```cpp
#include "ClientProtocol.h"
#include <string>
#include "Player.h"
#include "WeaponList.h"
#include "ObjectSizes.h"
#include "ServerFatalError.h"

ClientProtocol::ClientProtocol(Socket&& socket, Gtk::Window& window) :
        Protocol(std::move(socket)), window(window) {}

ClientProtocol::ClientProtocol(ClientProtocol&& other) :
        Protocol(std::move(other)), window(other.window) {}

ClientProtocol::~ClientProtocol() {}

void ClientProtocol::sendMoveAction(char action) {
    Buffer buffer;
    buffer.setNext(ACTION);
    buffer.setNext(MOVE_ACTION);
    buffer.setNext(action);
    this->sendBuffer(buffer);
}

void ClientProtocol::sendChangeWeapon(const std::string& weapon) {
    Buffer buffer;
    buffer.setNext(ACTION);
    buffer.setNext(CHANGE_WEAPON_ACTION);
    this->sendStringBuffer(buffer, weapon);
    this->sendBuffer(buffer);
}

void
ClientProtocol::sendWeaponShoot(int32_t angle, int32_t power, int32_t time) {
    Buffer buffer;
    buffer.setNext(ACTION);
    buffer.setNext(SHOOT_WEAPON);
    this->sendIntBuffer(buffer, angle);
    this->sendIntBuffer(buffer, power);
    this->sendIntBuffer(buffer, time);
    this->sendBuffer(buffer);
}

void ClientProtocol::sendWeaponSelfDirectedShoot(const Position& pos) {
    Buffer buffer;
    buffer.setNext(ACTION);
    buffer.setNext(SHOOT_SELF_DIRECTED);

    this->sendIntBuffer(buffer, pos.getX() * UNIT_TO_SEND);
    this->sendIntBuffer(buffer, pos.getY() * UNIT_TO_SEND);

    this->sendBuffer(buffer);
}

void ClientProtocol::updateScope(int angle) {
    Buffer buffer;
    buffer.setNext(ACTION);
    buffer.setNext(MOVE_SCOPE);

    this->sendIntBuffer(buffer, angle);

    this->sendBuffer(buffer);
}

void ClientProtocol::sendEndGame() {
    Buffer buffer;
    buffer.setNext(END_GAME);
```



```cpp
    this->sendBuffer(buffer);
}

void ClientProtocol::receiveStartGame() {
    Buffer buffer = std::move(this->receiveBuffer());
}

void ClientProtocol::receiveBackgroundImage(WorldView& world) {
    Buffer buffer = std::move(this->receiveBuffer());
    world.setBackgroundImage(buffer);
}

void ClientProtocol::receiveTurnData(Turn& turn) {
    Buffer buffer = std::move(this->receiveBuffer());
    int max_time = this->receiveIntBuffer(buffer);
    int time_after_shoot = this->receiveIntBuffer(buffer);
    turn.setTime(max_time, time_after_shoot);
}

void ClientProtocol::receivePlayers(PlayersList& players_list) {
    Buffer buffer = std::move(this->receiveBuffer());
    int quantity = this->receiveIntBuffer(buffer);

    for (int i = 0; i < quantity; i++) {
        Buffer buffer = std::move(this->receiveBuffer());

        int id = this->receiveIntBuffer(buffer);
        std::string name = this->receiveStringBuffer(buffer);

        players_list.addPlayer(id, name);
    }
}

void ClientProtocol::receiveGirders(ViewsList& viewsList) {
    Buffer buffer = std::move(this->receiveBuffer());
    int quantity = this->receiveIntBuffer(buffer);

    for (int i = 0; i < quantity; i++) {
        Buffer buffer = std::move(this->receiveBuffer());;

        int size = this->receiveIntBuffer(buffer);
        float pos_x = this->receiveIntBuffer(buffer) / UNIT_TO_SEND;
        float pos_y = this->receiveIntBuffer(buffer) / UNIT_TO_SEND;
        int rotation = this->receiveIntBuffer(buffer);
        viewsList.addGirder(size, pos_x, pos_y, rotation);
    }
}

void ClientProtocol::receiveWeaponsAmmo(WeaponList& weapon_list) {
    Buffer buffer = std::move(this->receiveBuffer());
    int quantity = this->receiveIntBuffer(buffer);

    for (int i = 0; i < quantity; i++) {
        Buffer buffer = std::move(this->receiveBuffer());

        std::string name = this->receiveStringBuffer(buffer);
        int ammo = this->receiveIntBuffer(buffer);
        weapon_list.add(name, ammo);
    }
}

void ClientProtocol::sendBuffer(Buffer& buffer) {
    try {
        Protocol::sendBuffer(buffer);
    } catch(const std::exception& e) {
        ServerFatalError error(this->window);
```

jun 10, 18 15:44 **ClientProtocol.cpp** Page 3/3

```
133         }
134 }
```

jun 10, 18 15:47 **ClientProtocol.h** Page 1/2

```
1  #ifndef __CLIENTPROTOCOL_H__
2  #define __CLIENTPROTOCOL_H__
3
4  #include "Socket.h"
5  #include "Protocol.h"
6  #include "Position.h"
7  #include "ViewsList.h"
8  #include "PlayersList.h"
9  #include "Turn.h"
10 #include <gtkmm/window.h>
11 #include <string>
12
13 class Player;
14
15 class WeaponList;
16
17 /* Clase que se encarga de enviar y recibir mensajes del socket
18  * con un formato determinado */
19 class ClientProtocol : public Protocol {
20 private:
21     Gtk::Window& window;
22
23 public:
24     /* Constructor */
25     ClientProtocol(Socket&& socket, Gtk::Window& window);
26
27     /* Constructor por movimiento */
28     ClientProtocol(ClientProtocol&& other);
29
30     /* Destructor */
31     ~ClientProtocol();
32
33     /* Envia un mensaje que indica una accion de movimiento */
34     void sendMoveAction(char action);
35
36     /* Envia un mensaje que indica una accion de cambio de arma
37      * con el nombre del arma */
38     void sendChangeWeapon(const std::string& weapon);
39
40     /* Envia un mensaje de accion de disparo, con el angulo, la potencia
41      * y el tiempo de explosion */
42     void sendWeaponShoot(int32_t angle, int32_t power, int32_t time);
43
44     /* Envia un mensaje de accion de disparo teledirigido con
45      * la posicion del disparo */
46     void sendWeaponSelfDirectedShoot(const Position& pos);
47
48     /* Envia un mesaje que indica el cambio del angulo del scope */
49     void updateScope(int angle);
50
51     /* Envia un mensaje de finalizacion de juego */
52     void sendEndGame();
53
54     /* Recibe el comienzo del juego */
55     void receiveStartGame();
56
57     /* Recibe y setea la imagen de fondo */
58     void receiveBackgroundImage(WorldView& world);
59
60     /* Recibe los datos del turno */
61     void receiveTurnData(Turn& turn);
62
63     /* Recibe los jugadores de la partida junto con su
64      * id y su nombre */
65     void receivePlayers(PlayersList& players_list);
66
```

jun 10, 18 15:47 **ClientProtocol.h** Page 2/2

```
67      /* Recibe la vigas presentes en el mapa junto con su tamaÃ±o,
68       * su posicion y su rotacion */
69      void receiveGirders(ViewsList& viewsList);
70
71      /* Recibe las armas presentes en el juego junto con
72       * su municion */
73      void receiveWeaponsAmmo(WeaponList& weapon_list);
74
75      /* Envia el contenido del buffer */
76      void sendBuffer(Buffer& buffer) override;
77  };
78
79  #endif
```

jun 12, 18 0:20 **DataReceiver.cpp** Page 1/2

```
1   #include "DataReceiver.h"
2   #include "Player.h"
3   #include <glibmm/main.h>
4   #include <string>
5   #include "ObjectSizes.h"
6
7   DataReceiver::DataReceiver(Player& player) :
8           player(player), protocol(player.getProtocol()) {}
9
10  DataReceiver::~DataReceiver() {}
11
12  void DataReceiver::run() {
13      try {
14          this->initialConfig();
15          while (this->running) {
16              Buffer data = this->protocol.receiveBuffer();
17              if (*data.getPointer() == END_GAME) {
18                  this->stop();
19              }
20              sigc::slot<bool> my_slot = sigc::bind(sigc::mem_fun(*this,
21                                          &DataReceiver::analizeReceivedData), data);
22              Glib::signal_idle().connect(my_slot);
23          }
24      } catch(const std::exception& e) {
25          if (this->running) {
26              this->player.getScreen().close();
27          }
28      }
29  }
30
31  void DataReceiver::initialConfig() {
32      this->protocol.receiveStartGame();
33      this->protocol.receiveBackgroundImage(this->player.getScreen().getWorld());
34      this->protocol.receiveTurnData(this->player.getTurn());
35      this->protocol.receivePlayers(this->player.getScreen().getPlayersView());
36      this->protocol.receiveGirders(this->player.getViewsList());
37      this->protocol.receiveWeaponsAmmo(this->player.getWeapons());
38      this->player.getScreen().show();
39  }
40
41  bool DataReceiver::analizeReceivedData(Buffer buffer) {
42      char action = buffer.getNext();
43
44      if (action == START_TURN) {
45          int worm_id = Protocol::receiveIntBuffer(buffer);
46          int player_id = Protocol::receiveIntBuffer(buffer);
47          float wind = Protocol::receiveIntBuffer(buffer) / UNIT_TO_SEND;
48          this->player.startTurn(worm_id, player_id, wind);
49      } else if (action == END_GAME) {
50          std::string winner = Protocol::receiveStringBuffer(buffer);
51          this->player.endGame(winner);
52      } else if (action == END_TURN) {
53          this->player.endTurn();
54      } else if (action == CHANGE_WEAPON_ACTION) {
55          std::string weapon(Protocol::receiveStringBuffer(buffer));
56          this->player.getViewsList().removeScopeVisibility();
57          this->player.getViewsList().changeWeapon(weapon);
58      } else if (action == MOVE_SCOPE) {
59          int angle = Protocol::receiveIntBuffer(buffer);
60          this->player.getViewsList().updateScope(angle);
61      } else if (action == SHOOT_WEAPON_ACTION) {
62          std::string weapon(Protocol::receiveStringBuffer(buffer));
63          this->player.getViewsList().removeScopeVisibility();
64          this->player.getViewsList().shoot(weapon);
65          this->player.getMusicPlayer().playWeaponShotSound(weapon);
66      } else if (action == MOVING_OBJECT) {
```

jun 12, 18 0:20 **DataReceiver.cpp** Page 2/2

```
67          char type = buffer.getNext();
68          int id = Protocol::receiveIntBuffer(buffer);
69
70          if (type == WORM_TYPE) {
71              int player_id = Protocol::receiveIntBuffer(buffer);
72              int pos_x = Protocol::receiveIntBuffer(buffer);
73              int pos_y = Protocol::receiveIntBuffer(buffer);
74              int life = Protocol::receiveIntBuffer(buffer);
75              char dir = buffer.getNext();
76              bool colliding = buffer.getNext();
77              this->player.getViewsList().updateWormData(id, player_id, pos_x,
78                                                         pos_y, life, dir, colliding);
79              this->player.getViewsList().removeScopeVisibility();
80          } else if (type == WEAPON_TYPE) {
81              std::string weapon(Protocol::receiveStringBuffer(buffer));
82
83              int pos_x = Protocol::receiveIntBuffer(buffer);
84              int pos_y = Protocol::receiveIntBuffer(buffer);
85              this->player.getViewsList().updateWeaponData(id, weapon, pos_x,
86                                                           pos_y);
87          }
88      } else if (action == DEAD_OBJECT) {
89          char type = buffer.getNext();
90          int id = Protocol::receiveIntBuffer(buffer);
91          if (type == WORM_TYPE) {
92              this->player.getViewsList().removeWorm(id);
93          } else if (type == WEAPON_TYPE) {
94              this->player.getViewsList().removeWeapon(id);
95          }
96      } else if (action == MOVE_ACTION) {
97          char movement = buffer.getNext();
98          this->player.getMusicPlayer().playJumpSound(movement);
99      }
100     return false;
101 }
```

jun 10, 18 15:44 **DataReceiver.h** Page 1/1

```
1  #ifndef __DATARECEIVER_H__
2  #define __DATARECEIVER_H__
3
4  #include "Thread.h"
5  #include "ClientProtocol.h"
6
7  class Player;
8
9  /* Clase que se encarga de recibir los mensajes
10  * enviados por el servidor */
11 class DataReceiver : public Thread {
12 private:
13     Player& player;
14     ClientProtocol& protocol;
15
16     /* Recibe los datos de la configuracion inicial */
17     void initialConfig();
18
19     /* Analiza los datos recibidos */
20     bool analizeReceivedData(Buffer buffer);
21
22 public:
23     /* Constructor */
24     explicit DataReceiver(Player& player);
25
26     /* Destructor */
27     ~DataReceiver();
28
29     /* Comienza a recibir mensajes del protocolo */
30     void run() override;
31 };
32
33
34 #endif
```

jun 10, 18 15:49 **main.cpp** Page 1/1

```cpp
#include <gtkmm/application.h>
#include <gtkmm/window.h>
#include "ServerMenu.h"
#include "Path.h"

int main(int argc, char* argv[]) {
    auto app = Gtk::Application::create(argc, argv);
    Gtk::Window window;
    window.maximize();

    window.set_title(CLIENT_WINDOW_NAME);

    window.set_icon_from_file(ICON_PATH);

    ServerMenu server_menu(window);

    app->run(window);

    return 0;
}
```

jun 10, 18 15:10 **ButtonBuilder.cpp** Page 1/1

```cpp
#include "ButtonBuilder.h"
#include <string>
#include <gtkmm/label.h>
#include <gdkmm/rgba.h>

void ButtonBuilder::buildButton(Gtk::Button* button) {
    std::string text = button->get_label();
    Gtk::Label* label = (Gtk::Label*) button->get_child();
    label->set_markup("<b>" + text + "</b>");
    label->override_color(Gdk::RGBA("black"));
}
```

jun 10, 18 14:54 **ButtonBuilder.h** Page 1/1

```h
#ifndef WORMS_BUTTONBUILDER_H
#define WORMS_BUTTONBUILDER_H

#include <gtkmm/button.h>

class ButtonBuilder {
public:
    /* Modifica la visualizaciÃ³n del label del boton */
    static void buildButton(Gtk::Button* button);
};


#endif //WORMS_BUTTONBUILDER_H
```

jun 20, 18 18:42 **CreateGameMenu.cpp** Page 1/1

```cpp
#include "CreateGameMenu.h"
#include <string>
#include "Path.h"
#include "GamePlayers.h"

const std::string PATH = GLADE_PATH + "client_CreateGameMenu.glade";

CreateGameMenu::CreateGameMenu(Gtk::Window& window, MenuView& first_menu,
                ClientProtocol& protocol, std::string&& name, int quantity) :
    SelectableListMenu(window, first_menu, protocol, std::move(name), PATH) {
    this->builder->get_widget("game_name", this->game_name);
    this->builder->get_widget("players_number", this->players_number);
    this->builder->get_widget("games", this->games);

    this->configure(quantity);

    this->builder->get_widget("create_game_menu", this->menu);

    this->addMenu();
}

CreateGameMenu::~CreateGameMenu() {}

void CreateGameMenu::selectButtonPressed(Glib::ustring map_name) {
    std::string name(this->game_name->get_text());
    if (name.empty()) {
        this->showError("Debe ingresar el nombre de la partida");
        return;
    }

    size_t players = this->players_number->get_value_as_int();
    if (players < min_players || players > max_players) {
        std::string message("El numero de jugadores debe estar entre ");
        message += std::to_string(min_players) + std::string(" y ");
        message += std::to_string(max_players);
        this->showError(message);
        return;
    }

    try {
        this->protocol.sendString(map_name);
        this->protocol.sendString(name);
        this->protocol.sendLength(players);
        bool result = this->protocol.receiveChar();
        if (!result) {
            this->showErrorAndRestart("Ocurrio un error al crear la partida");
        } else {
            this->waitToPlayers();
        }
    } catch(const SocketException& e) {
        this->showFatalError();
    }
}
```

jun 10, 18 15:14 **CreateGameMenu.h** Page 1/1

```
#ifndef __CREATEGAMEMENU__
#define __CREATEGAMEMENU__

#include <gtkmm/entry.h>
#include <gtkmm/spinbutton.h>
#include <string>
#include "SelectableListMenu.h"

/* Clase que se encarga de los pasos necesarios para que el
 * jugador cree una partida */
class CreateGameMenu : public SelectableListMenu {
private:
    Gtk::Entry* game_name;
    Gtk::SpinButton* players_number;

    /* Handler del boton de seleccion */
    void selectButtonPressed(Glib::ustring map_name) override;

public:
    /* Constructor */
    CreateGameMenu(Gtk::Window& window, MenuView& first_menu,
                   ClientProtocol& protocol, std::string&& name, int quantity);

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

#endif
```

jun 20, 18 18:42 **GameMenu.cpp** Page 1/2

```
#include "GameMenu.h"
#include <string>
#include "Path.h"
#include "CreateGameMenu.h"
#include "JoinGameMenu.h"
#include "ButtonBuilder.h"

const std::string PATH = GLADE_PATH + "client_GameMenu.glade";

GameMenu::GameMenu(Gtk::Window& window, ClientProtocol& protocol) :
                                        MenuView(window, *this, protocol, PATH) {
    this->builder->get_widget("player_name", this->player_name);

    this->builder->get_widget("game_menu", this->menu);

    this->addMenu();

    Gtk::Button* create_game, * join_game;

    this->builder->get_widget("create_game", create_game);
    this->builder->get_widget("join_game", join_game);

    ButtonBuilder::buildButton(create_game);
    ButtonBuilder::buildButton(join_game);

    create_game->signal_clicked().connect(sigc::mem_fun(*this,
                                            &GameMenu::createButtonPressed));
    join_game->signal_clicked().connect(sigc::mem_fun(*this,
                                            &GameMenu::joinButtonPressed));
}

GameMenu::~GameMenu() {}

void GameMenu::createButtonPressed() {
    if (this->selectAction(CREATE_GAME_ACTION)) {
        std::string name(this->player_name->get_text());
        int quantity = this->protocol.receiveLength();
        if (quantity == 0) {
            this->showErrorAndRestart("No hay mapas para crear una partida");
        } else {
            this->hideWarningBox();
            this->next_menu = std::unique_ptr<MenuView>(
                    new CreateGameMenu(this->window, *this, this->protocol,
                                       std::move(name), quantity));
        }
    }
}

void GameMenu::joinButtonPressed() {
    if (this->selectAction(JOIN_GAME_ACTION)) {
        std::string name(this->player_name->get_text());
        int quantity = this->protocol.receiveLength();
        if (quantity == 0) {
            this->showErrorAndRestart("No hay partidas disponibles");
        } else {
            this->hideWarningBox();
            this->next_menu = std::unique_ptr<MenuView>(
                    new JoinGameMenu(this->window, *this, this->protocol,
                                     std::move(name), quantity));
        }
    }
}

bool GameMenu::selectAction(char action) {
    std::string name(this->player_name->get_text());
    if (name.empty()) {
```

