

jun 12, 18 14:03

## ClientProtocol.cpp

Page 1/3

```

1  #include "ClientProtocol.h"
2  #include <string>
3  #include "Player.h"
4  #include "WeaponList.h"
5  #include "ObjectSizes.h"
6  #include "ServerFatalError.h"
7
8  ClientProtocol::ClientProtocol(Socket&& socket, Gtk::Window& window) :
9      Protocol(std::move(socket)), window(window) {}
10
11 ClientProtocol::ClientProtocol(ClientProtocol&& other) :
12     Protocol(std::move(other)), window(other.window) {}
13
14 ClientProtocol::~ClientProtocol() {}
15
16 void ClientProtocol::sendMoveAction(char action) {
17     Buffer buffer;
18     buffer.setNext(ACTION);
19     buffer.setNext(MOVE_ACTION);
20     buffer.setNext(action);
21     this->sendBuffer(buffer);
22 }
23
24 void ClientProtocol::sendChangeWeapon(const std::string& weapon) {
25     Buffer buffer;
26     buffer.setNext(ACTION);
27     buffer.setNext(CHANGE_WEAPON_ACTION);
28     this->sendStringBuffer(buffer, weapon);
29     this->sendBuffer(buffer);
30 }
31
32 void
33 ClientProtocol::sendWeaponShoot(int32_t angle, int32_t power, int32_t time) {
34     Buffer buffer;
35     buffer.setNext(ACTION);
36     buffer.setNext(SHOOT_WEAPON);
37     this->sendIntBuffer(buffer, angle);
38     this->sendIntBuffer(buffer, power);
39     this->sendIntBuffer(buffer, time);
40     this->sendBuffer(buffer);
41 }
42
43 void ClientProtocol::sendWeaponSelfDirectedShoot(const Position& pos) {
44     Buffer buffer;
45     buffer.setNext(ACTION);
46     buffer.setNext(SHOOT_SELF_DIRECTED);
47
48     this->sendIntBuffer(buffer, pos.getX() * UNIT_TO_SEND);
49     this->sendIntBuffer(buffer, pos.getY() * UNIT_TO_SEND);
50
51     this->sendBuffer(buffer);
52 }
53
54 void ClientProtocol::updateScope(int angle) {
55     Buffer buffer;
56     buffer.setNext(ACTION);
57     buffer.setNext(MOVE_SCOPE);
58
59     this->sendIntBuffer(buffer, angle);
60
61     this->sendBuffer(buffer);
62 }
63
64 void ClientProtocol::sendEndGame() {
65     Buffer buffer;
66     buffer.setNext(END_GAME);

```

jun 12, 18 14:03

## ClientProtocol.cpp

Page 2/3

```

67     this->sendBuffer(buffer);
68 }
69
70 void ClientProtocol::receiveStartGame() {
71     Buffer buffer = std::move(this->receiveBuffer());
72 }
73
74 void ClientProtocol::receiveBackgroundImage(WorldView& world) {
75     Buffer buffer = std::move(this->receiveBuffer());
76     world.setBackgroundImage(buffer);
77 }
78
79 void ClientProtocol::receiveTurnData(Turn& turn) {
80     Buffer buffer = std::move(this->receiveBuffer());
81     int max_time = this->receiveIntBuffer(buffer);
82     int time_after_shoot = this->receiveIntBuffer(buffer);
83     turn.setTime(max_time, time_after_shoot);
84 }
85
86 void ClientProtocol::receivePlayers(PlayersList& players_list) {
87     Buffer buffer = std::move(this->receiveBuffer());
88     int quantity = this->receiveIntBuffer(buffer);
89
90     for (int i = 0; i < quantity; i++) {
91         Buffer buffer = std::move(this->receiveBuffer());
92
93         int id = this->receiveIntBuffer(buffer);
94         std::string name = this->receiveStringBuffer(buffer);
95
96         players_list.addPlayer(id, name);
97     }
98 }
99
100 void ClientProtocol::receiveGirders(ViewsList& viewsList) {
101     Buffer buffer = std::move(this->receiveBuffer());
102     int quantity = this->receiveIntBuffer(buffer);
103
104     for (int i = 0; i < quantity; i++) {
105         Buffer buffer = std::move(this->receiveBuffer());
106
107         int size = this->receiveIntBuffer(buffer);
108         float pos_x = this->receiveIntBuffer(buffer) / UNIT_TO_SEND;
109         float pos_y = this->receiveIntBuffer(buffer) / UNIT_TO_SEND;
110         int rotation = this->receiveIntBuffer(buffer);
111         viewsList.addGirder(size, pos_x, pos_y, rotation);
112     }
113 }
114
115 void ClientProtocol::receiveWeaponsAmmo(WeaponList& weapon_list) {
116     Buffer buffer = std::move(this->receiveBuffer());
117     int quantity = this->receiveIntBuffer(buffer);
118
119     for (int i = 0; i < quantity; i++) {
120         Buffer buffer = std::move(this->receiveBuffer());
121
122         std::string name = this->receiveStringBuffer(buffer);
123         int ammo = this->receiveIntBuffer(buffer);
124         weapon_list.add(name, ammo);
125     }
126 }
127
128 void ClientProtocol::sendBuffer(Buffer& buffer) {
129     try {
130         Protocol::sendBuffer(buffer);
131     } catch (const std::exception& e) {
132         ServerFatalError error(this->window);

