Carga la informacion de cambio de angulo en el buffer
60          static Buffer sendUpdateScope(int angle);
61
62          //Carga la informacion de fin del juego en el buffer
63          static Buffer sendEndGame(const std::string& winner);
64
65          //Carga la informacion de fin del turno
```

Jun 06, 18 20:15 **ServerProtocol.h** Page 2/2

```
66          static Buffer sendEndTurn();
67  };
68
69  #endif
```

May 26, 18 12:13 **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> distribution(this->min_velocity, this-
    >max_velocity);
20      std::uniform_int_distribution<int> direction(-1, 1); //Acepto velocidad 0
21
22      this->velocity = distribution(rng);
23      this->velocity *= direction(rng);
24  }
```

Jun 05, 18 14:07 **Wind.h** Page 1/1

```
#ifndef __WIND_H__
#define __WIND_H__

#include "GameParameters.h"

class Wind{
    private:
        float min_velocity;
        float max_velocity;
        float velocity;

    public:
        Wind(GameParameters& parameters);
        ~Wind();

        //Devuelve la velocidad del viento
        float getVelocity() const;

        //Actualiza la velocidad del viento
        void update();
};


#endif
```

Jun 05, 18 14:07 **World.cpp** Page 1/3

```
#include "World.h"
#include "Weapon.h"
#include "BottomBorder.h"
#include "b2WorldCallbacks.h"
#include "Fragment.h"

World::World(GameParameters& parameters): world(b2Vec2(0, parameters.getGravity()
)),
    wind(parameters), is_active(true),
    sleep_time(parameters.getWorldSleepAfterStep()), time_step(parameters.getWor
ldTimeStep()){

    this->world.SetAllowSleeping(true);
    this->world.SetContinuousPhysics(true);
    this->world.SetContactListener(&this->collision_listener);
    this->world.SetContactFilter(&this->collision_listener);
    this->initialize();
}

World::~World(){}

void World::run(){
    int32 velocityIterations = 8;   //how strongly to correct velocity
    int32 positionIterations = 3;   //how strongly to correct position

    while(this->running){
        std::this_thread::sleep_for(std::chrono::milliseconds(this->sleep_time))
;

        this->addAllFragments();

        std::lock_guard<std::mutex> lock(this->mutex);

        this->world.Step(this->time_step, velocityIterations, positionIterations
);

        this->is_active = false;
        for (auto it = this->objects.begin(); it != this->objects.end(); it++){
            if ((*it)->isDead()){
                this->removeObject(*it);
            } else if ((*it)->isActive()){
                this->is_active = true;
                b2Body* body = (*it)->getBody();
                if (body && (*it)->isWindAffected()){
                    body->ApplyForceToCenter(b2Vec2(this->wind.getVelocity(), 0)
, false);
                }
            }
        }
    }
}

void World::addAllFragments(){
    std::lock_guard<std::mutex> lock(this->mutex);

    for (auto it = this->fragments_to_add.begin(); it != this->fragments_to_add.
end(); it++){
        b2BodyDef body_def;
        b2Vec2 pos = ((Fragment *) it->get())->getShootPosition();
        (*it)->getBodyDef(body_def, pos);
        this->initializeObject(*it, &body_def);
    }
    this->fragments_to_add.clear();
}

bool World::isActive(){
```

```
61      std::lock_guard<std::mutex> lock(this->mutex);
62      return this->is_active;
63  }
64
65  void World::update(){
66      std::lock_guard<std::mutex> lock(this->mutex);
67      this->wind.update();
68  }
69
70  void World::addObject(physical_object_ptr object, const b2Vec2& pos){
71      b2BodyDef body_def;
72      object->getBodyDef(body_def, pos);
73
74      std::lock_guard<std::mutex> lock(this->mutex);
75      this->initializeObject(object, &body_def);
76  }
77
78  void World::initializeObject(physical_object_ptr object, b2BodyDef* body_def){
79      object->initializeBody(this->world.CreateBody(body_def));
80      if (body_def->type != b2_staticBody){
81          this->objects.push_back(object);
82      } else {
83          this->girders.push_back(object);
84      }
85  }
86
87  void World::addWeaponFragment(physical_object_ptr fragment){
88      this->fragments_to_add.push_back(fragment);
89  }
90
91  void World::removeTimedWeapon(Weapon& weapon){
92      b2Body* body = weapon.getBody();
93      if (body){
94          this->world.DestroyBody(body);
95          weapon.destroyBody();
96      }
97  }
98
99  void World::removeObject(physical_object_ptr object){
100     b2Body* body = object->getBody();
101     if (body){
102         this->world.DestroyBody(body);
103         object->destroyBody();
104     }
105 }
106
107 void World::initialize(){
108     physical_object_ptr bottom_border(new BottomBorder(*this));
109     this->addObject(bottom_border, b2Vec2(0, 0));
110 }
111
112 void World::setLinearVelocity(PhysicalObject& object, b2Vec2& velocity){
113     std::lock_guard<std::mutex> lock(this->mutex);
114     b2Body* body = object.getBody();
115     if (body){
116         body->SetGravityScale(1);
117         body->SetLinearVelocity(velocity);
118     }
119 }
120
121 b2Body* World::getClosestObject(RayCastWeaponExploded* callback, b2Vec2 center,
    b2Vec2 end){
122     this->world.RayCast(callback, center, end);
123     return callback->getClosestWorm();
124 }
125
```



```
126 float World::getWind() const{
127     return this->wind.getVelocity();
128 }
129
130 std::list<physical_object_ptr>& World::getObjectsList(){
131     return this->objects;
132 }
133
134 std::list<physical_object_ptr>& World::getGirdersList(){
135     return this->girders;
136 }
137
138 std::mutex& World::getMutex(){
139     return this->mutex;
140 }
```