jun 20, 18 18:42 **GameMenu.cpp** Page 2/2

```
67              this->showError("Debe ingresar su nombre");
68              return false;
69          }
70          try {
71              this->protocol.sendChar(action);
72              this->protocol.sendString(name);
73              this->window.remove();
74              return true;
75          } catch(const SocketException& e) {
76              this->showFatalError();
77              return false;
78          }
79      }
```

jun 10, 18 15:19 **GameMenuField.cpp** Page 1/1

```
1   #include "GameMenuField.h"
2   #include <gdkmm/rgba.h>
3   #include <string>
4   #include "Path.h"
5   #include "ButtonBuilder.h"
6
7   GameMenuField::GameMenuField(const std::string& title) : container(true, 20) {
8       size_t extension = title.rfind(YAML_EXTENSION);
9       this->title.set_markup(title.substr(0, extension));
10      this->title.override_color(Gdk::RGBA("black"));
11      this->container.pack_start(this->title);
12      this->container.pack_end(this->button);
13
14      this->button.set_label("Seleccionar");
15      ButtonBuilder::buildButton(&this->button);
16  }
17
18  GameMenuField::~GameMenuField() {}
19
20  GameMenuField::GameMenuField(GameMenuField&& other) :
21          title(std::move(other.title)), button(std::move(other.button)),
22          container(std::move(other.container)) {}
23
24  Gtk::Container& GameMenuField::getContainer() {
25      return this->container;
26  }
27
28  Gtk::Button& GameMenuField::getButton() {
29      return this->button;
30  }
```

jun 10, 18 15:16 **GameMenuField.h** Page 1/1

```h
#ifndef __GAMEMENUFIELD_H__
#define __GAMEMENUFIELD_H__

#include <gtkmm/hvbox.h>
#include <gtkmm/label.h>
#include <gtkmm/button.h>
#include <string>

class GameMenuField {
private:
    Gtk::Label title;
    Gtk::Button button;
    Gtk::HBox container;

public:
    /* Constructor */
    explicit GameMenuField(const std::string& title);

    /* Destructor */
    ~GameMenuField();

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


    /* Devuelve el contenedor del menu */
    Gtk::Container& getContainer();

    /* Devuelve el boton del menu */
    Gtk::Button& getButton();
};


#endif
```

jun 10, 18 14:54 **GameMenu.h** Page 1/1

```h
#ifndef __GAMEMENU__
#define __GAMEMENU__

#include <gtkmm/entry.h>
#include <string>
#include <memory>
#include "ClientProtocol.h"
#include "MenuView.h"

/* Clase que se encarga de controlar el menu del juego */
class GameMenu : public MenuView {
private:
    Gtk::Entry* player_name;

    /* Crea el boton de creacion de partida */
    void createButtonPressed();

    /* Crea el boton de unirse a partida */
    void joinButtonPressed();

    /* Envia la accion implementada */
    bool selectAction(char action);

public:
    /* Constructor */
    GameMenu(Gtk::Window& window, ClientProtocol& protocol);

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

#endif
```

jun 21, 18 12:41 **JoinGameMenu.cpp** Page 1/1

```cpp
#include "JoinGameMenu.h"
#include <string>
#include "Path.h"
#include "WaitingRoom.h"

const std::string PATH = GLADE_PATH + "client_JoinGameMenu.glade";

JoinGameMenu::JoinGameMenu(Gtk::Window& window, MenuView& first_menu,
                ClientProtocol& protocol, std::string&& name, int quantity) :
    SelectableListMenu(window, first_menu, protocol, std::move(name), PATH) {
    this->builder->get_widget("games", this->games);

    this->configure(quantity);

    this->builder->get_widget("join_game_menu", this->menu);

    this->addMenu();
}

JoinGameMenu::~JoinGameMenu() {}


void JoinGameMenu::selectButtonPressed(Glib::ustring game_name) {
    try {
        this->protocol.sendString(game_name);
        bool result = this->protocol.receiveChar();
        if (!result) {
            this->showErrorAndRestart(
                    "Ocurrio un error al unirse a la partida");
        } else {
            this->waitToPlayers();
        }
    } catch(const SocketException& e) {
        this->showFatalError();
    }
}
```

jun 10, 18 15:21 **JoinGameMenu.h** Page 1/1

```cpp
#ifndef __JOINGAMEMENU__
#define __JOINGAMEMENU__

#include <string>
#include "SelectableListMenu.h"

/* Clase que se encarga de los pasos necesarios para que el
 * jugador se una a una partida */
class JoinGameMenu : public SelectableListMenu {
private:
    /* Handler del boton de unirse a partida */
    void selectButtonPressed(Glib::ustring game_name) override;

public:
    /* Constructor */
    JoinGameMenu(Gtk::Window& window, MenuView& first_menu,
                 ClientProtocol& protocol, std::string&& name, int quantity);

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

#endif
```

jun 20, 18 18:42 **Menu.cpp** Page 1/1

```cpp
#include "Menu.h"
#include <string>
#include "Path.h"
#include "ButtonBuilder.h"

Menu::Menu(const std::string& path, Gtk::Window& window) : window(window) {
    this->builder = Gtk::Builder::create_from_file(path);

    this->builder->get_widget("error", this->error);

    this->builder->get_widget("quit_game", this->quit);

    ButtonBuilder::buildButton(this->quit);

    this->builder->get_widget("title", this->title);
    this->title->set(TITLE_MENU_IMAGE);

    this->builder->get_widget("background", this->background);
    this->background->set(BACKGROUND_MENU_IMAGE);

    this->builder->get_widget("warning1", this->warning1);
    this->warning1->set(WARNING_IMAGE);

    this->builder->get_widget("warning2", this->warning2);
    this->warning2->set(WARNING_IMAGE);

    this->quit->signal_clicked().connect(
            sigc::mem_fun(*this, &Menu::quitButtonPressed));

    this->builder->get_widget("warning_box", this->warning_box);
}

Menu::~Menu() {}

void Menu::quitButtonPressed() {
    this->window.close();
}

void Menu::hideWarningBox() {
    this->error->hide();
    this->warning1->hide();
    this->warning2->hide();
}

void Menu::showError(const std::string& error) {
    if (error.empty()) {
        this->hideWarningBox();
    } else {
        this->error->set_label(error);
        this->warning_box->show_all();
    }
}
```

jun 20, 18 15:36 **Menu.h** Page 1/1

```cpp
#ifndef WORMS_MENU_H
#define WORMS_MENU_H

#include <gtkmm/button.h>
#include <gtkmm/label.h>
#include <gtkmm/window.h>
#include <gtkmm/image.h>
#include <gtkmm/builder.h>
#include <gtkmm/box.h>
#include <string>

class Menu {
protected:
    Gtk::Label* error;
    Gtk::Button* quit;
    Gtk::Window& window;
    Gtk::Image* title;
    Gtk::Image* background;
    Gtk::Image* warning1;
    Gtk::Image* warning2;
    Gtk::Box* warning_box;
    Glib::RefPtr<Gtk::Builder> builder;

    /* Handler del boton de salir */
    void quitButtonPressed();

    /* Muestra un mensaje de error */
    void showError(const std::string& error);

    /* Oculta el mensaje de error */
    void hideWarningBox();

public:
    /* Constructor */
    Menu(const std::string& path, Gtk::Window& window);

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


#endif //WORMS_MENU_H
```

jun 20, 18 18:42 **MenuView.cpp** Page 1/1

```cpp
#include "MenuView.h"
#include <string>
#include "ServerFatalError.h"

MenuView::MenuView(Gtk::Window& window, MenuView& main_menu,
                   ClientProtocol& protocol, const std::string& path) :
        Menu(path, window), protocol(protocol), main_menu(main_menu) {}

MenuView::~MenuView() {
    delete this->menu;
}

void MenuView::showFatalError() {
    ServerFatalError error(this->window);
}

void MenuView::showErrorAndRestart(const std::string& error) {
    this->window.remove();
    this->main_menu.showError(error);
    this->window.add(*this->main_menu.menu);
}

void MenuView::addMenu() {
    this->window.add(*this->menu);
    this->window.show_all();
    this->hideWarningBox();
}
```

jun 20, 18 18:42 **MenuView.h** Page 1/1

```cpp
#ifndef __MENUVIEW_H__
#define __MENUVIEW_H__

#include <gtkmm/container.h>
#include <memory>
#include <string>
#include "ClientProtocol.h"
#include "Menu.h"

class MenuView : public Menu {
protected:
    std::unique_ptr<MenuView> next_menu;
    ClientProtocol& protocol;
    MenuView& main_menu;
    Gtk::Container* menu;

    /* Muestra un mensaje de error y cierra la aplicacion*/
    void showFatalError();

    /* Muestra un mensaje de error y reinicia */
    void showErrorAndRestart(const std::string& error);

public:
    /* Constructor */
    MenuView(Gtk::Window& window, MenuView& main_menu, ClientProtocol& protocol,
             const std::string& path);

    /* Destructor */
    virtual ~MenuView();

    /* Agrega el menu al container y el container al window */
    void addMenu();
};

#endif
```

jun 22, 18 12:22 **SelectableListMenu.cpp** Page 1/2

```cpp
#include "SelectableListMenu.h"
#include <string>
#include "ButtonBuilder.h"

SelectableListMenu::SelectableListMenu(Gtk::Window& window,
                                       MenuView& first_menu,
                                       ClientProtocol& protocol,
                                       std::string&& name,
                                       const std::string& path) :
        MenuView(window, first_menu, protocol, path),
        player_name(std::move(name)) {
    this->builder->get_widget("turn_back", this->turn_back);
    ButtonBuilder::buildButton(this->turn_back);
    this->turn_back->signal_clicked().connect(
            sigc::mem_fun(*this, &SelectableListMenu::turnBackButtonPressed));
}

SelectableListMenu::~SelectableListMenu() {}

void SelectableListMenu::turnBackButtonPressed() {
    std::string string;
    try {
        this->protocol.sendString(string);
        this->showErrorAndRestart(string);
    } catch(const std::exception& e) {
        this->showFatalError();
    }
}

void SelectableListMenu::configure(int quantity) {
    try {
        for (int i = 0; i < quantity; i++) {
            std::string field = this->protocol.receiveString();
            this->addField(field);
        }
    } catch(const SocketException& e) {
        this->showFatalError();
    }

    for (auto it = this->fields.begin(); it != this->fields.end(); ++it) {
        this->games->pack_start(it->getContainer());
    }
    this->games->show();
}

void SelectableListMenu::addField(const std::string& field_name) {
    GameMenuField field(field_name);
    this->fields.push_back(std::move(field));
    this->fields.back().getButton().signal_clicked().connect(
            sigc::bind<Glib::ustring>(sigc::mem_fun(*this,
                        &SelectableListMenu::selectButtonPressed), field_name));
}

bool SelectableListMenu::createPlayer() {
    try {
        this->player = std::unique_ptr<Player>(
                new Player(this->protocol, this->player_name, this->window,
                           this->main_menu));
    } catch(const std::exception& e) {
        this->showFatalError();
    }
    return false;
}

void SelectableListMenu::waitToPlayers() {
    this->window.remove();
```

jun 22, 18 12:22 **SelectableListMenu.cpp** Page 2/2

```cpp
    this->window.add(this->waiting_room.getWidget());
    this->window.show_all();
    sigc::slot<bool> my_slot = sigc::mem_fun(*this,
                                             &SelectableListMenu::createPlayer);
    Glib::signal_idle().connect(my_slot);
}
```

jun 22, 18 12:22 **SelectableListMenu.h** Page 1/1

```
#ifndef __SELECTABLELISTMENU_H__
#define __SELECTABLELISTMENU_H__

#include <gtkmm/box.h>
#include <gtkmm/button.h>
#include <memory>
#include <string>
#include <vector>
#include "ClientProtocol.h"
#include "MenuView.h"
#include "WaitingRoom.h"
#include "Player.h"
#include "GameMenuField.h"

class SelectableListMenu : public MenuView {
protected:
    Gtk::Box* games;
    std::string player_name;
    WaitingRoom waiting_room;
    std::vector<GameMenuField> fields;
    std::unique_ptr<Player> player;
    Gtk::Button* turn_back;

    /* Realiza la configuracion del juego */
    void configure(int quantity);

    /* Agrega un campo a la lista */
    void addField(const std::string& field_name);

    /* Crea un nuevo jugador */
    bool createPlayer();

    /* Handler del boton de seleccion */
    virtual void selectButtonPressed(Glib::ustring field_name) = 0;

    /* Handler del boton volver */
    void turnBackButtonPressed();

    /* Muestra el mensaje esperando jugadores */
    void waitToPlayers();

public:
    /* Constructor */
    SelectableListMenu(Gtk::Window& window, MenuView& first_menu,
                       ClientProtocol& protocol, std::string&& name,
                       const std::string& path);

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

#endif
```

jun 10, 18 15:17 **ServerFatalError.cpp** Page 1/1

```
#include "ServerFatalError.h"
#include <gtkmm/messagedialog.h>
#include <string>

const std::string ERROR_MSG("Ocurrio un error con la conexion del servidor");

ServerFatalError::ServerFatalError(Gtk::Window& window) {
    Gtk::MessageDialog dialog(window, ERROR_MSG, false, Gtk::MESSAGE_ERROR,
                              Gtk::BUTTONS_CLOSE, true);
    dialog.run();
    window.close();
}

ServerFatalError::~ServerFatalError() {}
```

jun 10, 18 15:14 **ServerFatalError.h** Page 1/1

```h
#ifndef __SERVERFATALERROR_H__
#define __SERVERFATALERROR_H__

#include <gtkmm/window.h>

/* Clase que se encarga de mostrar un error fatal
 * con la conexiÃ³n entre el servidor y el cliente */
class ServerFatalError {
public:
    /* Constructor */
    explicit ServerFatalError(Gtk::Window& window);

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

#endif
```

jun 20, 18 15:36 **ServerMenu.cpp** Page 1/1

```cpp
#include "ServerMenu.h"
#include <string>
#include "Path.h"
#include "ButtonBuilder.h"

const std::string PATH = GLADE_PATH + "client_ServerMenu.glade";

ServerMenu::ServerMenu(Gtk::Window& window) : Menu(PATH, window) {
    this->builder->get_widget("host", this->host);
    this->builder->get_widget("service", this->service);
    this->builder->get_widget("connect", this->connect);

    ButtonBuilder::buildButton(this->connect);

    this->builder->get_widget("server_menu", this->menu);

    this->window.add(*this->menu);
    this->window.show_all();

    this->hideWarningBox();

    this->connect->signal_clicked().connect(
            sigc::mem_fun(*this, &ServerMenu::connectButtonPressed));
}

ServerMenu::~ServerMenu() {
    delete this->menu;
}

void ServerMenu::connectButtonPressed() {
    std::string host(this->host->get_text());
    if (host.empty()) {
        this->showError("Debe ingresar un host");
        return;
    }

    std::string service(this->service->get_text());
    if (service.empty()) {
        this->showError("Debe ingresar un servicio");
        return;
    }

    this->connectToServer(host, service);
}

void ServerMenu::connectToServer(const std::string& host,
                                 const std::string& service) {
    try {
        Socket socket(Socket::Client(host.c_str(), service.c_str()));
        this->protocol.reset(
                new ClientProtocol(std::move(socket), this->window));
        this->window.remove();
        this->next_menu = std::unique_ptr<MenuView>(
                new GameMenu(this->window, *this->protocol));
    } catch(const SocketException& e) {
        this->showError("No pudo conectarse al servidor");
    }
}
```

jun 10, 18 15:20 **ServerMenu.h** Page 1/1

```
#ifndef __SERVERMENU__
#define __SERVERMENU__

#include <gtkmm/button.h>
#include <gtkmm/entry.h>
#include <string>
#include <memory>
#include "ClientProtocol.h"
#include "GameMenu.h"
#include "MenuView.h"
#include "Menu.h"

/* Menu de conexion con el servidor */
class ServerMenu : public Menu {
private:
    Gtk::Entry* host;
    Gtk::Entry* service;
    Gtk::Button* connect;
    Gtk::Container* menu;
    std::unique_ptr<MenuView> next_menu;
    std::unique_ptr<ClientProtocol> protocol;

    /* Handler del boton de conexion */
    void connectButtonPressed();

    /* Intenta realizar una conexion con el servidor */
    void connectToServer(const std::string& host, const std::string& service);

public:
    /* Constructor */
    explicit ServerMenu(Gtk::Window& window);

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

#endif
```

jun 22, 18 13:50 **WaitingRoom.cpp** Page 1/1

```
#include "WaitingRoom.h"
#include "Path.h"
#include <string>

const std::string begining("<span size='20000'>");
const std::string ending("</span>");

#define SPACING 30

WaitingRoom::WaitingRoom(): container(true, SPACING){
    this->label.set_use_markup(true);
    this->label.set_markup(begining + "Esperando jugadores..." + ending);
    this->image.set(WAITING_ROOM_IMAGE);
    this->container.pack_start(this->image, Gtk::PACK_SHRINK);
    this->container.pack_start(this->label, Gtk::PACK_SHRINK);
    this->container.set_halign(Gtk::ALIGN_CENTER);
    this->container.set_valign(Gtk::ALIGN_CENTER);
}

WaitingRoom::~WaitingRoom() {}

Gtk::Widget& WaitingRoom::getWidget() {
    return this->container;
}
```

jun 22, 18 12:22 **WaitingRoom.h** Page 1/1

```
#ifndef __WAITINGROOM_H__
#define __WAITINGROOM_H__

#include <gtkmm/label.h>
#include <gtkmm/image.h>
#include <gtkmm/hvbox.h>

/* Label de que indica la espera a otros jugadores */
class WaitingRoom {
private:
    Gtk::VBox container;
    Gtk::Label label;
    Gtk::Image image;

public:
    /* Constructor */
    WaitingRoom();

    /* Destructor */
    ~WaitingRoom();

    /* Devuelve el contenedor del mensaje */
    Gtk::Widget& getWidget();
};


#endif
```