```

jun 12, 18 14:03

## ClientProtocol.cpp

Page 3/3

```

133     }
134 }
```

jun 12, 18 14:03

## 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 mensaje 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 12, 18 14:03

## ClientProtocol.h

Page 2/2

```

67      /* Recibe la vigas presentes en el mapa junto con su tamaño,
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 14:03

## 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::analyzeReceivedData), 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::analyzeReceivedData(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.getMediaPlayer().playWeaponShotSound(weapon);
66     } else if (action == MOVING_OBJECT) {

```

jun 12, 18 14:03

**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.getMediaPlayer().playJumpSound(movement);
99     }
100     return false;
101 }

```

jun 12, 18 14:03

**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 analyzeReceivedData(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 12, 18 14:03

main.cpp

Page 1/1

```
1 #include <gtkmm/application.h>
2 #include <gtkmm/window.h>
3 #include "ServerMenu.h"
4 #include "Path.h"
5
6 int main(int argc, char* argv[]) {
7     auto app = Gtk::Application::create(argc, argv);
8     Gtk::Window window;
9     window.maximize();
10
11     window.set_title(CLIENT_WINDOW_NAME);
12
13     window.set_icon_from_file(ICON_PATH);
14
15     ServerMenu server_menu(window);
16
17     app->run(window);
18
19     return 0;
20 }
```

jun 12, 18 14:03

ButtonBuilder.cpp

Page 1/1

```
1 #include "ButtonBuilder.h"
2 #include <string>
3 #include <gtkmm/label.h>
4 #include <gdkmm/rgba.h>
5
6 void ButtonBuilder::buildButton(Gtk::Button* button) {
7     std::string text = button->get_label();
8     Gtk::Label* label = (Gtk::Label*) button->get_child();
9     label->set_markup("<b>" + text + "</b>");
10    label->override_color(Gdk::RGBA("black"));
11 }
```

jun 12, 18 14:03

**ButtonBuilder.h**

Page 1/1

```

1  #ifndef WORMS_BUTTONBUILDER_H
2  #define WORMS_BUTTONBUILDER_H
3
4  #include <gtkmm/button.h>
5
6  class ButtonBuilder {
7  public:
8      /* Modifica la visualizaci3n del label del boton */
9      static void buildButton(Gtk::Button* button);
10 };
11
12
13 #endif //WORMS_BUTTONBUILDER_H

```

jun 25, 18 19:28

**CreateGameMenu.cpp**

Page 1/1