Jun 05, 18 14:07 **World.h** Page 1/2

```
#ifndef __WORLD_H__
#define __WORLD_H__

#include "Thread.h"
#include "b2World.h"
#include "b2Body.h"
#include "PhysicalObject.h"
#include "CollisionListener.h"
#include "RayCastWeaponExploded.h"
#include "Wind.h"
#include <mutex>
#include <list>

class Weapon;

class World: public Thread{
    private:
        b2World world;
        Wind wind;
        std::mutex mutex;
        CollisionListener collision_listener;
        std::list<physical_object_ptr> objects;
        std::list<physical_object_ptr> girders;
        std::list<physical_object_ptr> fragments_to_add;
        bool is_active;
        int sleep_time;
        float time_step;

        //Inicializa el mundo
        void initialize();

        //Remueve un objeto del mundo
        void removeObject(physical_object_ptr object);

        //Inicializa un objeto recien agregado al mundo
        void initializeObject(physical_object_ptr object, b2BodyDef* body_def);

        //Agrega todos los fragmentos de armas al mundo
        void addAllFragments();

    public:
        World(GameParameters& parameters);
        ~World();

        void run() override;

        //Agrega el objeto al mundo en la posicion indicada
        void addObject(physical_object_ptr object, const b2Vec2& pos);

        //Agrega un fragmento de arma
        void addWeaponFragment(physical_object_ptr fragment);

        //Elimina una arma del mundo
        void removeTimedWeapon(Weapon& weapon);

        //Setea la velocidad de un objeto
        void setLinearVelocity(PhysicalObject& object, b2Vec2& velocity);

        //Devuelve true si alguno de los objetos esta en movimiento
        bool isActive();

        //Actualiza el mundo
        void update();

        //Devuelve la velocidad del viento
        float getWind() const;
```

Jun 05, 18 14:07 **World.h** Page 2/2

```

        //Devuelve el objeto mas cercano entre al centro en la direccion end - center
        b2Body* getClosestObject(RayCastWeaponExploded* callback, b2Vec2 center, b2Vec2 end);

        std::list<physical_object_ptr>& getObjectsList();
        std::list<physical_object_ptr>& getGirdersList();

        std::mutex& getMutex();
};


#endif
```