jun 10, 18 14:54 **Handlers.cpp** Page 1/3

```
#include "Handlers.h"
#include <gtkmm/adjustment.h>
#include <gdk/gdkkeysyms.h>
#include "Player.h"
#include "ViewPositionTransformer.h"
#include "WeaponNames.h"

const char SPACE = ' ';
const int WEAPONS_DEFAULT_TIME = 3;
const char ASCII_OFFSET = 48;
const char ASCII_1 = 49;
const char ASCII_5 = 53;
const int MAX_TIME = 3000;
const int ANGLE_STEP = 6;

Handlers::Handlers(Player& player, ViewsList& view_list, WeaponList& weapons,
                   WorldView& world) :
        player(player), view_list(view_list), weapons(weapons), world(world),
        scroll_handler(world.getWindow()), power_accumulator(*this, MAX_TIME) {
    this->has_shoot = false;
    this->current_angle = DEFAULT_ANGLE;
    this->weapons_time = WEAPONS_DEFAULT_TIME;
    this->enabled = false;
}

Handlers::~Handlers() {}

void Handlers::enableAll() {
    this->weapons_time = WEAPONS_DEFAULT_TIME;
    this->current_angle = DEFAULT_ANGLE;
    this->has_shoot = false;
    this->enabled = true;

    this->player.getProtocol().updateScope(DEFAULT_ANGLE);

    Gtk::Container* window = this->world.getWindow().get_parent()->get_parent();

    window->set_can_focus(true);
    window->grab_focus();

    window->signal_key_press_event().connect(
            sigc::mem_fun(*this, &Handlers::keyPressHandler));
    window->signal_key_release_event().connect(
            sigc::mem_fun(*this, &Handlers::keyReleaseHandler));
    this->world.getWindow().signal_button_press_event().connect(
            sigc::mem_fun(*this, &Handlers::onButtonPressEvent));
}

void Handlers::disableAll() {
    this->enabled = false;
}

bool Handlers::isEnabled() const {
    return this->enabled;
}

void Handlers::powerAccumStopped(int power) {
    this->player.shoot(this->current_angle, power, this->weapons_time);
}

bool Handlers::keyPressHandler(GdkEventKey* key_event) {
    if (!this->enabled) {
        return true;
    }

    if (key_event->keyval == GDK_KEY_Left) {
```

```
67          this->player.getProtocol().sendMoveAction(MOVE_LEFT);
68      } else if (key_event->keyval == GDK_KEY_Right) {
69          this->player.getProtocol().sendMoveAction(MOVE_RIGHT);
70      } else if (key_event->keyval == GDK_KEY_Return) {
71          this->player.getProtocol().sendMoveAction(JUMP);
72      } else if (key_event->keyval == GDK_KEY_BackSpace) {
73          this->player.getProtocol().sendMoveAction(ROLLBACK);
74      } else if (key_event->keyval == GDK_KEY_Up) {
75          if (!this->weapons.getCurrentWeapon().hasScope()) {
76              return true;
77          }
78          if (this->current_angle < MAX_WEAPON_ANGLE) {
79              this->current_angle += ANGLE_STEP;
80          }
81          this->player.getProtocol().updateScope(this->current_angle);
82      } else if (key_event->keyval == GDK_KEY_Down) {
83          if (!this->weapons.getCurrentWeapon().hasScope()) {
84              return true;
85          }
86          if (this->current_angle > MIN_WEAPON_ANGLE) {
87              this->current_angle -= ANGLE_STEP;
88          }
89          this->player.getProtocol().updateScope(this->current_angle);
90      } else if (key_event->keyval >= ASCII_1 && key_event->keyval <= ASCII_5) {
91          this->weapons_time = key_event->keyval - ASCII_OFFSET;
92      } else if (key_event->keyval == SPACE && key_event->type == GDK_KEY_PRESS) {
93          if (this->weapons.getCurrentWeapon().isSelfDirected()) {
94              return true;
95          }
96          if (!this->weapons.getCurrentWeapon().hasAmmo()) {
97              return true;
98          }
99          if (this->has_shoot) {
100             return true;
101         }
102         this->has_shoot = true;
103         if (!this->weapons.getCurrentWeapon().hasVariablePower()) {
104             this->player.shoot(this->current_angle, -1, this->weapons_time);
105         } else {
106             this->power_accumulator.start();
107         }
108     }
109     return true;
110 }
111
112 bool Handlers::keyReleaseHandler(GdkEventKey* key_event) {
113     if (!this->enabled) {
114         return true;
115     }
116
117     if (key_event->type == GDK_KEY_RELEASE) {
118         if (key_event->keyval == SPACE) {
119             if (this->weapons.getCurrentWeapon().isSelfDirected()) {
120                 return true;
121             }
122             if (!this->weapons.getCurrentWeapon().hasVariablePower()) {
123                 return true;
124             }
125             if (!this->weapons.getCurrentWeapon().hasAmmo()) {
126                 this->player.getMusicPlayer().playNoAmmo();
127                 return true;
128             }
129             this->power_accumulator.stop();
130         }
131     }
132     return true;
```



```
133 }
134
135 bool Handlers::onButtonPressEvent(GdkEventButton* event) {
136     if (!this->enabled) {
137         return true;
138     }
139
140     if (!this->weapons.getCurrentWeapon().isSelfDirected()) {
141         return true;
142     }
143     if (!this->weapons.getCurrentWeapon().hasAmmo()) {
144         this->player.getMusicPlayer().playNoAmmo();
145         return true;
146     }
147     if (this->has_shoot) {
148         return true;
149     }
150     if ((event->type == GDK_BUTTON_PRESS) && (event->button == 1)) {
151         float x = event->x;
152         float y = event->y;
153         x += this->world.getWindow().get_hadjustment()->get_value();
154         y += this->world.getWindow().get_vadjustment()->get_value();
155         Position position(x, y);
156         Position newPosition = ViewPositionTransformer(
157                 this->world.getLayout()).transformToPosition(position);
158         this->has_shoot = true;
159         this->player.shoot(newPosition);
160     }
161     return true;
162 }
163
164 int Handlers::getCurrentAngle() const {
165     return this->current_angle;
166 }
167
168 void Handlers::stop() {
169     this->scroll_handler.stop();
170 }
```

```
1  #ifndef __HANDLERS__H__
2  #define __HANDLERS__H__
3
4  #include <gdk/gdk.h>
5  #include "WeaponPowerAccum.h"
6  #include "ScrollHandler.h"
7
8  class Player;
9
10 class ViewsList;
11
12 class WeaponList;
13
14 class WorldView;
15
16 /* Clase que se encarga de definir los handlers del teclado y
17    del mouse. */
18 class Handlers {
19 private:
20     Player& player;
21     ViewsList& view_list;
22     WeaponList& weapons;
23     WorldView& world;
24     ScrollHandler scroll_handler;
25
26     bool has_shoot;
27     int current_angle;
28     int weapons_time;
29     bool enabled;
30
31     WeaponPowerAccum power_accumulator;
32
33
34 public:
35     /* Constructor */
36     Handlers(Player& player, ViewsList& view_list, WeaponList& weapons,
37              WorldView& world);
38
39     /* Destructor */
40     ~Handlers();
41
42     /* Handler completo para el presionado de teclas. Indica
43        los pasos que se deben realizar al presionar una tecla
44        especifica */
45     bool keyPressHandler(GdkEventKey* key_event);
46
47     /* Handler completo para la liberaciÃ³n de teclas. Indica
48        los pasos que se deben realizar al liberar una tecla
49        especifica */
50     bool keyReleaseHandler(GdkEventKey* key_event);
51
52     /* Handler del mouse. Indica los pasos que se deben realizar
53        al utilizar el mouse */
54     bool onButtonPressEvent(GdkEventButton* event);
55
56     /* Habilita todos los handlers */
57     void enableAll();
58
59     /* Deshabilita todos los handlers */
60     void disableAll();
61
62     /* Devuelve true si los handlers estan habilitados */
63     bool isEnabled() const;
64
65     /* Realiza el shoot del player */
66     void powerAccumStopped(int power);
```



```
67
68     /* Devuelve el angulo actual del scope */
69     int getCurrentAngle() const;
70
71     /* Detiene los handlers */
72     void stop();
73 };
74
75 #endif
```

jun 10, 18 15:55 **Player.cpp** Page 1/2

```cpp
#include "Player.h"
#include <string>
#include "WeaponNames.h"

Player::Player(ClientProtocol& protocol, const std::string& name,
               Gtk::Window& window, MenuView& main_menu) :
        protocol(protocol), name(name),
        screen(window, main_menu, *this, this->weapons),
        turn(*this, this->screen.getTurnLabel()),
        view_list(this->screen.getWorld(), *this, this->screen.getPlayersView(),
                  musicPlayer),
        data_receiver(*this),
        handlers(*this, this->view_list, this->weapons,
                 this->screen.getWorld()) {
    this->musicPlayer.playMusic();
    this->data_receiver.start();
}

Player::~Player() {
    this->data_receiver.stop();
    this->data_receiver.join();
}

void Player::startTurn(int worm_id, int player_id, float wind) {
    this->view_list.setCurrentWorm(worm_id);
    this->screen.getWindView().update(wind);
    const std::string& current_player = this->screen.getPlayersView().getPlayer(
            player_id);
    if (current_player == this->name) {
        //Es mi turno
        this->musicPlayer.playStartTurnSound();
        this->handlers.enableAll();
        this->changeWeapon(this->weapons.getCurrentWeapon().getName());
        this->screen.getTurnLabel().beginTurn();
        this->turn.start();
    } else {
        this->screen.getTurnLabel().beginTurn(current_player);
    }
}

void Player::endTurn() {
    this->turn.stop();
    this->screen.getTurnLabel().endTurn();
    this->view_list.removeScopeVisibility();
}

void Player::endGame(const std::string& winner) {
    this->data_receiver.stop();
    this->screen.getTurnLabel().setEndGame();
    this->view_list.setVictory();
    this->protocol.sendEndGame();
    this->handlers.stop();
    this->screen.setWinner(winner, this->name == winner);
}

void Player::shootWeapon() {
    this->turn.reduceTime();
    this->weapons.getCurrentWeapon().shoot();
}

void Player::changeWeapon(std::string weapon) {
    this->musicPlayer.playSelectWeaponSound();
    this->weapons.changeWeapon(weapon);
    if (this->handlers.isEnabled()) {
        this->protocol.sendChangeWeapon(weapon);
    }
```

jun 10, 18 15:55 **Player.cpp** Page 2/2

```cpp
}

void Player::shoot(Position position) {
    this->shootWeapon();
    this->protocol.sendWeaponSelfDirectedShoot(position);
    this->screen.getWeaponsView().updateAmmo(this->weapons.getCurrentWeapon());
}

void Player::playTickTime() {
    this->musicPlayer.playTickSound();
}

void Player::shoot(int angle, int power, int time) {
    this->shootWeapon();
    if (!this->weapons.getCurrentWeapon().isTimed()) {
        time = -1;
    }
    if (!this->weapons.getCurrentWeapon().hasScope()) {
        angle = MAX_WEAPON_ANGLE * 8;
    }
    this->protocol.sendWeaponShoot(angle, power, time);
    this->view_list.removeScopeVisibility();
    this->screen.getWeaponsView().updateAmmo(this->weapons.getCurrentWeapon());
}

ViewsList& Player::getViewsList() {
    return this->view_list;
}

ScreenView& Player::getScreen() {
    return this->screen;
}

WeaponList& Player::getWeapons() {
    return this->weapons;
}

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

MusicPlayer& Player::getMusicPlayer() {
    return this->musicPlayer;
}

Turn& Player::getTurn() {
    return this->turn;
}
```

```
#ifndef __CLIENTPLAYER_H__
#define __CLIENTPLAYER_H__

#include <memory>
#include <gtkmm/window.h>
#include <string>
#include "MenuView.h"
#include "ClientProtocol.h"
#include "Turn.h"
#include "Weapon.h"
#include "WeaponList.h"
#include "ScreenView.h"
#include "ViewsList.h"
#include "Position.h"
#include "DataReceiver.h"
#include "Handlers.h"
#include "MusicPlayer.h"

class Player {
private:
    ClientProtocol& protocol;
    std::string name;
    WeaponList weapons;
    ScreenView screen;
    Turn turn;
    ViewsList view_list;
    DataReceiver data_receiver;
    Handlers handlers;
    MusicPlayer musicPlayer;

    /* Reduce el tiempo del turno y actualiza la municion */
    void shootWeapon();

public:
    /* Constructor */
    Player(ClientProtocol& protocol, const std::string& name,
           Gtk::Window& window, MenuView& main_menu);

    /* Destructor */
    ~Player();


    /* Comienza el turno. Si es el turno del jugador entonces,
       habilita los handlers, sino muestra los movimientos realizados
       por el otro jugador */
    void startTurn(int worm_id, int player_id, float wind);

    /* Finaliza el turno del jugador actual */
    void endTurn();

    /* Finaliza el juego */
    void endGame(const std::string& winner);

    /* Cambia el arma actual por la espeificada */
    void changeWeapon(std::string weapon);

    /* Realiza el disparo del arma con el angulo, potencia
       y tiempo pasados */
    void shoot(int angle, int power, int time);

    /* Realiza el disparo del arma en la posicion pasada */
    void shoot(Position position);

    /* Reproduce el sonido de falta de tiempo */
    void playTickTime();

```



```
    /* Devuelve la lista de los elementos presentes en la vista */
    ViewsList& getViewsList();

    /* Devuelve la vista */
    ScreenView& getScreen();

    /* Devuelve la lista de armas */
    WeaponList& getWeapons();

    /* Devuelve el protocolo */
    ClientProtocol& getProtocol();

    /* Devuelve el music player */
    MusicPlayer& getMusicPlayer();

    /* Devuelve el turno */
    Turn& getTurn();
};

#endif
```

jun 10, 18 14:54 **Turn.cpp** Page 1/1

```cpp
#include "Turn.h"
#include <glibmm/main.h>
#include "Player.h"

const int TIME_DEFAULT = 60;
const int REDUCTION_TIME_DEFAULT = 3;
const int LIMIT_TIME = 10;

Turn::Turn(Player& player, TurnLabel& time_label) :
        actual_time(TIME_DEFAULT), player(player), time_label(time_label),
        max_time(TIME_DEFAULT), reduction_time(REDUCTION_TIME_DEFAULT) {}

Turn::~Turn() {}

bool Turn::startCallBack() {
    if (this->actual_time <= LIMIT_TIME) {
        this->player.playTickTime();
    }

    this->actual_time--;
    if (this->actual_time < 0) {
        return false;
    }
    this->time_label.setTime(this->actual_time);
    return true;
}

void Turn::start() {
    this->actual_time = this->max_time;
    this->my_connection = Glib::signal_timeout().connect(
            sigc::mem_fun(*this, &Turn::startCallBack), 1000);
}

void Turn::reduceTime() {
    this->actual_time = this->reduction_time;
}

void Turn::stop() {
    if (this->my_connection.connected()) {
        this->my_connection.disconnect();
    }
}

void Turn::setTime(int time, int reduction_time) {
    this->max_time = time;
    this->reduction_time = reduction_time;
}
```

jun 10, 18 14:54 **Turn.h** Page 1/1

```cpp
#ifndef __CLIENTTURN_H__
#define __CLIENTTURN_H__

#include "TurnLabel.h"

class Player;

/* Clase que se encarga de contar el tiempo del turno */
class Turn {
private:
    int actual_time;
    Player& player;
    TurnLabel& time_label;
    sigc::connection my_connection;
    int max_time;
    int reduction_time;

    /* Callback de start */
    bool startCallBack();

public:
    /* Constructor */
    Turn(Player& player, TurnLabel& time_label);

    /* Destructor */
    ~Turn();


    /* Comienza la cuenta regresiva del turno actualizando el
     * label que muestra el tiempo */
    void start();

    /* Reduce el tiempo restante del turno a 3 segundos */
    void reduceTime();

    /* Detiene el contador y finaliza el turno */
    void stop();

    /* Setea los tiempos */
    void setTime(int time, int reduction_time);
};

#endif
```

jun 10, 18 15:53 **DistanceWeapon.cpp** Page 1/1

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

DistanceWeapon::DistanceWeapon(std::string name, int ammo, bool time) :
        Weapon(name, ammo) {
    this->has_Scope = true;
    this->is_Timed = time;
}

DistanceWeapon::~DistanceWeapon() {}

DistanceWeapon::DistanceWeapon(DistanceWeapon&& other) : Weapon(
        std::move(other)) {}

bool DistanceWeapon::hasVariablePower() const {
    return true;
}

```

jun 10, 18 15:54 **DistanceWeapon.h** Page 1/1

```cpp
#ifndef __CLIENTDISTANCEWEAPON_H__
#define __CLIENTDISTANCEWEAPON_H__

#include "Weapon.h"
#include <string>

/* Clase que se encarga de representar a las armas de distancia */
class DistanceWeapon : public Weapon {
public:
    /* Constructor */
    DistanceWeapon(std::string name, int ammo, bool time = false);

    /* Destructor */
    ~DistanceWeapon();

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


    /* Devuelve true si el arma tiene potencia variable */
    bool hasVariablePower() const override;
};

#endif
```

jun 10, 18 15:49 **MeleeWeapon.cpp** Page 1/1

```cpp
#include "MeleeWeapon.h"
#include <limits>
#include <string>

MeleeWeapon::MeleeWeapon(std::string name, int ammo, bool scope, bool time) :
        Weapon(name, ammo) {
    this->has_Scope = scope;
    this->is_Timed = time;
}

MeleeWeapon::MeleeWeapon(MeleeWeapon&& other) : Weapon(std::move(other)) {}

```

jun 10, 18 15:53 **MeleeWeapon.h** Page 1/1

```cpp
#ifndef __CLIENTMELEEWEAPON_H__
#define __CLIENTMELEEWEAPON_H__

#include "Weapon.h"
#include <string>

/* Clase que se encarga de representar las armas de cuerpo a cuerpo */
class MeleeWeapon : public Weapon {
public:
    /* Constructor */
    MeleeWeapon(std::string name, int ammo, bool scope, bool time = false);

    /* Destructor */
    ~MeleeWeapon() {}

    /* Constructor por movimiento */
    MeleeWeapon(MeleeWeapon&& other);
};

#endif
```

jun 10, 18 14:54 **WeaponPowerAccum.cpp** Page 1/1

```cpp
#include "WeaponPowerAccum.h"
#include "Handlers.h"

const int TIME_STEP = 50;
const int MINIMUM_POWER = 1000;
const int POWER_STEP = 15;

WeaponPowerAccum::WeaponPowerAccum(Handlers& handlers, int time) :
        actual_time(0), max_time(time), handlers(handlers) {}

WeaponPowerAccum::~WeaponPowerAccum() {}

bool WeaponPowerAccum::startCallBack() {
    this->actual_time += TIME_STEP;
    this->power += POWER_STEP;

    if (this->actual_time == this->max_time) {
        this->handlers.powerAccumStopped(this->power);
        return false;
    }
    return true;
}

void WeaponPowerAccum::start() {
    this->actual_time = 0;
    this->power = MINIMUM_POWER;
    this->my_connection = Glib::signal_timeout().connect(
            sigc::mem_fun(*this, &WeaponPowerAccum::startCallBack), TIME_STEP);
}

void WeaponPowerAccum::stop() {
    if (this->my_connection.connected()) {
        this->my_connection.disconnect();
        this->handlers.powerAccumStopped(this->power);
    }
}
```

jun 10, 18 14:54 **WeaponPowerAccum.h** Page 1/1