```

1  #include "CreateGameMenu.h"
2  #include <string>
3  #include "Path.h"
4  #include "GamePlayers.h"
5
6  const std::string PATH = GLADE_PATH + "client_CreateGameMenu.glade";
7
8  CreateGameMenu::CreateGameMenu(Gtk::Window& window, MenuView& first_menu,
9      ClientProtocol& protocol, std::string&& name, int quantity) :
10      SelectableListMenu(window, first_menu, protocol, std::move(name), PATH) {
11      this->builder->get_widget("game_name", this->game_name);
12      this->builder->get_widget("players_number", this->players_number);
13      this->builder->get_widget("games", this->games);
14
15      this->configure(quantity);
16
17      this->builder->get_widget("create_game_menu", this->menu);
18
19      this->addMenu();
20  }
21
22  CreateGameMenu::~CreateGameMenu() {}
23
24  void CreateGameMenu::selectButtonPressed(Glib::ustring map_name) {
25      std::string name(this->game_name->get_text());
26      if (name.empty()) {
27          this->showError("Debe ingresar el nombre de la partida");
28          return;
29      }
30
31      size_t players = this->players_number->get_value_as_int();
32      if (players < min_players || players > max_players) {
33          std::string message("El numero de jugadores debe estar entre ");
34          message += std::to_string(min_players) + std::string(" y ");
35          message += std::to_string(max_players);
36          this->showError(message);
37          return;
38      }
39
40      try {
41          this->protocol.sendString(map_name);
42          this->protocol.sendString(name);
43          this->protocol.sendLength(players);
44          bool result = this->protocol.receiveChar();
45          if (!result) {
46              this->showErrorAndRestart("Ocurrio un error al crear la partida");
47          } else {
48              this->waitToPlayers();
49          }
50      } catch(const SocketException& e) {
51          this->showFatalError();
52      }
53  }

```

jun 12, 18 14:03

## CreateGameMenu.h

Page 1/1

```

1  #ifndef __CREATEGAMEMENU__
2  #define __CREATEGAMEMENU__
3
4  #include <gtkmm/entry.h>
5  #include <gtkmm/spinbutton.h>
6  #include <string>
7  #include "SelectableListMenu.h"
8
9  /* Clase que se encarga de los pasos necesarios para que el
10   * jugador cree una partida */
11  class CreateGameMenu : public SelectableListMenu {
12  private:
13      Gtk::Entry* game_name;
14      Gtk::SpinButton* players_number;
15
16      /* Handler del boton de seleccion */
17      void selectButtonPressed(Glib::ustring map_name) override;
18
19  public:
20      /* Constructor */
21      CreateGameMenu(Gtk::Window& window, MenuView& first_menu,
22                    ClientProtocol& protocol, std::string&& name, int quantity);
23
24      /* Destructor */
25      ~CreateGameMenu();
26  };
27
28 #endif

```

jun 25, 18 19:28

## GameMenu.cpp

Page 1/2

```

1  #include "GameMenu.h"
2  #include <string>
3  #include "Path.h"
4  #include "CreateGameMenu.h"
5  #include "JoinGameMenu.h"
6  #include "ButtonBuilder.h"
7
8  const std::string PATH = GLADE_PATH + "client_GameMenu.glade";
9
10 GameMenu::GameMenu(Gtk::Window& window, ClientProtocol& protocol) :
11     MenuView(window, *this, protocol, PATH) {
12     this->builder->get_widget("player_name", this->player_name);
13
14     this->builder->get_widget("game_menu", this->menu);
15
16     this->addMenu();
17
18     Gtk::Button* create_game, * join_game;
19
20     this->builder->get_widget("create_game", create_game);
21     this->builder->get_widget("join_game", join_game);
22
23     ButtonBuilder::buildButton(create_game);
24     ButtonBuilder::buildButton(join_game);
25
26     create_game->signal_clicked().connect(sigc::mem_fun(*this,
27                                                         &GameMenu::createButtonPressed));
28     join_game->signal_clicked().connect(sigc::mem_fun(*this,
29                                                         &GameMenu::joinButtonPressed));
30 }
31
32 GameMenu::~GameMenu() {}
33
34 void GameMenu::createButtonPressed() {
35     if (this->selectAction(CREATE_GAME_ACTION)) {
36         std::string name(this->player_name->get_text());
37         int quantity = this->protocol.receiveLength();
38         if (quantity == 0) {
39             this->showErrorAndRestart("No hay mapas para crear una partida");
40         } else {
41             this->hideWarningBox();
42             this->next_menu = std::unique_ptr<MenuView>(
43                 new CreateGameMenu(this->window, *this, this->protocol,
44                                     std::move(name), quantity));
45         }
46     }
47 }
48
49 void GameMenu::joinButtonPressed() {
50     if (this->selectAction(JOIN_GAME_ACTION)) {
51         std::string name(this->player_name->get_text());
52         int quantity = this->protocol.receiveLength();
53         if (quantity == 0) {
54             this->showErrorAndRestart("No hay partidas disponibles");
55         } else {
56             this->hideWarningBox();
57             this->next_menu = std::unique_ptr<MenuView>(
58                 new JoinGameMenu(this->window, *this, this->protocol,
59                                     std::move(name), quantity));
60         }
61     }
62 }
63
64 bool GameMenu::selectAction(char action) {
65     std::string name(this->player_name->get_text());
66     if (name.empty()) {

```

jun 25, 18 19:28

**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 12, 18 14:03

**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 12, 18 14:03

## GameMenuField.h

Page 1/1