Jun 06, 18 22:31 **Table of Content** Page 1/2

Table of Contents

1 *ClientHandler.cpp...* sheets 1 to 1 (1) pages 1- 2 86 lines
2 *ClientHandler.h.....* sheets 2 to 2 (1) pages 3- 3 39 lines
3 *GamesList.cpp.......* sheets 2 to 3 (2) pages 4- 5 91 lines
4 *GamesList.h.........* sheets 3 to 3 (1) pages 6- 6 41 lines
5 *MapsList.cpp........* sheets 4 to 4 (1) pages 7- 7 23 lines
6 *MapsList.h.........* sheets 4 to 4 (1) pages 8- 8 17 lines
7 *Server.cpp.........* sheets 5 to 5 (1) pages 9- 9 60 lines
8 *Server.h...........* sheets 5 to 5 (1) pages 10- 10 38 lines
9 *CollisionData.cpp...* sheets 6 to 6 (1) pages 11- 11 16 lines
10 *CollisionData.h.....* sheets 6 to 6 (1) pages 12- 12 24 lines
11 *CollisionListener.cpp* sheets 7 to 7 (1) pages 13- 14 80 lines
12 *CollisionListener.h.* sheets 8 to 8 (1) pages 15- 15 26 lines
13 *RayCastWeaponExploded.cpp* sheets 8 to 8 (1) pages 16- 16 34 lines
14 *RayCastWeaponExploded.h* sheets 9 to 9 (1) pages 17- 17 27 lines
15 *Game.cpp...........* sheets 9 to 10 (2) pages 18- 19 120 lines
16 *Game.h.............* sheets 10 to 10 (1) pages 20- 20 58 lines
17 *GameParameters.cpp..* sheets 11 to 12 (2) pages 21- 24 181 lines
18 *GameParameters.h....* sheets 13 to 13 (1) pages 25- 26 87 lines
19 *Player.cpp.........* sheets 14 to 14 (1) pages 27- 27 62 lines
20 *Player.h...........* sheets 14 to 14 (1) pages 28- 28 61 lines
21 *Turn.cpp...........* sheets 15 to 15 (1) pages 29- 30 87 lines
22 *Turn.h.............* sheets 16 to 16 (1) pages 31- 31 53 lines
23 *WormsList.cpp.......* sheets 16 to 16 (1) pages 32- 32 52 lines
24 *WormsList.h.........* sheets 17 to 17 (1) pages 33- 33 43 lines
25 *main.cpp...........* sheets 17 to 17 (1) pages 34- 34 30 lines
26 *BottomBorder.cpp....* sheets 18 to 18 (1) pages 35- 35 23 lines
27 *BottomBorder.h......* sheets 18 to 18 (1) pages 36- 36 22 lines
28 *Girder.cpp.........* sheets 19 to 19 (1) pages 37- 37 46 lines
29 *Girder.h...........* sheets 19 to 19 (1) pages 38- 38 36 lines
30 *PhysicalObject.cpp..* sheets 20 to 20 (1) pages 39- 40 85 lines
31 *PhysicalObject.h....* sheets 21 to 21 (1) pages 41- 42 74 lines
32 *AirAttack.cpp.......* sheets 22 to 22 (1) pages 43- 43 26 lines
33 *AirAttack.h.........* sheets 22 to 22 (1) pages 44- 44 24 lines
34 *AirAttackMissile.cpp* sheets 23 to 23 (1) pages 45- 45 15 lines
35 *AirAttackMissile.h..* sheets 23 to 23 (1) pages 46- 46 19 lines
36 *Banana.cpp.........* sheets 24 to 24 (1) pages 47- 47 25 lines
37 *Banana.h...........* sheets 24 to 24 (1) pages 48- 48 20 lines
38 *Bat.cpp............* sheets 25 to 25 (1) pages 49- 49 23 lines
39 *Bat.h..............* sheets 25 to 25 (1) pages 50- 50 19 lines
40 *Bazooka.cpp........* sheets 26 to 26 (1) pages 51- 51 15 lines
41 *Bazooka.h..........* sheets 26 to 26 (1) pages 52- 52 18 lines
42 *Dynamite.cpp.......* sheets 27 to 27 (1) pages 53- 53 11 lines
43 *Dynamite.h.........* sheets 27 to 27 (1) pages 54- 54 16 lines
44 *FragmentableWeapon.cpp* sheets 28 to 28 (1) pages 55- 55 24 lines
45 *FragmentableWeapon.h* sheets 28 to 28 (1) pages 56- 56 20 lines
46 *Fragment.cpp........* sheets 29 to 29 (1) pages 57- 57 19 lines
47 *Fragment.h.........* sheets 29 to 29 (1) pages 58- 58 23 lines
48 *GreenGrenade.cpp....* sheets 30 to 30 (1) pages 59- 59 11 lines
49 *GreenGrenade.h......* sheets 30 to 30 (1) pages 60- 60 16 lines
50 *HolyGrenade.cpp.....* sheets 31 to 31 (1) pages 61- 61 11 lines
51 *HolyGrenade.h.......* sheets 31 to 31 (1) pages 62- 62 16 lines
52 *Mortar.cpp.........* sheets 32 to 32 (1) pages 63- 63 15 lines
53 *MortarFragment.cpp..* sheets 32 to 32 (1) pages 64- 64 15 lines
54 *MortarFragment.h....* sheets 33 to 33 (1) pages 65- 65 19 lines
55 *Mortar.h............* sheets 33 to 33 (1) pages 66- 66 18 lines
56 *RedGrenade.cpp......* sheets 34 to 34 (1) pages 67- 67 11 lines
57 *RedGrenadeFragment.cpp* sheets 34 to 34 (1) pages 68- 68 11 lines
58 *RedGrenadeFragment.h* sheets 35 to 35 (1) pages 69- 69 17 lines
59 *RedGrenade.h........* sheets 35 to 35 (1) pages 70- 70 16 lines
60 *Teleportation.cpp...* sheets 36 to 36 (1) pages 71- 71 26 lines
61 *Teleportation.h.....* sheets 36 to 36 (1) pages 72- 72 22 lines
62 *Weapon.cpp.........* sheets 37 to 37 (1) pages 73- 74 108 lines
63 *WeaponExplodeTime.cpp* sheets 38 to 38 (1) pages 75- 75 31 lines
64 *WeaponExplodeTime.h.* sheets 38 to 38 (1) pages 76- 76 28 lines
65 *WeaponFactory.cpp...* sheets 39 to 39 (1) pages 77- 77 54 lines