```cpp
#ifndef __CLIENTTIMER_H__
#define __CLIENTTIMER_H__

#include <glibmm/main.h>

class Handlers;

/* Clase que simula a un contador */
class WeaponPowerAccum {
private:
    int actual_time;
    int max_time;
    int power;
    Handlers& handlers;
    sigc::connection my_connection;

    /* Callback de start */
    bool startCallBack();

public:
    /* Constructor */
    WeaponPowerAccum(Handlers& handlers, int time);

    /* Destructor */
    ~WeaponPowerAccum();

    /* Cuenta el tiempo transcurrido y llama al metodo timerStopped
       de la clase Handler con este tiempo */
    void start();

    /* Detiene el contador */
    void stop();
};

#endif
```

jun 10, 18 14:54 **AirAttack.cpp** Page 1/1

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

AirAttack::AirAttack(int ammo) : SelfDirectedWeapon(AIR_ATTACK_NAME, ammo) {}

AirAttack::~AirAttack() {}

AirAttack::AirAttack(AirAttack&& other) : SelfDirectedWeapon(
        std::move(other)) {}

```

jun 10, 18 15:51 **AirAttack.h** Page 1/1

```cpp
#ifndef __CLIENTAIRATTACK_H__
#define __CLIENTAIRATTACK_H__

#include "SelfDirectedWeapon.h"

/* Clase que representa al arma AirStrike */
class AirAttack : public SelfDirectedWeapon {
public:
    /* Constructor */
    explicit AirAttack(int ammo);

    /* Destructor */
    ~AirAttack();

    /* Constructor por movimiento */
    AirAttack(AirAttack&& other);
};

#endif
```

jun 09, 18 19:06 **Banana.cpp** Page 1/1

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

Banana::Banana(int ammo) : DistanceWeapon(BANANA_NAME, ammo, true) {}

Banana::~Banana() {}

Banana::Banana(Banana&& other) : DistanceWeapon(std::move(other)) {}

```

jun 10, 18 15:51 **Banana.h** Page 1/1

```cpp
#ifndef __CLIENTBANANA_H__
#define __CLIENTBANANA_H__

#include "DistanceWeapon.h"

/* Clase que representa al arma Banana */
class Banana : public DistanceWeapon {
public:
    /* Constructor */
    explicit Banana(int ammo);

    /* Destructor */
    ~Banana();

    /* Constructor por movimiento */
    Banana(Banana&& other);
};

#endif
```

jun 10, 18 14:54 **Bat.cpp** Page 1/1

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

Bat::Bat(int ammo) : MeleeWeapon(BAT_NAME, ammo, true) {}

Bat::~Bat() {}

Bat::Bat(Bat&& other) : MeleeWeapon(std::move(other)) {}

```

jun 10, 18 15:51 **Bat.h** Page 1/1

```cpp
#ifndef __CLIENTBAT_H__
#define __CLIENTBAT_H__

#include "MeleeWeapon.h"

/* Clase que representa al arma Bat de baseball */
class Bat : public MeleeWeapon {
public:
    /* Constructor */
    explicit Bat(int ammo);

    /* Destructor */
    ~Bat();

    /* Constructor por movimiento */
    Bat(Bat&& other);
};

#endif
```

jun 10, 18 14:54 **Bazooka.cpp** Page 1/1

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

Bazooka::Bazooka(int ammo) : DistanceWeapon(BAZOOKA_NAME, ammo) {}

Bazooka::~Bazooka() {}

Bazooka::Bazooka(Bazooka&& other) : DistanceWeapon(std::move(other)) {}

```

jun 10, 18 15:51 **Bazooka.h** Page 1/1

```cpp
#ifndef __CLIENTBAZOOKA_H__
#define __CLIENTBAZOOKA_H__

#include "DistanceWeapon.h"

/* Clase que representa al arma Bazooka */
class Bazooka : public DistanceWeapon {
public:
    /* Constructor */
    explicit Bazooka(int ammo);

    /* Destructor */
    ~Bazooka();

    /* Constructor por movimiento */
    Bazooka(Bazooka&& other);
};

#endif
```

jun 10, 18 14:54 **Dynamite.cpp** Page 1/1

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

Dynamite::Dynamite(int ammo) : MeleeWeapon(DYNAMITE_NAME, ammo, false, true) {}

Dynamite::~Dynamite() {}

Dynamite::Dynamite(Dynamite&& other) : MeleeWeapon(std::move(other)) {}

```

jun 10, 18 15:51 **Dynamite.h** Page 1/1

```cpp
#ifndef __CLIENTDYNAMITE_H__
#define __CLIENTDYNAMITE_H__

#include "MeleeWeapon.h"

/* Clase que representa al arma Dinamita */
class Dynamite : public MeleeWeapon {
public:
    /* Constructor */
    explicit Dynamite(int ammo);

    /* Destructor */
    ~Dynamite();

    /* Constructor por movimiento */
    Dynamite(Dynamite&& other);
};

#endif
```

jun 10, 18 14:54 **GreenGrenade.cpp** Page 1/1

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

GreenGrenade::GreenGrenade(int ammo) :
        DistanceWeapon(GREEN_GRENADE_NAME, ammo, true) {}

GreenGrenade::~GreenGrenade() {}

GreenGrenade::GreenGrenade(GreenGrenade&& other) : DistanceWeapon(
        std::move(other)) {}

```

jun 10, 18 15:51 **GreenGrenade.h** Page 1/1

```cpp
#ifndef __CLIENTGREENGRENADE_H__
#define __CLIENTGREENGRENADE_H__

#include "DistanceWeapon.h"

/* Clase que representa al arma Granada verde */
class GreenGrenade : public DistanceWeapon {
public:
    /* Constructor */
    explicit GreenGrenade(int ammo);

    /* Destructor */
    ~GreenGrenade();

    /* Constructor por movimiento */
    GreenGrenade(GreenGrenade&& other);
};

#endif
```

jun 10, 18 14:54 **HolyGrenade.cpp** Page 1/1

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

HolyGrenade::HolyGrenade(int ammo) :
        DistanceWeapon(HOLY_GRENADE_NAME, ammo, true) {}

HolyGrenade::~HolyGrenade() {}

HolyGrenade::HolyGrenade(HolyGrenade&& other) : DistanceWeapon(
        std::move(other)) {}

```

jun 10, 18 15:51 **HolyGrenade.h** Page 1/1

```cpp
#ifndef __CLIENTHOLYGRENADE_H__
#define __CLIENTHOLYGRENADE_H__

#include "DistanceWeapon.h"

/* Clase que representa al arma Granada santa */
class HolyGrenade : public DistanceWeapon {
public:
    /* Constructor */
    explicit HolyGrenade(int ammo);

    /* Destructor */
    ~HolyGrenade();

    /* Constructor por movimiento */
    HolyGrenade(HolyGrenade&& other);
};

#endif
```

jun 10, 18 14:54 **Mortar.cpp** Page 1/1

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

Mortar::Mortar(int ammo) : DistanceWeapon(MORTAR_NAME, ammo, false) {}

Mortar::~Mortar() {}

Mortar::Mortar(Mortar&& other) : DistanceWeapon(std::move(other)) {}

```

jun 10, 18 15:51 **Mortar.h** Page 1/1

```cpp
#ifndef __CLIENTMORTAR_H__
#define __CLIENTMORTAR_H__

#include "DistanceWeapon.h"

/* Clase que representa al arma Mortero */
class Mortar : public DistanceWeapon {
public:
    /* Constructor */
    explicit Mortar(int ammo);

    /* Destructor */
    ~Mortar();

    /* Constructor por movimiento */
    Mortar(Mortar&& other);
};

#endif
```

jun 10, 18 14:54 **RedGrenade.cpp** Page 1/1

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

RedGrenade::RedGrenade(int ammo) :
        DistanceWeapon(RED_GRENADE_NAME, ammo, true) {}

RedGrenade::~RedGrenade() {}

RedGrenade::RedGrenade(RedGrenade&& other) : DistanceWeapon(std::move(other)) {}

```

jun 10, 18 15:51 **RedGrenade.h** Page 1/1

```cpp
#ifndef __CLIENTREDGRENADE_H__
#define __CLIENTREDGRENADE_H__

#include "DistanceWeapon.h"

/* Clase que representa al arma Granada roja */
class RedGrenade : public DistanceWeapon {
public:
    /* Constructor */
    explicit RedGrenade(int ammo);

    /* Destructor */
    ~RedGrenade();

    /* Constructor por movimiento */
    RedGrenade(RedGrenade&& other);
};

#endif
```

jun 10, 18 14:54 **Teleportation.cpp** Page 1/1

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

Teleportation::Teleportation(int ammo) : SelfDirectedWeapon(TELEPORT_NAME,
                                                            ammo) {}

Teleportation::~Teleportation() {}

Teleportation::Teleportation(Teleportation&& other) : SelfDirectedWeapon(
        std::move(other)) {}

```

jun 10, 18 15:51 **Teleportation.h** Page 1/1

```cpp
#ifndef __CLIENTTELEPORTATION_H__
#define __CLIENTTELEPORTATION_H__

#include "SelfDirectedWeapon.h"

/* Clase que representa al arma Teletransportador */
class Teleportation : public SelfDirectedWeapon {
public:
    /* Constructor */
    explicit Teleportation(int ammo);

    /* Destructor */
    ~Teleportation();

    /* Constructor por movimiento */
    Teleportation(Teleportation&& other);
};

#endif
```

jun 10, 18 15:54 **SelfDirectedWeapon.cpp** Page 1/1

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

SelfDirectedWeapon::SelfDirectedWeapon(std::string name, int ammo) : Weapon(
        name, ammo) {}

SelfDirectedWeapon::~SelfDirectedWeapon() {}

SelfDirectedWeapon::SelfDirectedWeapon(SelfDirectedWeapon&& other) : Weapon(
        std::move(other)) {}

bool SelfDirectedWeapon::isSelfDirected() const {
    return true;
}

```

jun 10, 18 15:53 **SelfDirectedWeapon.h** Page 1/1

```cpp
#ifndef __SELFDIRECTEDWEAPON_H__
#define __SELFDIRECTEDWEAPON_H__

#include "Weapon.h"
#include <string>

/* Clase que representa las armas teledirigidas */
class SelfDirectedWeapon : public Weapon {
public:
    /* Constructor */
    SelfDirectedWeapon(std::string name, int ammo);

    /* Destructor */
    ~SelfDirectedWeapon();

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

    /* Devuelve true si es teledirigida */
    bool isSelfDirected() const override;
};

#endif
```

jun 10, 18 15:53 **Weapon.cpp** Page 1/1

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

Weapon::Weapon(std::string name, int ammo) :
        name(name), ammo(ammo), has_Scope(false), is_Timed(false) {}

Weapon::~Weapon() {}

Weapon::Weapon(Weapon&& other) {
    this->name = std::move(other.name);
    this->ammo = std::move(other.ammo);
    this->has_Scope = std::move(other.has_Scope);
    this->is_Timed = std::move(other.is_Timed);
}

Weapon& Weapon::operator=(Weapon&& other) {
    this->name = std::move(other.name);
    this->ammo = std::move(other.ammo);
    this->has_Scope = std::move(other.has_Scope);
    this->is_Timed = std::move(other.is_Timed);
    return *this;
}

bool Weapon::hasScope() const {
    return this->has_Scope;
}

bool Weapon::isSelfDirected() const {
    return false;
}

bool Weapon::isTimed() const {
    return this->is_Timed;
}

bool Weapon::hasVariablePower() const {
    return false;
}

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

void Weapon::shoot() {
    if (this->ammo <= 100)
        this->ammo--;
}

bool Weapon::hasAmmo() const {
    return this->ammo > 0;
}

unsigned int Weapon::getAmmo() const {
    return this->ammo;
}

```

jun 10, 18 14:54 **Weapon.h** Page 1/1

```cpp
#ifndef __CLIENTWEAPON_H__
#define __CLIENTWEAPON_H__

#include <string>

/* Clase que se encarga de representar a las armas del juego */
class Weapon {
protected:
    std::string name;
    unsigned int ammo;
    bool has_Scope;
    bool is_Timed;

public:
    /* Constructor */
    Weapon(std::string name, int ammo);

    /* Destructor */
    ~Weapon();

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

    /* Operador = por movimiento */
    Weapon& operator=(Weapon&& other);


    /* Devuelve true si el arma tiene mira */
    virtual bool hasScope() const;

    /* Devuelve true si el arma es teledirigida */
    virtual bool isSelfDirected() const;

    /* Devuelve true si el arma es por tiempo */
    virtual bool isTimed() const;

    /* Devuelve true si el arma tiene potencia variable */
    virtual bool hasVariablePower() const;

    /* Devuelve el nombre del arma */
    virtual const std::string& getName() const;

    /* Disminuye la cantidad de municiones del arma */
    virtual void shoot();

    /* Devuelve true si el arma tiene balas */
    virtual bool hasAmmo() const;

    /* Devuelve la cantidad de balas */
    unsigned int getAmmo() const;
};

#endif
```

jun 10, 18 15:49 **WeaponList.cpp** Page 1/1

```cpp
#include "WeaponList.h"
#include <utility>
#include <string>
#include "WeaponNames.h"

WeaponList::WeaponList() : current_weapon(DEFAULT_WEAPON) {}

WeaponList::~WeaponList() {}

void WeaponList::add(std::string weapon, int ammo) {
    WeaponsFactory factory;
    this->weapons.insert(std::pair<std::string, weapon_ptr>(weapon, std::move(
            factory.createWeapon(weapon, ammo))));
}

void WeaponList::changeWeapon(std::string weapon) {
    this->current_weapon = weapon;
}

Weapon& WeaponList::getCurrentWeapon() {
    return *this->weapons.at(this->current_weapon);
}

WeaponList::iterator WeaponList::begin() {
    return this->weapons.begin();
}

WeaponList::iterator WeaponList::end() {
    return this->weapons.end();
}

```

jun 10, 18 15:52 **WeaponList.h** Page 1/1

```cpp
#ifndef __CLIENTWEAPONLIST_H__
#define __CLIENTWEAPONLIST_H__

#include <map>
#include <string>
#include "Weapon.h"
#include "WeaponsFactory.h"

/* Clase que se encarga de almacenar las armas del juego */
class WeaponList {
private:
    typedef std::map<std::string, weapon_ptr> WeaponsList;
    WeaponsList weapons;
    std::string current_weapon;

public:
    /* Constructor */
    WeaponList();

    /* Destructor */
    ~WeaponList();


    /* Agrega un arma a la lista */
    void add(std::string weapon, int ammo);

    /* Devuelve el arma actual */
    Weapon& getCurrentWeapon();

    /* Cambia el arma actual por la especificada */
    void changeWeapon(std::string weapon);

    typedef WeaponsList::iterator iterator;
    typedef WeaponsList::const_iterator const_iterator;

    iterator begin();

    iterator end();
};


#endif
```

jun 10, 18 15:50 **WeaponsFactory.cpp** Page 1/1

```cpp
#include "WeaponsFactory.h"
#include "WeaponNames.h"
#include <string>
#include "AirAttack.h"
#include "Banana.h"
#include "Bat.h"
#include "Bazooka.h"
#include "Dynamite.h"
#include "GreenGrenade.h"
#include "HolyGrenade.h"
#include "Mortar.h"
#include "RedGrenade.h"
#include "Teleportation.h"


WeaponsFactory::WeaponsFactory() {}

WeaponsFactory::~WeaponsFactory() {}

weapon_ptr WeaponsFactory::createWeapon(std::string weapon, int ammo) {
    if (weapon == AIR_ATTACK_NAME)
        return weapon_ptr(new AirAttack(ammo));
    else if (weapon == BANANA_NAME)
        return weapon_ptr(new Banana(ammo));
    else if (weapon == BAT_NAME)
        return weapon_ptr(new Bat(ammo));
    else if (weapon == BAZOOKA_NAME)
        return weapon_ptr(new Bazooka(ammo));
    else if (weapon == DYNAMITE_NAME)
        return weapon_ptr(new Dynamite(ammo));
    else if (weapon == GREEN_GRENADE_NAME)
        return weapon_ptr(new GreenGrenade(ammo));
    else if (weapon == HOLY_GRENADE_NAME)
        return weapon_ptr(new HolyGrenade(ammo));
    else if (weapon == MORTAR_NAME)
        return weapon_ptr(new Mortar(ammo));
    else if (weapon == RED_GRENADE_NAME)
        return weapon_ptr(new RedGrenade(ammo));
    return weapon_ptr(new Teleportation(ammo));
}
```

jun 10, 18 15:52 **WeaponsFactory.h** Page 1/1

```cpp
#ifndef __CLIENTWEAPONSFACTORY_H__
#define __CLIENTWEAPONSFACTORY_H__

#include <memory>
#include <string>
#include "Weapon.h"

typedef std::unique_ptr<Weapon> weapon_ptr;

/* Clase que se encarga de crear las armas del juego */
class WeaponsFactory {
public:
    /* Constructor */
    WeaponsFactory();

    /* Destructor */
    ~WeaponsFactory();


    /* Crea el arma especificada con las municiones especificadas */
    weapon_ptr createWeapon(std::string weapon, int ammo);
};


#endif
```

jun 10, 18 14:54 **MusicPath.h** Page 1/1

```
#ifndef WORMS_MUSICPATH_H
#define WORMS_MUSICPATH_H

#include <string>
#include "Path.h"

const std::string BACKGROUND_MUSIC = SOUNDS_PATH + "BackgroundMusic.mp3";
const std::string START_TURN_SOUND = SOUNDS_PATH + "Misc/StartRound.wav";
const std::string TICK_SOUND = SOUNDS_PATH + "Misc/TimerTick.wav";
const std::string RUN_AWAY_SOUND = SOUNDS_PATH + "Worms/RunAway.wav";
const std::string DEATH_SOUND = SOUNDS_PATH + "Worms/Death.wav";
const std::string DAMAGE_RECEIVE_SOUND =
        SOUNDS_PATH + "Worms/DamageReceive.wav";
const std::string EXPLOSION_SOUND = SOUNDS_PATH + "Weapons/Explosion.wav";
const std::string TELEPORT_SOUND = SOUNDS_PATH + "Weapons/Teleportation.wav";
const std::string BAT_SOUND = SOUNDS_PATH + "Weapons/BaseballSound.wav";
const std::string HOLY_GRENADE_SOUND = SOUNDS_PATH + "Weapons/HolyGrenade.wav";
const std::string AIR_ATTACK_SOUND = SOUNDS_PATH + "Weapons/AirAttack.wav";
const std::string SHOOT_SOUND = SOUNDS_PATH + "Weapons/ShootWeapon.wav";
const std::string ROLLBACK_SOUND = SOUNDS_PATH + "Misc/RollBack.wav";
const std::string JUMP_SOUND = SOUNDS_PATH + "Misc/Jump.wav";
const std::string SELECT_WEAPON_SOUND = SOUNDS_PATH + "Misc/SelectWeapon.wav";
const std::string NO_AMMO_SOUND = SOUNDS_PATH + "Misc/NoAmmo.wav";
const std::string VICTORY_SOUND = SOUNDS_PATH + "Worms/Victory.WAV";

#endif //WORMS_MUSICPATH_H
```

jun 10, 18 15:48 **MusicPlayer.cpp** Page 1/3

```
#include "MusicPlayer.h"
#include <map>
#include <string>
#include "MusicPlayerException.h"
#include "WeaponNames.h"
#include "Protocol.h"
#include "MusicPath.h"

MusicPlayer::MusicPlayer() {
    this->music = NULL;
    // Initialize SDL.
    if (SDL_Init(SDL_INIT_AUDIO) < 0) {
        throw MusicPlayerException("Error al inicializar SDL");
    }

    //Initialize SDL_mixer
    if (Mix_OpenAudio(22050, MIX_DEFAULT_FORMAT, 2, 4096) == -1) {
        throw MusicPlayerException("Error al inicializar SDL mixer");
    }

    // Load background music
    this->music = Mix_LoadMUS(BACKGROUND_MUSIC.c_str());
    if (this->music == NULL) {
    }
}

MusicPlayer::~MusicPlayer() {
    Mix_HaltChannel(-1);
    this->stop();
    if (this->music != NULL) {
        Mix_FreeMusic(this->music);
    }
    std::map<int, Mix_Chunk*>::iterator iter;
    for (iter = this->effects.begin(); iter != this->effects.end(); iter++) {
        Mix_FreeChunk(iter->second);
    }
    // quit SDL_mixer
    Mix_CloseAudio();
    Mix_Quit();
    SDL_Quit();
}

void MusicPlayer::check(int channel) {
    if (this->effects.find(channel) != this->effects.end()) {
        // elimino el audio anterior de este canal
        Mix_FreeChunk(this->effects.at(channel));
        this->effects.erase(channel);
    }
    std::map<int, Mix_Chunk*>::iterator iter = this->effects.begin();
    while (iter != this->effects.end()) {
        if (!Mix_Playing(iter->first)) {
            Mix_FreeChunk(iter->second);
            iter = this->effects.erase(iter);
        } else {
            iter++;
        }
    }
}

void MusicPlayer::addEffect(const std::string& audio) {
    int channel;
    Mix_Chunk* effect = NULL;
    effect = Mix_LoadWAV(audio.c_str());
    if (effect == NULL) {
        return;
    }
```

```
67      if ((channel = Mix_PlayChannel(-1, effect, 0)) == -1) {
68          Mix_FreeChunk(effect);
69          return;
70      }
71      this->check(channel);
72      this->effects.insert(std::make_pair(channel, effect));
73  }
74
75  void MusicPlayer::playMusic() {
76      Mix_PlayMusic(this->music, -1);
77      Mix_VolumeMusic(MIX_MAX_VOLUME / 4);
78  }
79
80  void MusicPlayer::playStartTurnSound() {
81      this->addEffect(START_TURN_SOUND);
82  }
83
84  void MusicPlayer::playTickSound() {
85      this->addEffect(TICK_SOUND);
86  }
87
88  void MusicPlayer::playDeathSound() {
89      this->addEffect(DEATH_SOUND);
90  }
91
92  void MusicPlayer::playDamageReceiveSound() {
93      this->addEffect(DAMAGE_RECEIVE_SOUND);
94  }
95
96  void MusicPlayer::playExplosionSound(const std::string& weapon) {
97      if (weapon == HOLY_GRENADE_NAME) {
98          this->addEffect(HOLY_GRENADE_SOUND);
99      } else {
100         this->addEffect(EXPLOSION_SOUND);
101     }
102 }
103
104 void MusicPlayer::playVictory() {
105     this->addEffect(VICTORY_SOUND);
106 }
107
108 void MusicPlayer::playNoAmmo() {
109     this->addEffect(NO_AMMO_SOUND);
110 }
111
112 void MusicPlayer::stop() {
113     Mix_HaltMusic();
114 }
115
116 void MusicPlayer::playWeaponShotSound(const std::string& weapon) {
117     if (weapon == TELEPORT_NAME) {
118         this->addEffect(TELEPORT_SOUND);
119     } else if (weapon == BAT_NAME) {
120         this->addEffect(BAT_SOUND);
121     } else if (weapon == DYNAMITE_NAME) {
122         this->addEffect(RUN_AWAY_SOUND);
123     } else if (weapon == AIR_ATTACK_NAME) {
124         this->addEffect(AIR_ATTACK_SOUND);
125     } else {
126         this->addEffect(SHOOT_SOUND);
127     }
128 }
129
130 void MusicPlayer::playJumpSound(char action) {
131     if (action == ROLLBACK) {
132         this->addEffect(ROLLBACK_SOUND);
```



```
133     } else if (action == JUMP) {
134         this->addEffect(JUMP_SOUND);
135     }
136 }
137
138 void MusicPlayer::playSelectWeaponSound() {
139     this->addEffect(SELECT_WEAPON_SOUND);
140 }
```

jun 10, 18 14:54 **MusicPlayerException.cpp** Page 1/1

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

MusicPlayerException::MusicPlayerException(std::string msg) : msg(msg) {
    this->msg.insert(0, "Error en Music Player: ");
}

MusicPlayerException::~MusicPlayerException() {}

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

jun 10, 18 14:54 **MusicPlayerException.h** Page 1/1

```cpp
#ifndef __MUSICPLAYEREXCEPTION_H__
#define __MUSICPLAYEREXCEPTION_H__

#include <exception>
#include <string>

class MusicPlayerException : public std::exception {
private:
    std::string msg;

public:
    //Crea la excepcion
    explicit MusicPlayerException(std::string msg);

    //Destruye la excepcion
    virtual ~MusicPlayerException();

    //Devuelve el mensaje de error
    virtual const char* what() const noexcept;
};

#endif
```

jun 10, 18 14:54 **MusicPlayer.h** Page 1/2

```
#ifndef __MUSICPLAYER_H__
#define __MUSICPLAYER_H__

#include <SDL2/SDL.h>
#include <SDL2/SDL_mixer.h>
#include <map>
#include <string>

/* Clase que se enecarga de reproducir musica y efectos
 * de sonido */
class MusicPlayer {
private:
    Mix_Music* music; // Musica de fondo
    std::map<int, Mix_Chunk*> effects;

    /* Verifica si algunos efectos de la lista finalizaon y los
     * libera. AdemÃ¡s libera el efecto que se encuentre guardado
     * en la lista con clave channel */
    void check(int channel);

    /* Agrega un nuevo efecto a la lista y lo reproduce */
    void addEffect(const std::string& audio);

public:
    /* Constructor */
    MusicPlayer();

    /* Destructor */
    ~MusicPlayer();

    /* Reproduce la musica de fondo */
    void playMusic();

    /* Reproduce el sonido de inicio de turno */
    void playStartTurnSound();

    /* Reproduce el sonido de falta de tiempo */
    void playTickSound();

    /* Reproduce el sonido de muerte de un worm */
    void playDeathSound();

    /* Reproduce el sonido de daÃ±o recibido */
    void playDamageReceiveSound();

    /* Reproduce el sonido de la explosion */
    void playExplosionSound(const std::string& weapon);

    /* Reproduce el sonido de arma disparada */
    void playWeaponShotSound(const std::string& weapon);

    /* Reproduce el sonido de salto o rollback */
    void playJumpSound(char action);

    /* Reproduce el sonido de arma seleccionada */
    void playSelectWeaponSound();

    /* Reproduce el sonido de victoria */
    void playVictory();

    /* Reproduce el sonido de arma descargada */
    void playNoAmmo();

    /* Detiene la reproduccion de la musica de fondo */
    void stop();
};
```

jun 10, 18 14:54 **MusicPlayer.h** Page 2/2

```


#endif
```

jun 10, 18 15:36 **ExplosionView.cpp** Page 1/1

```cpp
#include "ExplosionView.h"
#include <gtkmm/image.h>
#include <glibmm/main.h>
#include "Path.h"

ExplosionView::ExplosionView(BulletView&& bullet) : bulletView(
        std::move(bullet)) {
    this->animation = Gdk::Pixbuf::create_from_file(EXPLOSION_ANIMATION);
    int width = this->animation->get_width();
    int height = this->animation->get_height();
    for (int i = 0; i < height / width; i++) {
        Glib::RefPtr<Gdk::Pixbuf> aux = Gdk::Pixbuf::create_subpixbuf(
                this->animation, 0, i * width, width, width);
        this->animation_vector.push_back(aux);
    }
    this->iter = this->animation_vector.begin();
}

ExplosionView::~ExplosionView() {}

ExplosionView::ExplosionView(ExplosionView&& other) :
        bulletView(std::move(other.bulletView)) {
    this->animation_vector = other.animation_vector;
    this->animation = other.animation;
    this->iter = this->animation_vector.begin();
}

bool ExplosionView::startCallBack() {
    Gtk::Image& image = (Gtk::Image&) this->bulletView.getWidget();
    image.set(*(this->iter));
    this->iter++;
    if (this->iter == this->animation_vector.end()) {
        this->bulletView.removeFromWorld();
        return false;
    }
    return true;
}

void ExplosionView::start() {
    Glib::signal_timeout().connect(
            sigc::mem_fun(*this, &ExplosionView::startCallBack), 40);
}

bool ExplosionView::hasFinished() {
    return this->iter == this->animation_vector.end();
}
```

jun 10, 18 15:36 **ExplosionView.h** Page 1/1

```cpp
#ifndef __CLIENTEXPLOSIONVIEW_H__
#define __CLIENTEXPLOSIONVIEW_H__

#include <vector>
#include <gdkmm/pixbuf.h>
#include "BulletView.h"

/* Clase que se encarga de reproducir la animacion de una explosion */
class ExplosionView {
private:
    BulletView bulletView;
    std::vector<Glib::RefPtr<Gdk::Pixbuf>> animation_vector;
    Glib::RefPtr<Gdk::Pixbuf> animation;
    std::vector<Glib::RefPtr<Gdk::Pixbuf>>::iterator iter;

    /* Callback de start */
    bool startCallBack();

public:
    /* Constructor */
    explicit ExplosionView(BulletView&& bullet);

    /* Destructor */
    ~ExplosionView();

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


    /* Realiza la animacion de la explosion */
    void start();

    /* Devuelve true si la animacion de la explosion finalizo */
    bool hasFinished();
};


#endif
```

jun 10, 18 15:35 **ExplosionViewList.cpp** Page 1/1

```cpp
#include "ExplosionViewList.h"
#include <list>

ExplosionViewList::ExplosionViewList() {}

ExplosionViewList::~ExplosionViewList() {}

void ExplosionViewList::check() {
    std::list<ExplosionView>::iterator iter;
    iter = this->animations.begin();
    while (iter != this->animations.end()) {
        if (iter->hasFinished()) {
            iter = this->animations.erase(iter);
        } else {
            ++iter;
        }
    }
}

void ExplosionViewList::addAndStart(ExplosionView&& animation) {
    this->check();
    this->animations.push_back(std::move(animation));
    this->animations.back().start();
}
```

jun 10, 18 15:37 **ExplosionViewList.h** Page 1/1

```cpp
#ifndef WORMS_EXPLOSIONVIEWLIST_H
#define WORMS_EXPLOSIONVIEWLIST_H

#include <list>
#include "ExplosionView.h"

/* Clase que se encarga de almacenar animaciones de explosiones */
class ExplosionViewList {
private:
    std::list<ExplosionView> animations;

    /* Verifica si alguna animacion de la lista finalizo y las
     * elimina de la lista */
    void check();

public:
    /* Constructor */
    ExplosionViewList();

    /* Destructor */
    ~ExplosionViewList();


    /* Agrega una animacion de explosion a la lista y la reproduce */
    void addAndStart(ExplosionView&& animation);
};


#endif //WORMS_EXPLOSIONVIEWLIST_H
```

jun 12, 18 0:20 **WalkingAnimation.cpp** Page 1/1

```cpp
#include "WalkingAnimation.h"
#include "Path.h"
#include "ObjectSizes.h"

#define DIR_RIGHT 1
#define DIR_LEFT -1

WalkingAnimation::WalkingAnimation(Gtk::Image* worm_image) :
        worm_image(worm_image), dir(DIR_RIGHT) {
    this->walk_image = Gdk::Pixbuf::create_from_file(WORMS_PATH + "walk.png");
    int width = this->walk_image->get_width();
    int height = this->walk_image->get_height();
    for (int i = 0; i < height / WORM_IMAGE_WIDTH; i++) {
        walk_queue.push(Gdk::Pixbuf::create_subpixbuf(this->walk_image, 0,
                                i * WORM_IMAGE_WIDTH, width, WORM_IMAGE_WIDTH));
    }
}

WalkingAnimation::~WalkingAnimation() {}

WalkingAnimation::WalkingAnimation(WalkingAnimation&& other) :
        walk_queue(std::move(other.walk_queue)),
        walk_image(std::move(other.walk_image)),
        worm_image(other.worm_image), dir(other.dir) {}

void WalkingAnimation::setMovementImage(char new_dir) {
    if (new_dir == this->dir) {
        this->walk_queue.push(std::move(this->walk_queue.front()));
        this->walk_queue.pop();
    }
    this->setStaticImage(new_dir);
}

void WalkingAnimation::setStaticImage(char new_dir) {
    this->dir = new_dir;
    this->worm_image->set(Gdk::Pixbuf::create_subpixbuf(this->walk_queue.back(),
                        WORM_IMAGE_WIDTH + this->dir * WORM_IMAGE_WIDTH, 0,
                        WORM_IMAGE_WIDTH, WORM_IMAGE_WIDTH));
}

void WalkingAnimation::updateWormImage(Gtk::Image* worm_image) {
    this->worm_image = worm_image;
}

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

jun 10, 18 15:32 **WalkingAnimation.h** Page 1/1

```cpp
#ifndef WORMS_WALKINGANIMATION_H
#define WORMS_WALKINGANIMATION_H

#include <gtkmm/image.h>
#include <gdkmm/pixbuf.h>
#include <queue>

/* Clase que se encarga de actualizar la imagen del worm al
 * moverse obteniendo una animacion del worm caminando */
class WalkingAnimation {
private:
    std::queue<Glib::RefPtr<Gdk::Pixbuf>> walk_queue;
    Glib::RefPtr<Gdk::Pixbuf> walk_image;
    Gtk::Image* worm_image;
    char dir;

public:
    /* Constructor*/
    explicit WalkingAnimation(Gtk::Image* worm_image);

    /* Destructor */
    ~WalkingAnimation();

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


    /* Actualiza la imagen del worm por la siguiente
     * imagen del worm caminando */
    void setMovementImage(char new_dir);

    /* Setea la imagen del worm por la imagen actual del
     * worm caminando */
    void setStaticImage(char new_dir);

    /* Devuelve la direccion del worm */
    char getDir() const;

    /* Actualiza el puntero de la imagen del worm */
    void updateWormImage(Gtk::Image* worm_image);
};


#endif //WORMS_WALKINGANIMATION_H
```

```cpp
#include "WeaponAnimation.h"
#include <glibmm/main.h>
#include <string>
#include <vector>
#include "WormView.h"
#include "Path.h"
#include "ObjectSizes.h"
#include "WeaponNames.h"

#define DIR_RIGHT 1

WeaponAnimation::WeaponAnimation(const std::string& weapon,
                                 Gtk::Image* worm_image) :
        worm_image(worm_image), angle(DEFAULT_ANGLE) {
    this->updateWeaponImage(weapon);
}

WeaponAnimation::~WeaponAnimation() {}

WeaponAnimation::WeaponAnimation(WeaponAnimation&& other) :
        scope_vector(std::move(other.scope_vector)),
        scope_image(std::move(other.scope_image)),
        worm_image(other.worm_image),
        angle(other.angle) {}

void WeaponAnimation::updateWeaponImage(const std::string& weapon) {
    this->scope_vector.clear();
    this->scope_image = Gdk::Pixbuf::create_from_file(
            WORMS_PATH + weapon + "_scope.png");
    int width = this->scope_image->get_width();
    int height = this->scope_image->get_height();
    for (int i = 0; i < height / WORM_IMAGE_WIDTH; i++) {
        this->scope_vector.push_back(
                Gdk::Pixbuf::create_subpixbuf(scope_image, 0,
                                              i * WORM_IMAGE_WIDTH, width,
                                              WORM_IMAGE_WIDTH));
    }
}

void WeaponAnimation::changeWeapon(const std::string& weapon, char dir) {
    this->updateWeaponImage(weapon);
    this->setWeaponImage(dir);
}

void WeaponAnimation::setWeaponImage(char dir) {
    int width = this->scope_vector[(90 + this->angle) / 6]->get_width() / 3;
    int height = this->scope_vector[(90 + this->angle) / 6]->get_height();
    this->worm_image->set(Gdk::Pixbuf::create_subpixbuf(
            this->scope_vector[(90 + this->angle) / 6], width + dir * width, 0,
            width, height));
}

bool WeaponAnimation::batHitCallBack(
        std::vector<Glib::RefPtr<Gdk::Pixbuf>>::iterator& iter, const int width,
        char dir) {
    this->worm_image->set(Gdk::Pixbuf::create_subpixbuf(*iter, 0, 0, width,
                                                        WORM_IMAGE_WIDTH));
    ++iter;
    if (iter == this->scope_vector.end()) {
        this->updateWeaponImage(BAT_NAME);
        this->setWeaponImage(dir);
        return false;
    }
    return true;
}