Jun 06, 18 22:31 **Table of Content** Page 2/2

66 *WeaponFactory.h.....* sheets 39 to 39 (1) pages 78- 78 22 lines
67 *Weapon.h...........* sheets 40 to 40 (1) pages 79- 79 64 lines
68 *Worm.cpp...........* sheets 40 to 42 (3) pages 80- 83 193 lines
69 *Worm.h.............* sheets 42 to 42 (1) pages 84- 84 67 lines
70 *DataSender.cpp.....* sheets 43 to 44 (2) pages 85- 87 172 lines
71 *DataSender.h........* sheets 44 to 45 (2) pages 88- 89 75 lines
72 *PlayerDataSender.cpp* sheets 45 to 45 (1) pages 90- 90 39 lines
73 *PlayerDataSender.h..* sheets 46 to 46 (1) pages 91- 91 39 lines
74 *ServerProtocol.cpp..* sheets 46 to 48 (3) pages 92- 95 197 lines
75 *ServerProtocol.h....* sheets 48 to 49 (2) pages 96- 97 70 lines
76 *Wind.cpp............* sheets 49 to 49 (1) pages 98- 98 25 lines
77 *Wind.h.............* sheets 50 to 50 (1) pages 99- 99 25 lines
78 *World.cpp...........* sheets 50 to 51 (2) pages 100-102 141 lines
79 *World.h............* sheets 52 to 52 (1) pages 103-104 79 lines