```



```cpp
void WeaponAnimation::weaponShootAnimation(const std::string& weapon, char dir){
    if (weapon != BAT_NAME) {
        return;
    }
    this->scope_image = Gdk::Pixbuf::create_from_file(BAT_HIT_ANIMATION);
    int width = this->scope_image->get_width() / 3;
    int height = this->scope_image->get_height();
    int pos_x = width + dir * width;
    this->scope_vector.clear();
    for (int i = 0; i < height / WORM_IMAGE_WIDTH; i++) {
        this->scope_vector.push_back(
                Gdk::Pixbuf::create_subpixbuf(scope_image, pos_x,
                                              i * WORM_IMAGE_WIDTH, width, WORM_IMAGE_WIDTH));
    }
    std::vector<Glib::RefPtr<Gdk::Pixbuf>>::iterator iter;
    iter = this->scope_vector.begin();
    sigc::slot<bool> my_slot = sigc::bind(
            sigc::mem_fun(*this, &WeaponAnimation::batHitCallBack), iter, width,
            dir);
    this->my_connection = Glib::signal_timeout().connect(my_slot, 12);
}

void WeaponAnimation::changeAngle(int angle, char dir) {
    this->angle = angle;
    this->setWeaponImage(dir);
}

void WeaponAnimation::updateWormImage(Gtk::Image* worm_image) {
    this->worm_image = worm_image;
}

void WeaponAnimation::stopShootAnimation() {
    if (this->my_connection.connected()) {
        this->my_connection.disconnect();
    }
}
```

jun 24, 18 18:03 **WeaponAnimation.h** Page 1/1

```
#ifndef WORMS_WEAPONANIMATION_H
#define WORMS_WEAPONANIMATION_H

#include <gtkmm/image.h>
#include <gdkmm/pixbuf.h>
#include <vector>
#include <string>

class WormView;

/* Clase que se encarga de controlar las animaciones
 * de las armas */
class WeaponAnimation {
private:
    std::vector<Glib::RefPtr<Gdk::Pixbuf>> scope_vector;
    Glib::RefPtr<Gdk::Pixbuf> scope_image;
    Gtk::Image* worm_image;
    int angle;
    sigc::connection my_connection;

    /* Actualiza las imagenes por las imagenes del arma nueva */
    void updateWeaponImage(const std::string& weapon);

    /* Callback */
    bool batHitCallBack(std::vector<Glib::RefPtr<Gdk::Pixbuf>>::iterator& iter,
                        const int width, char dir);

public:
    /* Constructor */
    WeaponAnimation(const std::string& weapon, Gtk::Image* worm_image);

    /* Destructor */
    ~WeaponAnimation();

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


    /* Cambia la imagen del worm con el arma actual por una imagen
     * del worm con la nueva arma */
    void changeWeapon(const std::string& weapon, char dir);

    /* Setea la imagen del worm con el arma actual apuntando
     * con el angulo especifico */
    void setWeaponImage(char dir);

    /* Realiza la animacion del disparo del arma */
    void weaponShootAnimation(const std::string& weapon, char dir);

    /* Actualiza el angulo, cambiando la imagen del arma
     * por la correspondiente */
    void changeAngle(int angle, char dir);

    /* Actualiza el puntero de la imagen del worm */
    void updateWormImage(Gtk::Image* worm_image);

    void stopShootAnimation();
};


#endif //WORMS_WEAPONANIMATION_H
```

jun 20, 18 18:11 **PlayerLifeLabel.cpp** Page 1/1

```
#include "PlayerLifeLabel.h"
#include <string>
#include "GamePlayers.h"

const std::string begining("<span color='");
const std::string middle("' font_family=\"monospace\"><b> ");
const std::string ending("</b></span>");

PlayerLifeLabel::PlayerLifeLabel() : id(0), player_name(""), life(0) {
    this->info_label.set_use_markup(true);
    this->id_label.set_use_markup(true);
    this->container.pack_start(this->id_label, Gtk::PACK_SHRINK);
    this->container.pack_start(this->info_label, Gtk::PACK_SHRINK);
}

PlayerLifeLabel::~PlayerLifeLabel() {}

void PlayerLifeLabel::setPlayerName(int id, const std::string& player_name) {
    this->id = id;
    this->player_name = player_name;
    this->updateLabel();
}

void PlayerLifeLabel::addLife(int life) {
    this->life += life;
    this->updateLabel();
}

void PlayerLifeLabel::reduceLife(int life) {
    this->life -= life;
    this->updateLabel();
}

Gtk::HBox& PlayerLifeLabel::getContainer() {
    return this->container;
}

void PlayerLifeLabel::updateLabel() {
    std::string id_message = begining + "white" + middle;
    id_message += std::to_string(this->id) + " " + ending;
    this->id_label.set_markup(id_message);
    this->id_label.override_background_color(Gdk::RGBA(colors[this->id]));
    std::string message = begining + "black" + middle + this->player_name;
    message += ": " + std::to_string(this->life) + ending;
    this->info_label.set_markup(message);
}
```

jun 19, 18 23:44 **PlayerLifeLabel.h** Page 1/1

```
#ifndef __PLAYERLIFELABEL_H__
#define __PLAYERLIFELABEL_H__

#include <gtkmm/label.h>
#include <gtkmm/hvbox.h>
#include <string>

/* Clase que se encarga de controlar el indicador de vida del jugador */
class PlayerLifeLabel {
private:
    int id;
    std::string player_name;
    int life;
    Gtk::HBox container;
    Gtk::Label info_label;
    Gtk::Label id_label;

    /* Actualiza la informacion del label */
    void updateLabel();

public:
    /* Constructor */
    PlayerLifeLabel();

    /* Destructor */
    ~PlayerLifeLabel();


    /* Establece el nombre del jugador */
    void setPlayerName(int id, const std::string& player_name);

    /* Agrega la vida al label */
    void addLife(int life);

    /* Disminuye la vida y actualiza la vista del label */
    void reduceLife(int life);

    /* Devuelve el contenedor con la informacion del jugador */
    Gtk::HBox& getContainer();
};


#endif
```

jun 19, 18 23:44 **PlayersList.cpp** Page 1/1

```
#include "PlayersList.h"
#include <glibmm/main.h>
#include <string>

#define SPACING 20

PlayersList::PlayersList() : container(false, SPACING) {
    this->title.set_use_markup(true);
    this->title.set_markup("<span><b><u>Jugadores</u></b></span>");
    this->container.pack_start(this->title, Gtk::PACK_SHRINK);
}

PlayersList::~PlayersList() {}

void PlayersList::addPlayer(int id, const std::string& name) {
    sigc::slot<bool> my_slot = sigc::bind(
            sigc::mem_fun(*this, &PlayersList::addPlayerCallBack), id, name);
    Glib::signal_idle().connect(my_slot);
}

bool PlayersList::addPlayerCallBack(int id, std::string name) {
    this->players[id] = name;
    this->labels[id].setPlayerName(id, name);
    this->container.pack_start(this->labels[id].getContainer(), Gtk::PACK_SHRINK
);
    return false;
}

const std::string& PlayersList::getPlayer(int id) const {
    return this->players.at(id);
}

Gtk::Container& PlayersList::getWindow() {
    return this->container;
}

void PlayersList::addPlayerLife(int player_id, int life) {
    this->labels[player_id].addLife(life);
}

void PlayersList::reducePlayerLife(int player_id, int life) {
    this->labels[player_id].reduceLife(life);
}
```

jun 12, 18 0:20 **PlayersList.h** Page 1/1

```
#ifndef __PLAYERSLIST_H__
#define __PLAYERSLIST_H__

#include <map>
#include <string>
#include <gtkmm/hvbox.h>
#include <gtkmm/label.h>
#include "PlayerLifeLabel.h"

/* Clase que se encarga de almacenar los nombres y las vidas
 * de todos los jugadores */
class PlayersList {
private:
    std::map<int, std::string> players;
    std::map<int, PlayerLifeLabel> labels;
    Gtk::VBox container;
    Gtk::Label title;

    bool addPlayerCallBack(int id, std::string name);

public:
    /* Constructor */
    PlayersList();

    /* Destructor */
    ~PlayersList();

    /* Agrega al jugador a la lista de jugadores y agrega su
     * informacion a la vista */
    void addPlayer(int id, const std::string& name);

    /* Devuelve el nombre del jugador */
    const std::string& getPlayer(int id) const;

    /* Devuelve el contenedor de los jugadores */
    Gtk::Container& getWindow();

    /* Agrega la informacion de la vida del jugador a la vista */
    void addPlayerLife(int player_id, int life);

    /* Reduce la vida del jugador y actualiza la vista */
    void reducePlayerLife(int player_id, int life);
};

#endif
```

jun 10, 18 15:29 **ScreenView.cpp** Page 1/2

```
#include "ScreenView.h"
#include "ServerFatalError.h"
#include <glibmm/main.h>
#include <string>

#define PADDING 10
#define SPACING 30

ScreenView::ScreenView(Gtk::Window& window, MenuView& main_menu, Player& player,
                       WeaponList& weapons) :
        left_view(false, SPACING), window(window),
        weapons_view(weapons, player),
        victory_view(window, main_menu) {
    this->left_view.pack_start(this->wind_view.getWindow(), Gtk::PACK_SHRINK);
    this->left_view.pack_start(this->players.getWindow(), Gtk::PACK_SHRINK);
    this->world_box.pack_start(this->left_view, Gtk::PACK_SHRINK, PADDING);
    this->world_box.pack_start(this->world.getContainer());
    this->world_box.pack_end(this->weapons_view.getWindow(), Gtk::PACK_SHRINK);

    this->screen.pack_start(this->turn_label.getWindow(), Gtk::PACK_SHRINK);
    this->screen.pack_end(this->world_box);
}

ScreenView::~ScreenView() {}

void ScreenView::show() {
    sigc::slot<bool> my_slot = sigc::mem_fun(*this, &ScreenView::showCallBack);
    Glib::signal_idle().connect(my_slot);
}

bool ScreenView::showCallBack() {
    this->weapons_view.update();
    this->window.remove();
    this->window.add(this->screen);
    this->window.show_all();
    return false;
}

void ScreenView::close() {
    sigc::slot<bool> my_slot = sigc::mem_fun(*this, &ScreenView::closeCallBack);
    Glib::signal_idle().connect(my_slot);
}

bool ScreenView::closeCallBack() {
    ServerFatalError error(this->window);
    return false;
}

WorldView& ScreenView::getWorld() {
    return this->world;
}

WeaponView& ScreenView::getWeaponsView() {
    return this->weapons_view;
}

TurnLabel& ScreenView::getTurnLabel() {
    return this->turn_label;
}

PlayersList& ScreenView::getPlayersView() {
    return this->players;
}

WindView& ScreenView::getWindView() {
    return this->wind_view;
```

jun 10, 18 15:29 **ScreenView.cpp** Page 2/2

```
67  }
68
69  void ScreenView::setWinner(const std::string& winner, bool i_win) {
70      this->victory_view.setWinner(winner, i_win);
71  }
```

jun 10, 18 15:30 **ScreenView.h** Page 1/2

```
1   #ifndef __CLIENTSCREENVIEW_H__
2   #define __CLIENTSCREENVIEW_H__
3
4   #include <gtkmm/hvbox.h>
5   #include <gtkmm/label.h>
6   #include <gtkmm/window.h>
7   #include <string>
8   #include "MenuView.h"
9   #include "WorldView.h"
10  #include "WeaponView.h"
11  #include "TurnLabel.h"
12  #include "PlayersList.h"
13  #include "WindView.h"
14  #include "VictoryWindow.h"
15
16  /* Clase que se encarga de almacenar los contenedores principales
17   * de la vista y mostrar su contenido */
18  class ScreenView {
19  private:
20      Gtk::VBox screen;
21      Gtk::HBox world_box;
22      Gtk::VBox left_view;
23      Gtk::Window& window;
24
25      WorldView world;
26      WeaponView weapons_view;
27      TurnLabel turn_label;
28      PlayersList players;
29      WindView wind_view;
30
31      VictoryWindow victory_view;
32
33      /* CallBacks */
34      bool showCallBack();
35
36      bool closeCallBack();
37
38  public:
39      /* Constructor */
40      ScreenView(Gtk::Window& window, MenuView& main_menu, Player& player,
41                 WeaponList& weapons);
42
43      /* Destructor */
44      ~ScreenView();
45
46      /* Muestra la pantalla en la ventana */
47      void show();
48
49      /* Cierra la ventana completamente */
50      void close();
51
52      /* Devuelve el WorldView */
53      WorldView& getWorld();
54
55      /* Devuelve el WeaponView */
56      WeaponView& getWeaponsView();
57
58      /* Devuelve el TurnLabel */
59      TurnLabel& getTurnLabel();
60
61      /* Devuelve el Players view */
62      PlayersList& getPlayersView();
63
64      /* Devuelve el wind view */
65      WindView& getWindView();
66
```

jun 10, 18 15:30 **ScreenView.h** Page 2/2

```
67      /* Muestra una ventana con el ganador */
68      void setWinner(const std::string& winner, bool i_win);
69  };
70
71  #endif
```

jun 10, 18 14:54 **TurnLabel.cpp** Page 1/1

```
1   #include "TurnLabel.h"
2   #include <string>
3
4   const std::string begining("<span size='20000'>");
5   const std::string ending("</span>");
6
7   TurnLabel::TurnLabel() {
8       this->message.set_use_markup(true);
9       this->message.set_markup(begining + "Worms" + ending);
10      this->label.pack_start(this->message);
11      this->label.pack_end(this->time);
12  }
13
14  TurnLabel::~TurnLabel() {}
15
16  void TurnLabel::beginTurn() {
17      std::string message = begining + "Tu turno" + ending;
18      this->message.set_markup(message);
19  }
20
21  void TurnLabel::beginTurn(const std::string& player_name) {
22      std::string message = begining + "Turno de " + player_name + ending;
23      this->message.set_markup(message);
24  }
25
26  void TurnLabel::endTurn() {
27      this->time.set_markup("");
28      this->message.set_markup(begining + "Termino el turno" + ending);
29  }
30
31  void TurnLabel::setTime(int time) {
32      this->time.set_markup(begining + std::to_string(time) + ending);
33  }
34
35  void TurnLabel::setEndGame() {
36      this->message.set_markup(begining + "Termino el juego" + ending);
37  }
38
39  Gtk::Container& TurnLabel::getWindow() {
40      return this->label;
41  }
```

jun 10, 18 15:26 **TurnLabel.h** Page 1/1

```
#ifndef __TURNLABEL_H__
#define __TURNLABEL_H__

#include <gtkmm/hvbox.h>
#include <gtkmm/label.h>
#include <string>

/* Clase que se encarga de controlar los labels que indican
 * el estado del turno */
class TurnLabel {
private:
    Gtk::Label message;
    Gtk::Label time;
    Gtk::HBox label;

public:
    /* Constructor */
    TurnLabel();

    /* Destructor */
    ~TurnLabel();


    /* Cambia el label indicando que es el turno del jugador */
    void beginTurn();

    /* Cambia el label indicando que es el turno del jugador
     * con nombre pasado por parametro */
    void beginTurn(const std::string& player_name);

    /* Cambia el label indicando que finalizo el turno del jugador */
    void endTurn();

    /* Cambia el label mostrando al ganador */
    void setEndGame();

    /* Cambia el label de tiempo al tiempo pasado por parametro */
    void setTime(int time);

    /* Devuelve el contenedor de la vista */
    Gtk::Container& getWindow();
};


#endif
```

jun 20, 18 18:46 **VictoryWindow.cpp** Page 1/2

```
#include "VictoryWindow.h"
#include <gtkmm/builder.h>
#include <string>
#include "Path.h"
#include "ButtonBuilder.h"

const std::string begining("<span color='black'");
const std::string medium(" font_family=\"monospace\" size='large'><b>");
const std::string end("</b></span>");

VictoryWindow::VictoryWindow(Gtk::Window& window, MenuView& main_menu) :
        window(window), main_menu(main_menu), was_closed(true) {
    Glib::RefPtr<Gtk::Builder> builder = Gtk::Builder::create_from_file(
            GLADE_PATH + "victory_window.glade");

    builder->get_widget("Menu", this->my_window);

    this->my_window->set_title(CLIENT_WINDOW_NAME);
    this->my_window->set_icon_from_file(ICON_PATH);

    builder->get_widget("victory_msg", victory_msg);
    this->victory_msg->set_use_markup(true);
    builder->get_widget("winner", winner);
    this->winner->set_use_markup(true);

    builder->get_widget("image", this->image);

    builder->get_widget("Return_menu", this->return_menu);
    builder->get_widget("quit", this->quit);

    ButtonBuilder::buildButton(this->quit);
    ButtonBuilder::buildButton(this->return_menu);

    this->return_menu->signal_clicked().connect(
            sigc::mem_fun(*this, &VictoryWindow::returnMenuButtonPressed));

    this->quit->signal_clicked().connect(
            sigc::mem_fun(*this, &VictoryWindow::quitButtonPressed));

    this->my_window->signal_delete_event().connect(
            sigc::mem_fun(*this, &VictoryWindow::on_delete_event));
}

VictoryWindow::~VictoryWindow() {}

bool VictoryWindow::on_delete_event(GdkEventAny* any_event) {
    gtk_widget_destroy((GtkWidget*) this->my_window->gobj());
    if (this->was_closed) {
        // Si se apreto el botÃ³n salir o el botÃ³n de cerrar
        this->window.close();
    }
    return true;
}

void VictoryWindow::returnMenuButtonPressed() {
    this->was_closed = false;
    this->my_window->close();
    this->window.remove();
    this->main_menu.addMenu();
}

void VictoryWindow::quitButtonPressed() {
    this->my_window->close();
}

void VictoryWindow::setWinner(const std::string& winner, bool i_win) {
```

jun 20, 18 18:46 **VictoryWindow.cpp** Page 2/2

```
67      std::string victory_message = begining + medium;
68      std::string winner_message = begining + medium;
69      if (winner.empty()) {
70          victory_message += "Empate";
71          this->image->set(TIE_IMAGE);
72      } else if (i_win) {
73          victory_message += "GANASTE!!!!";
74          this->image->set(WINNER_IMAGE);
75      } else {
76          victory_message += "Perdiste";
77          winner_message +=  "El ganador fue: " + winner;
78          this->image->set(LOSER_IMAGE);
79      }
80      victory_message += end;
81      winner_message += end;
82      this->victory_msg->set_markup(victory_message);
83      this->winner->set_markup(winner_message);
84      this->my_window->set_modal(true);
85      this->my_window->show_all();
86  }
```

jun 20, 18 18:29 **VictoryWindow.h** Page 1/1

```
1   #ifndef WORMS_VICTORYWINDOW_H
2   #define WORMS_VICTORYWINDOW_H
3
4   #include <gtkmm/window.h>
5   #include <gtkmm/button.h>
6   #include <gtkmm/label.h>
7   #include <gtkmm/image.h>
8   #include <string>
9   #include "MenuView.h"
10
11  /* Clase que se encarga de mostrar una ventana con
12   * un mensaje indicando el ganador de la partida cuando
13   * esta finaliza. */
14  class VictoryWindow {
15  private:
16      Gtk::Window* my_window;
17      Gtk::Window& window;
18      Gtk::Button* return_menu;
19      Gtk::Button* quit;
20      Gtk::Label* victory_msg;
21      Gtk::Label* winner;
22      Gtk::Image* image;
23      MenuView& main_menu;
24      bool was_closed;
25
26      /* Handler de la ventana al cerrarse */
27      bool on_delete_event(GdkEventAny* any_event);
28
29      /* Handler del boton de retorno al menu */
30      void returnMenuButtonPressed();
31
32      /* Handler del boton salir */
33      void quitButtonPressed();
34
35  public:
36      /* Constructor */
37      VictoryWindow(Gtk::Window& window, MenuView& main_menu);
38
39      /* Destructor */
40      ~VictoryWindow();
41
42
43      /* Establece el mensaje del ganador y muestra la ventana
44       * con este mensaje y los botones */
45      void setWinner(const std::string& winner, bool i_win);
46  };
47
48
49  #endif //WORMS_VICTORYWINDOW_H
```

jun 10, 18 15:29 **WeaponButton.cpp** Page 1/1

```cpp
#include "WeaponButton.h"
#include <string>
#include "Player.h"
#include "Path.h"

WeaponButton::WeaponButton(const std::string& weapon_name, unsigned int ammo,
                           Player& player) :
        weapon_name(weapon_name), player(player) {
    this->setLabel(ammo);
    std::string path = WEAPONS_PATH;
    path += weapon_name + ".png";
    this->image.set(path);
    this->button.set_image(this->image);
    this->button.set_always_show_image(true);
    this->button.signal_clicked().connect(
            sigc::mem_fun(*this, &WeaponButton::onClickedButton));
}

WeaponButton::~WeaponButton() {}

void WeaponButton::onClickedButton() {
    this->player.changeWeapon(weapon_name);
}

Gtk::Widget& WeaponButton::getButton() {
    return this->button;
}

void WeaponButton::setLabel(unsigned int ammo) {
    std::string label = "Ammo:\n  ";
    if (!ammo) {
        label += "0";
        button.set_sensitive(false);
    } else if (ammo > 100) {
        label += "âM−^HM−^^";
    } else {
        label += std::to_string(ammo);
    }
    this->button.set_label(label);
}

```

jun 10, 18 14:54 **WeaponButton.h** Page 1/1

```cpp
#ifndef __CLIENTWEAPONBUTTON_H__
#define __CLIENTWEAPONBUTTON_H__

#include <gtkmm/togglebutton.h>
#include <gtkmm/image.h>
#include <string>

class Player;

/* Clase que se encarga de mostrar el boton de un arma
 * junto con la informacion correspondiente a esa arma */
class WeaponButton {
private:
    std::string weapon_name;
    Player& player;
    Gtk::Button button;
    Gtk::Image image;

public:
    /* Constructor */
    WeaponButton(const std::string& weapon_name, unsigned int ammo,
                 Player& player);

    /* Destructor */
    ~WeaponButton();

    /* Devuelve el wiget del boton */
    Gtk::Widget& getButton();

    /* Setea el label del boton */
    void setLabel(unsigned int ammo);

    /* Handler del boton al ser clickeado */
    void onClickedButton();
};


#endif
```

jun 10, 18 15:25 **WeaponView.cpp** Page 1/1

```cpp
#include "WeaponView.h"
#include <glibmm/main.h>
#include <string>
#include <utility>
#include "Player.h"
#include "WeaponList.h"
#include "WeaponButton.h"

WeaponView::WeaponView(WeaponList& weapons, Player& player) :
        weapons(weapons), player(player) {}

WeaponView::~WeaponView() {}

void WeaponView::update() {
    WeaponList::iterator iter;
    int row = 1, column = 1;
    for (iter = this->weapons.begin(); iter != this->weapons.end(); iter++) {
        std::unique_ptr<WeaponButton> p(
                new WeaponButton(iter->second->getName(),
                                 iter->second->getAmmo(), this->player));
        this->buttons.insert(
                std::pair<std::string, std::unique_ptr<WeaponButton>>(
                        iter->second->getName(), std::move(p)));
        this->window.attach(
                this->buttons.at(iter->second->getName())->getButton(), column,
                row, 1, 1);
        row++;
    }
}

Gtk::Grid& WeaponView::getWindow() {
    return this->window;
}

void WeaponView::updateAmmo(const Weapon& weapon) {
    this->buttons[weapon.getName()]->setLabel(weapon.getAmmo());
}
```

jun 10, 18 14:54 **WeaponView.h** Page 1/1

```cpp
#ifndef __CLIENTWEAPONVIEW_H__
#define __CLIENTWEAPONVIEW_H__

#include <gtkmm/grid.h>
#include <unordered_map>
#include <memory>
#include <string>

class Player;

class WeaponList;

class WeaponButton;

class Weapon;

/* Clase que se encarga de mostrar los datos de las armas del juego
 * y de almacenar todos los botones de las armas */
class WeaponView {
private:
    WeaponList& weapons;
    Gtk::Grid window;
    Player& player;
    std::unordered_map<std::string, std::unique_ptr<WeaponButton>> buttons;

public:
    /* Constructor */
    WeaponView(WeaponList& weapons, Player& player);

    /* Destructor */
    ~WeaponView();


    /* Actualiza la informacion de todos los botones */
    void update();

    /* Actualiza la informacion de la municion del arma especifica */
    void updateAmmo(const Weapon& weapon);

    /* Devuelve el contenedor de la vista */
    Gtk::Grid& getWindow();
};

#endif
```

jun 10, 18 15:27 **WindView.cpp** Page 1/1

```cpp
#include "WindView.h"
#include <string>
#include "Path.h"

WindView::WindView() : container(false, 7) {
    this->container.pack_start(this->velocity, Gtk::PACK_SHRINK);
    this->container.pack_start(this->direction, Gtk::PACK_SHRINK);
    this->velocity.set_use_markup(true);
}

WindView::~WindView() {}

void WindView::update(float wind) {
    wind *= 10;
    std::string message = "<span><b><u>Viento</u></b>\n\n";
    std::string direction = "right";
    if (wind == 0) {
        direction = "no";
    } else if (wind < 0) {
        wind *= -1;
        direction = "left";
    }
    std::string velocity = std::to_string(wind);
    message += velocity.substr(0, 4) + "</span>";
    this->velocity.set_markup(message);
    this->direction.set(IMAGES_PATH + "arrow_" + direction + ".png");
}

Gtk::VBox& WindView::getWindow() {
    return this->container;
}
```

jun 10, 18 14:54 **WindView.h** Page 1/1

```cpp
#ifndef __WINDVIEW_H__
#define __WINDVIEW_H__

#include <gtkmm/hvbox.h>
#include <gtkmm/label.h>
#include <gtkmm/image.h>


/* Clase que se encarga de mostrar y actualizar
 * la informacion del viento */
class WindView {
private:
    Gtk::VBox container;
    Gtk::Label velocity;
    Gtk::Image direction;

public:
    /* Constructor */
    WindView();

    /* Destructor */
    ~WindView();

    /* Actualiza la vista del viento */
    void update(float wind);

    /* Devuelve el contenedor del viento */
    Gtk::VBox& getWindow();
};

#endif

```

```cpp
#include "WorldView.h"
#include <gtkmm/adjustment.h>
#include <glibmm/main.h>
#include <giomm/memoryinputstream.h>
#include "ViewPositionTransformer.h"
#include "Player.h"
#include "Math.h"
#include "Path.h"
#include "ObjectSizes.h"

WorldView::WorldView() {
    this->container.add_overlay(this->background);
    this->world.set_size(map_width, map_height);
    this->window.add_events(Gdk::BUTTON_PRESS_MASK);
    this->window.add(this->world);
    this->container.add_overlay(this->window);

    this->water.show(this->world);
    this->window.get_hadjustment()->set_value(map_width / 2);
    this->window.get_vadjustment()->set_value(map_height);
}

WorldView::~WorldView() {}

void WorldView::moveElement(Viewable& element, const Position& position) {
    float width = element.getWidth();
    float height = element.getHeight();
    Gtk::Widget& widget = element.getWidget();
    Position newPosition = ViewPositionTransformer(
            this->world).transformToScreenAndMove(position, width, height);
    this->world.move(widget, newPosition.getX(), newPosition.getY());
    if (element.hasFocus()) {
        this->setFocus(widget);
    }
}

void WorldView::moveScope(Gtk::Widget& scope, Gtk::Widget& worm, int angle) {
    float pos_x = this->world.child_property_x(worm).get_value();
    float pos_y = this->world.child_property_y(worm).get_value();
    pos_x += 50 * Math::cosDegrees(angle);
    pos_y -= 50 * Math::sinDegrees(angle);
    // Para que quede referenciado a la mitad de la imagen
    pos_x -= worm.get_width() / 2;
    this->world.move(scope, pos_x, pos_y);
}

void WorldView::removeElement(Gtk::Widget& element) {
    this->world.remove(element);
}

void WorldView::addElement(Viewable& element, const Position& position) {
    float width = element.getWidth();
    float height = element.getHeight();
    Gtk::Widget& widget = element.getWidget();
    Position newPosition = ViewPositionTransformer(
            this->world).transformToScreenAndMove(position, width, height);
    this->world.put(widget, newPosition.getX(), newPosition.getY());
    widget.show_all();
    if (element.hasFocus()) {
        this->setFocus(widget);
    }
}

Gtk::ScrolledWindow& WorldView::getWindow() {
    return this->window;
}
```



```cpp

Gtk::Layout& WorldView::getLayout() {
    return this->world;
}

void WorldView::setFocus(Gtk::Widget& element) {
    this->window.get_hadjustment()->set_value(element.get_allocation().get_x() -
                        this->window.get_hadjustment()->get_page_size() / 2);
    this->window.get_vadjustment()->set_value(element.get_allocation().get_y() -
                        this->window.get_vadjustment()->get_page_size() / 2);
}

void WorldView::setBackgroundImage(const Buffer& image) {
    sigc::slot<bool> my_slot = sigc::bind(
            sigc::mem_fun(*this, &WorldView::setBackgroundImageCallBack),
            image);
    Glib::signal_idle().connect(my_slot);
}

bool WorldView::setBackgroundImageCallBack(Buffer image) {
    auto screen = this->container.get_screen();
    size_t screen_width = screen->get_width();
    size_t screen_height = screen->get_height();
    auto pixbuf = Gio::MemoryInputStream::create();
    pixbuf->add_data(image.getPointer(), image.getMaxSize());
    auto aux = Gdk::Pixbuf::create_from_stream(pixbuf);
    size_t img_width = aux->get_width();
    size_t img_height = aux->get_height();
    for (size_t x = 0; x < screen_width; x += img_width) {
        for (size_t y = 0; y < screen_height; y += img_height) {
            Gtk::Image background_image(aux);
            background_image.show();
            this->background.put(background_image, x, y);
            this->background_images.push_back(std::move(background_image));
        }
    }
    return false;
}

Gtk::Container& WorldView::getContainer() {
    return this->container;
}
```

```
1  #ifndef __WORLDVIEW_H__
2  #define __WORLDVIEW_H__
3
4  #include <gtkmm/widget.h>
5  #include <gtkmm/layout.h>
6  #include <gtkmm/hvbox.h>
7  #include <gtkmm/scrolledwindow.h>
8  #include <gtkmm/overlay.h>
9  #include <string>
10 #include <vector>
11 #include "Viewable.h"
12 #include "Position.h"
13 #include "Water.h"
14 #include "Buffer.h"
15
16 class Player;
17
18 /* Clase que se encarga de mostrar objetos en posiciones
19  * especificas, moverlos y eliminarlos de la vista*/
20 class WorldView {
21 private:
22     Gtk::Overlay container;
23     Gtk::Layout background;
24     Gtk::Layout world;
25     Gtk::ScrolledWindow window;
26     std::vector<Gtk::Image> background_images;
27     Water water;
28
29     /* Coloca la imagen de fondo */
30     bool setBackgroundImageCallBack(Buffer image);
31
32 public:
33     /* Constructor */
34     WorldView();
35
36     /* Destructor */
37     ~WorldView();
38
39     /* Setea la imagen de fondo */
40     void setBackgroundImage(const Buffer& image);
41
42     /* Mueve el elemento pasado a la posicion especificada */
43     void moveElement(Viewable& element, const Position& position);
44
45     /* Mueve la mira a la posicion correspondiente para que tenga el angulo
46      * especificado por parametro */
47     void moveScope(Gtk::Widget& scope, Gtk::Widget& worm, int angle);
48
49     /* Remueve el elemento de la vista */
50     void removeElement(Gtk::Widget& element);
51
52     /* Agrega un elemento a la vista en la posicion especificada */
53     void addElement(Viewable& element, const Position& position);
54
55     /* Devuelve la vista del scrolledWindow */
56     Gtk::ScrolledWindow& getWindow();
57
58     /* Devuelve el container */
59     Gtk::Container& getContainer();
60
61     /* Devuelve la vista del Layout */
62     Gtk::Layout& getLayout();
63
64     /* Realiza focus en el elemento pasado */
65     void setFocus(Gtk::Widget& element);
66 };
```



```
67
68
69 #endif
```

jun 20, 18 12:44 **BulletView.cpp** Page 1/1

```cpp
#include "BulletView.h"
#include <string>
#include "ObjectSizes.h"

BulletView::BulletView(WorldView& worldView, std::string weapon, Position pos) :
        Viewable(worldView), weapon_name(std::move(weapon)) {
    std::string path(BULLETS_PATH);
    path += this->weapon_name;
    path += ".png";
    this->image.set(path);
    this->addToWorld(pos);
}

BulletView::~BulletView() {}

BulletView::BulletView(BulletView&& other) :
        Viewable(std::move(other)), image(std::move(other.image)),
        weapon_name(std::move(other.weapon_name)) {}

void BulletView::updateData(const Position& new_pos) {
    this->move(new_pos);
}

Gtk::Widget& BulletView::getWidget() {
    return this->image;
}

float BulletView::getWidth() const {
    return weapon_size;
}

float BulletView::getHeight() const {
    return weapon_size;
}

std::string BulletView::getName() {
    return this->weapon_name;
}
```

jun 20, 18 12:44 **BulletView.h** Page 1/1

```cpp
#ifndef __CLIENTBULLETVIEW_H__
#define __CLIENTBULLETVIEW_H__

#include <gtkmm/widget.h>
#include <gtkmm/image.h>
#include <string>
#include "Viewable.h"

/* Clase que se encarga de controlar la vista de las balas */
class BulletView : public Viewable {
private:
    Gtk::Image image;
    std::string weapon_name;

public:
    /* Constructor */
    BulletView(WorldView& worldView, std::string weapon, Position pos);

    /* Destructor */
    ~BulletView();

    /* Constructor por movimient */
    BulletView(BulletView&& other);

    /* Actualiza la posicion de la bala en la vista */
    void updateData(const Position& new_pos);

    /* Devuelve el contenedor de la bala */
    Gtk::Widget& getWidget() override;

    /* Devuelve el ancho de la bala */
    float getWidth() const override;

    /* Devuelve el alto de la bala */
    float getHeight() const override;

    /* Devuelve el nombre del arma de la bala */
    std::string getName();
};


#endif
```

jun 20, 18 12:44 **GirderView.cpp** Page 1/1

```cpp
#include "GirderView.h"
#include <string>
#include "GirderSize.h"

GirderView::GirderView(WorldView& worldView, size_t size, Position pos,
                       int rotation) :
        Viewable(worldView), size(size), rotation(rotation) {
    std::string path(GIRDER_PATH);
    path += std::to_string(size);
    path += "_";
    path += std::to_string(rotation);
    path += ".png";
    this->image.set(path);
    this->addToWorld(pos);
}

GirderView::~GirderView() {}

GirderView::GirderView(GirderView&& other) : Viewable(std::move(other)),
        image(std::move(other.image)), size(other.size),
        rotation(other.rotation) {}

Gtk::Widget& GirderView::getWidget() {
    return this->image;
}

float GirderView::getWidth() const {
    return GirderSize::getGirderWidthMeters(this->size, this->rotation);
}

float GirderView::getHeight() const {
    return GirderSize::getGirderHeightMeters(this->size, this->rotation);
}
```

jun 20, 18 12:44 **GirderView.h** Page 1/1

```cpp
#ifndef __GIRDERVIEW_H__
#define __GIRDERVIEW_H__

#include <gtkmm/widget.h>
#include <gtkmm/image.h>
#include <string>
#include "Viewable.h"

/* Clase que se encaga de controlar la vista de las vigas */
class GirderView : public Viewable {
private:
    Gtk::Image image;
    int size;
    int rotation;

public:
    /* Constructor */
    GirderView(WorldView& worldView, size_t size, Position pos, int rotation);

    /* Destructor */
    ~GirderView();

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

    /* Devuelve el contenedor de la viga */
    Gtk::Widget& getWidget() override;

    /* Devuelve el ancho de la viga */
    float getWidth() const override;

    /* Devuelve el alto de la viga */
    float getHeight() const override;
};


#endif
```

jun 20, 18 13:14 **Scope.cpp** Page 1/1

```cpp
#include "Scope.h"
#include "Path.h"
#include "WorldView.h"
#include "WeaponNames.h"
#include "ObjectSizes.h"

Scope::Scope(WorldView& world) : Viewable(world) {
    this->scope.set(SCOPE_IMAGE);
    this->angle = DEFAULT_ANGLE;
    this->addToWorld(Position(0, 0));
}

Scope::~Scope() {}

void Scope::update(int angle, WormView& worm) {
    this->angle = angle;
    char dir = worm.getDir();
    if (dir == DIR_LEFT)
        angle = 180 - angle;
    this->worldView.moveScope(this->scope, worm.getWidget(), angle);
    this->scope.show();
    worm.updateScope(this->angle);
}

void Scope::update(WormView& worm) {
    this->update(this->angle, worm);
}

void Scope::hide() {
    if (this->scope.is_visible()) {
        this->scope.hide();
    }
}

Gtk::Widget& Scope::getWidget() {
    return this->scope;
}

float Scope::getWidth() const {
    return scope_size;
}

float Scope::getHeight() const {
    return scope_size;
}
```

jun 20, 18 13:09 **Scope.h** Page 1/1

```cpp
#ifndef __SCOPE_H__
#define __SCOPE_H__

#include <gtkmm/image.h>
#include "Viewable.h"
#include "WormView.h"

/* Clase que se encarga de controlar la imagen
 * de la mira del arma */
class Scope : public Viewable {
private:
    Gtk::Image scope;
    int angle;

public:
    /* Constructor */
    explicit Scope(WorldView& world);

    /* Destructor */
    ~Scope();

    /* Actualiza la posicion del scope */
    void update(int angle, WormView& worm);

    /* Actualiza la posicion del scope */
    void update(WormView& worm);

    /* Esconde el scope */
    void hide();

    /* Devuelve el contenedor del scope */
    Gtk::Widget& getWidget() override;

    /* Devuelve el ancho del scope */
    float getWidth() const override;

    /* Devuelve el alto del scope */
    float getHeight() const override;
};

#endif
```

jun 20, 18 13:15 **Viewable.cpp** Page 1/1

```cpp
#include "Viewable.h"
#include "WorldView.h"


Viewable::Viewable(WorldView& worldView) : has_focus(false),
                                           worldView(worldView) {}

Viewable::~Viewable() {}

Viewable::Viewable(Viewable&& other) : has_focus(other.has_focus),
                                       worldView(other.worldView) {}

void Viewable::move(const Position& pos) {
    this->worldView.moveElement(*this, pos);
}

void Viewable::removeFromWorld() {
    this->worldView.removeElement(this->getWidget());
}

void Viewable::addToWorld(const Position& pos) {
    this->worldView.addElement(*this, pos);
}

void Viewable::setFocus(bool focus) {
    this->has_focus = focus;
}

bool Viewable::hasFocus() const {
    return this->has_focus;
}
```

jun 20, 18 13:09 **Viewable.h** Page 1/1

```cpp
#ifndef __VIEWABLE_H__
#define __VIEWABLE_H__

#include <gtkmm/widget.h>
#include "Position.h"
#include "Path.h"

class WorldView;

/* Clase que se encarga de controlar los objetos visuales */
class Viewable {
private:
    bool has_focus;

protected:
    WorldView& worldView;

    /* Agrega al objeto visual a la vista */
    void addToWorld(const Position& pos);

    /* Mueve al objeto visual a la posicion especificada */
    void move(const Position& pos);

public:
    /* Constructor */
    explicit Viewable(WorldView& worldView);

    /* Destructor */
    virtual ~Viewable();

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

    /* Devuelve el contenedor del objeto visual */
    virtual Gtk::Widget& getWidget() = 0;

    /* Remueve al objeto visual de la vista */
    void removeFromWorld();

    /* Establece si al objeto visual se le puede hacer focus o no */
    void setFocus(bool focus);

    /* Devuelve true si el objeto visual es focuseable */
    bool hasFocus() const;

    /* Devuelve el ancho del viewable */
    virtual float getWidth() const = 0;

    /* Devuelve el alto del viewable */
    virtual float getHeight() const = 0;
};

#endif
```

jun 10, 18 15:41 **WormLifeView.cpp** Page 1/1

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

const std::string begining("<span color='white'><b>");
const std::string ending("</b></span>");

WormLifeView::WormLifeView(int life, const std::string& color) : color(color) {
    this->label.set_use_markup(true);
    this->updateLife(life);
}

WormLifeView::~WormLifeView() {}

WormLifeView::WormLifeView(WormLifeView&& other) :
        label(std::move(other.label)), color(std::move(other.color)) {}

void WormLifeView::updateLife(int life) {
    this->label.override_background_color(Gdk::RGBA(this->color));
    this->label.set_markup(begining + std::to_string(life) + ending);
}

Gtk::Widget& WormLifeView::getWidget() {
    return this->label;
}
```

jun 10, 18 15:38 **WormLifeView.h** Page 1/1

```cpp
#ifndef __WORMLIFEVIEW_H__
#define __WORMLIFEVIEW_H__

#include <gtkmm/label.h>
#include <string>

/* Clase que se encarga de controlar el label de la vida
 * del worm */
class WormLifeView {
private:
    Gtk::Label label;
    std::string color;

public:
    /* Constructor */
    WormLifeView(int life, const std::string& color);

    /* Destructor */
    ~WormLifeView();

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

    /* Actualiza el label de vida del worm */
    void updateLife(int life);

    /* Devuelve el contenedor de la vida */
    Gtk::Widget& getWidget();
};


#endif
```

```cpp
#include "WormView.h"
#include <string>
#include <glibmm/main.h>
#include "ObjectSizes.h"
#include "WeaponNames.h"
#include "GamePlayers.h"

WormView::WormView(WorldView& worldView, int life, char dir, Position pos,
                   int player_id) :
        Viewable(worldView), player_id(player_id), life(life), is_moving(false),
        last_position(Position(-1, -1)), label(life, colors[player_id]),
        walkingAnimation(&this->image),
        weaponAnimation(DEFAULT_WEAPON, &this->image) {
    this->worm.attach(this->label.getWidget(), 0, 0, 1, 1);
    this->worm.attach(this->image, 0, 1, 1, 1);
    this->walkingAnimation.setStaticImage(DIR_RIGHT);
    this->addToWorld(pos);
}

WormView::~WormView() {
    this->weaponAnimation.stopShootAnimation();
}

WormView::WormView(WormView&& other) : Viewable(std::move(other)),
        player_id(other.player_id), life(other.life),
        is_moving(other.is_moving), last_position(other.last_position),
        label(std::move(other.label)), image(std::move(other.image)),
        worm(std::move(other.worm)),
        walkingAnimation(std::move(other.walkingAnimation)),
        weaponAnimation(std::move(other.weaponAnimation)) {
    this->weaponAnimation.updateWormImage(&this->image);
    this->walkingAnimation.updateWormImage(&this->image);
}

void WormView::updateData(int new_life, char new_dir, const Position& new_pos,
                          bool colliding, bool is_current_worm, bool has_shot) {
    if (new_life != this->life) {
        this->label.updateLife(new_life);
    }
    this->life = new_life;
    this->is_moving = !(this->last_position == new_pos);
    this->last_position = new_pos;
    this->setNewImage(new_dir, colliding, is_current_worm, has_shot);
    this->move(new_pos);
}

void WormView::updateScope(int angle) {
    this->weaponAnimation.changeAngle(angle, this->getDir());
}

void WormView::changeWeapon(const std::string& weapon) {
    this->weaponAnimation.changeWeapon(weapon, this->getDir());
}

void WormView::setNewImage(char dir, bool colliding, bool is_current_worm,
                           bool has_shot) {
    this->walkingAnimation.setStaticImage(dir);
    if (is_current_worm) {
        if (!this->is_moving && !has_shot && colliding) {
            this->weaponAnimation.setWeaponImage(dir);
        } else if (colliding) {
            this->walkingAnimation.setMovementImage(dir);
        }
    }
}

```



```cpp
Gtk::Widget& WormView::getWidget() {
    return this->worm;
}

float WormView::getWidth() const {
    return worm_size;
}

float WormView::getHeight() const {
    return worm_size + 0.5;
}

Gtk::Image& WormView::getImage() {
    return this->image;
}

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

char WormView::getDir() const {
    return this->walkingAnimation.getDir();
}

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

bool WormView::isMoving() const {
    return this->is_moving;
}

void WormView::setVictory() {
    this->image.set(VICTORY_ANIMATION);
}

void WormView::weaponShoot(const std::string& weapon) {
    this->weaponAnimation.weaponShootAnimation(weapon, this->getDir());
}

void WormView::resetFocus() {
    this->is_moving = false;
    this->setFocus(false);
    this->walkingAnimation.setStaticImage(this->getDir());
}
```

```
#ifndef __WORMVIEW_H__
#define __WORMVIEW_H__

#include <gtkmm/widget.h>
#include <gtkmm/image.h>
#include <gtkmm/grid.h>
#include <gdkmm/pixbuf.h>
#include <vector>
#include <string>
#include "Viewable.h"
#include "WormLifeView.h"
#include "WalkingAnimation.h"
#include "WeaponAnimation.h"

#define DIR_RIGHT 1
#define DIR_LEFT -1

/* Clase que se encarga de controlar la vista de los worms */
class WormView : public Viewable {
private:
    int player_id;
    int life;
    bool is_moving;
    Position last_position;
    WormLifeView label;
    Gtk::Image image;
    Gtk::Grid worm;
    WalkingAnimation walkingAnimation;
    WeaponAnimation weaponAnimation;

    /* Actualiza la imagen del worm a la correspondiente segun las
     * condiciones en las que se encuentra este */
    void
    setNewImage(char dir, bool colliding, bool is_current_worm, bool has_shot);

public:
    /* Constructor */
    WormView(WorldView& worldView, int life, char dir, Position pos,
             int player_id);

    /* Destructor */
    ~WormView();

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


    /* Actualiza la posicion y vida del worm */
    void updateData(int new_life, char new_dir, const Position& new_pos,
                    bool colliding, bool is_current_worm, bool has_shot);

    /* Actualiza la imagen del arma con el angulo actual */
    void updateScope(int angle);

    /* Actualiza el arma del worm y cambia la imagen */
    void changeWeapon(const std::string& weapon);

    /* Devuelve la direccion del worm */
    char getDir() const;

    /* Elimina la imagen del arma del worm */
    void removeWeaponImage();

    /* Devuelve la vida del worm */
    int getLife() const;

```



```
    /* Devuelve el id del player que controla al worm */
    int getPlayerId() const;

    /* Devuelve el contenedor donde se encuentra la vista del worm */
    Gtk::Widget& getWidget() override;

    /* Devuelve el ancho del worm */
    float getWidth() const override;

    /* Devuelve el alto del worm */
    float getHeight() const override;

    /* Devuelve la imagen que contiene al worm */
    Gtk::Image& getImage();

    /* Cambia la imagen del worm por la animacion del worm
     * festejando la victoria */
    void setVictory();

    /* Devuelve true si el gusano se esta moviendo */
    bool isMoving() const;

    /* Realiza la animacion del disparo del arma */
    void weaponShoot(const std::string& weapon);

    /* Resetea el focus del gusano */
    void resetFocus();
};


#endif
```

```
1  #include "ViewsList.h"
2  #include <glibmm/main.h>
3  #include <string>
4  #include "ObjectSizes.h"
5  #include "WeaponNames.h"
6  #include "Player.h"
7
8  ViewsList::ViewsList(WorldView& world, Player& player,
9                       PlayersList& players_list, MusicPlayer& musicPlayer) :
10         world(world), player(player), players_list(players_list), scope(world),
11         musicPlayer(musicPlayer) {
12     this->current_worm_id = -1;
13     this->weapon_focused = -1;
14     this->worm_focused = -1;
15 }
16
17 ViewsList::~ViewsList() {}
18
19
20 void ViewsList::removeWorm(int id) {
21     std::unordered_map<int, WormView>::iterator it = this->worms.find(id);
22     if (it != this->worms.end()) {
23         this->players_list.reducePlayerLife(it->second.getPlayerId(),
24                                             it->second.getLife());
25         it->second.removeFromWorld();
26         this->worms.erase(it);
27         this->musicPlayer.playDeathSound();
28         this->checkMovingWorms();
29     }
30 }
31
32 void ViewsList::removeWeapon(int id) {
33     std::unordered_map<int, BulletView>::iterator it = this->weapons.find(id);
34     if (it != this->weapons.end()) {
35         if (it->second.getName() != BAT_NAME) {
36             this->musicPlayer.playExplosionSound(it->second.getName());
37             ExplosionView explosion(std::move(it->second));
38             this->animation.addAndStart(std::move(explosion));
39         }
40         this->weapons.erase(it);
41
42         if (this->weapon_focused == id) {
43             this->weapon_focused = -2;
44             this->checkMovingWorms();
45         }
46     }
47 }
48
49 void ViewsList::updateWormData(int id, int player_id, float pos_x, float pos_y,
50                                int life, char dir, bool colliding) {
51     std::unordered_map<int, WormView>::iterator it = this->worms.find(id);
52     Position pos(pos_x / UNIT_TO_SEND, pos_y / UNIT_TO_SEND);
53     if (it == this->worms.end()) {
54         //Worm no existe
55         WormView worm(this->world, life, dir, pos, player_id);
56         this->worms.insert(std::make_pair(id, std::move(worm)));
57         this->players_list.addPlayerLife(player_id, life);
58     } else {
59         //Worm existe
60         int current_life = it->second.getLife();
61         if (current_life != life) {
62             this->players_list.reducePlayerLife(player_id, current_life - life);
63             if (id == this->current_worm_id) {
64                 this->musicPlayer.playDamageReceiveSound();
65             }
66         }
```



```
67         it->second.updateData(life, dir, pos, colliding,
68                               id == this->current_worm_id,
69                               this->weapon_focused != -1);
70         this->checkMovingWorms();
71     }
72 }
73
74 void
75 ViewsList::updateWeaponData(int id, const std::string& weapon_name, float pos_x,
76                             float pos_y) {
77     std::unordered_map<int, BulletView>::iterator it = this->weapons.find(id);
78     Position pos(pos_x / UNIT_TO_SEND, pos_y / UNIT_TO_SEND);
79     if (it == this->weapons.end()) {
80         //Weapon no existe
81         BulletView weapon(this->world, weapon_name, pos);
82         if (this->weapon_focused < 0) {
83             weapon.setFocus(true);
84             this->weapon_focused = id;
85             this->removeWormFocus();
86         }
87         this->weapons.insert(std::make_pair(id, std::move(weapon)));
88     } else {
89         //Weapon existe
90         it->second.updateData(pos);
91     }
92 }
93
94 void ViewsList::changeWeapon(const std::string& weapon_name) {
95     std::unordered_map<int, WormView>::iterator it = this->worms.find(this->curr
    ent_worm_id);
96     it->second.changeWeapon(weapon_name);
97     if (WeaponsFactory().createWeapon(weapon_name, 1)->hasScope()) {
98         this->scope.update(it->second);
99     }
100 }
101
102 void ViewsList::updateScope(int angle) {
103     std::unordered_map<int, WormView>::iterator it = this->worms.find(this->curr
    ent_worm_id);
104     if (it == this->worms.end()) {
105         return;
106     }
107     this->scope.update(angle, it->second);
108 }
109
110 void ViewsList::removeScopeVisibility() {
111     this->scope.hide();
112 }
113
114 bool ViewsList::addGirderCallBack(size_t size, Position pos, int rotation) {
115     GirderView girder(this->world, size, pos, rotation);
116     this->girders.push_back(std::move(girder));
117     return false;
118 }
119
120 void ViewsList::addGirder(size_t size, float pos_x, float pos_y, int rotation) {
121     sigc::slot<bool> my_slot = sigc::bind(
122             sigc::mem_fun(*this, &ViewsList::addGirderCallBack), size,
123             Position(pos_x, pos_y), rotation);
124     Glib::signal_idle().connect(my_slot);
125 }
126
127 void ViewsList::setCurrentWorm(int id) {
128     this->removeWormFocus();
129     std::unordered_map<int, WormView>::iterator it;
130     for (it = this->worms.begin(); it != this->worms.end(); ++it) {
```

jun 20, 18 18:42 **ViewsList.cpp** Page 3/3

```
131         it->second.resetFocus();
132     }
133     this->current_worm_id = id;
134     this->worm_focused = id;
135     this->weapon_focused = -1;
136     WormView& worm = this->worms.at(id);
137     this->world.setFocus(worm.getWidget());
138     worm.setFocus(true);
139 }
140
141 void ViewsList::removeWormFocus() {
142     std::unordered_map<int, WormView>::iterator it = this->worms.find(this->worm
    _focused);
143     if (it != this->worms.end()) {
144         it->second.resetFocus();
145     }
146     this->worm_focused = -1;
147 }
148
149 void ViewsList::checkMovingWorms() {
150     if (this->weapon_focused != -2) {
151         return;
152     }
153
154     std::unordered_map<int, WormView>::iterator it = this->worms.find(this->worm
    _focused);
155     if (it == this->worms.end() || !it->second.isMoving()) {
156         this->removeWormFocus();
157         for (auto it2 = this->worms.begin(); it2 != this->worms.end(); ++it2) {
158             if (it2->second.isMoving()) {
159                 this->worm_focused = it2->first;
160                 it2->second.setFocus(true);
161                 this->world.setFocus(it2->second.getWidget());
162                 return;
163             }
164         }
165     }
166 }
167
168 void ViewsList::setVictory() {
169     if (this->worms.empty()) {
170         return;
171     }
172     std::unordered_map<int, WormView>::iterator iter;
173     for (iter = this->worms.begin(); iter != this->worms.end(); iter++) {
174         this->musicPlayer.playVictory();
175         iter->second.setVictory();
176         this->world.setFocus(iter->second.getWidget());
177     }
178 }
179
180 void ViewsList::shoot(const std::string& weapon) {
181     this->worms.at(this->current_worm_id).weaponShoot(weapon);
182 }
```

jun 10, 18 15:09 **ViewsList.h** Page 1/2

```
1  #ifndef __VIEWSLIST_H__
2  #define __VIEWSLIST_H__
3
4  #include <unordered_map>
5  #include <vector>
6  #include <string>
7  #include "WorldView.h"
8  #include "WormView.h"
9  #include "BulletView.h"
10 #include "GirderView.h"
11 #include "PlayersList.h"
12 #include "ExplosionView.h"
13 #include "ExplosionViewList.h"
14 #include "MusicPlayer.h"
15 #include "Scope.h"
16
17 /* Clase que se encarga de almacenar los objetos visibles */
18 class ViewsList {
19 private:
20     WorldView& world;
21     Player& player;
22     PlayersList& players_list;
23     std::unordered_map<int, WormView> worms;
24     std::unordered_map<int, BulletView> weapons;
25     std::vector<GirderView> girders;
26     int current_worm_id;
27     int weapon_focused;
28     int worm_focused;
29     ExplosionViewList animation;
30     Scope scope;
31     MusicPlayer& musicPlayer;
32
33     /* Elimina el focus sobre el worm */
34     void removeWormFocus();
35
36     /* CallBacks */
37     bool addGirderCallBack(size_t size, Position pos, int rotation);
38
39 public:
40     /* Constructor */
41     ViewsList(WorldView& world, Player& player, PlayersList& players_list,
42               MusicPlayer& musicPlayer);
43
44     /* Destructor */
45     ~ViewsList();
46
47     /* Elimina al worm de la vista actualizando la vida del player */
48     void removeWorm(int id);
49
50     /* Elimina la vista del arma y la reemplaza por la animacion de la explosion
    */
51     void removeWeapon(int id);
52
53     /* Actualiza la posicion y la vida del worm */
54     void
55     updateWormData(int id, int player_id, float pos_x, float pos_y, int life,
56                    char dir, bool colliding);
57
58     /* Actualiza la posicion del arma */
59     void updateWeaponData(int id, const std::string& weapon_name, float pos_x,
60                           float pos_y);
61
62     /* CallBack de changeWeapon */
63     bool changeWeaponCallBack(const std::string& weapon_name);
64
65     /* Actualiza la vista del worm con el arma nueva */
```

jun 10, 18 15:09 **ViewsList.h** Page 2/2

```
66      void changeWeapon(const std::string& weapon_name);
67
68      /* Actualiza la posicion del scope */
69      void updateScope(int angle);
70
71      /* Esconde la vista del scope */
72      void removeScopeVisibility();
73
74      /* Agrega una viga a la vista en la posicion indicada y
75       * con la rotacion indicada */
76      void addGirder(size_t size, float pos_x, float pos_y, int rotation);
77
78      /* Actualiza el worm actual y hace focus en este */
79      void setCurrentWorm(int id);
80
81      /* Actualiza la imagen de los worms ganadores por la animacion
82       * de los worms festejando */
83      void setVictory();
84
85      /* Chequea si el gusano actual se esta moviendo, caso contario
86      le da el focus a otro */
87      void checkMovingWorms();
88
89      /* Realiza la animacion del disparo del arma */
90      void shoot(const std::string& weapon);
91  };
92
93
94  #endif
```

jun 24, 18 18:11 **Table of Content** Page 1/2

jun 24, 18 18:11 **Table of Content** Page 2/2