```

1  #ifndef __GAMEMENUFIELD_H__
2  #define __GAMEMENUFIELD_H__
3
4  #include <gtkmm/hvbox.h>
5  #include <gtkmm/label.h>
6  #include <gtkmm/button.h>
7  #include <string>
8
9  class GameMenuField {
10 private:
11     Gtk::Label title;
12     Gtk::Button button;
13     Gtk::HBox container;
14
15 public:
16     /* Constructor */
17     explicit GameMenuField(const std::string& title);
18
19     /* Destructor */
20     ~GameMenuField();
21
22     /* Constructor por movimiento */
23     GameMenuField(GameMenuField&& other);
24
25
26     /* Devuelve el contenedor del menu */
27     Gtk::Container& getContainer();
28
29     /* Devuelve el boton del menu */
30     Gtk::Button& getButton();
31 };
32
33 #endif

```

jun 12, 18 14:03

## GameMenu.h

Page 1/1

```

1  #ifndef __GAMEMENU__
2  #define __GAMEMENU__
3
4  #include <gtkmm/entry.h>
5  #include <string>
6  #include <memory>
7  #include "ClientProtocol.h"
8  #include "MenuView.h"
9
10 /* Clase que se encarga de controlar el menu del juego */
11 class GameMenu : public MenuView {
12 private:
13     Gtk::Entry* player_name;
14
15     /* Crea el boton de creacion de partida */
16     void createButtonPressed();
17
18     /* Crea el boton de unirse a partida */
19     void joinButtonPressed();
20
21     /* Envia la accion implementada */
22     bool selectAction(char action);
23
24 public:
25     /* Constructor */
26     GameMenu(Gtk::Window& window, ClientProtocol& protocol);
27
28     /* Destructor */
29     ~GameMenu();
30 };
31
32 #endif

```

jun 25, 18 19:28

## JoinGameMenu.cpp

Page 1/1

```

1  #include "JoinGameMenu.h"
2  #include <string>
3  #include "Path.h"
4  #include "WaitingRoom.h"
5
6  const std::string PATH = GLADE_PATH + "client_JoinGameMenu.glade";
7
8  JoinGameMenu::JoinGameMenu(Gtk::Window& window, MenuView& first_menu,
9                             ClientProtocol& protocol, std::string&& name, int quantity) :
10     SelectableListMenu(window, first_menu, protocol, std::move(name), PATH) {
11     this->builder->get_widget("games", this->games);
12
13     this->configure(quantity);
14
15     this->builder->get_widget("join_game_menu", this->menu);
16
17     this->addMenu();
18 }
19
20 JoinGameMenu::~JoinGameMenu() {}
21
22
23 void JoinGameMenu::selectButtonPressed(Glib::ustring game_name) {
24     try {
25         this->protocol.sendString(game_name);
26         bool result = this->protocol.receiveChar();
27         if (!result) {
28             this->showErrorAndRestart(
29                 "Ocurrio un error al unirse a la partida");
30         } else {
31             this->waitToPlayers();
32         }
33     } catch(const SocketException& e) {
34         this->showFatalError();
35     }
36 }

```

jun 12, 18 14:03

## JoinGameMenu.h

Page 1/1

```

1  #ifndef __JOINGAMEMENU__
2  #define __JOINGAMEMENU__
3
4  #include <string>
5  #include "SelectableListMenu.h"
6
7  /* Clase que se encarga de los pasos necesarios para que el
8   * jugador se una a una partida */
9  class JoinGameMenu : public SelectableListMenu {
10 private:
11     /* Handler del boton de unirse a partida */
12     void selectButtonPressed(Glib::ustring game_name) override;
13
14 public:
15     /* Constructor */
16     JoinGameMenu(Gtk::Window& window, MenuView& first_menu,
17                 ClientProtocol& protocol, std::string&& name, int quantity);
18
19     /* Destructor */
20     ~JoinGameMenu();
21 };
22
23 #endif

```

jun 25, 18 19:28

## Menu.cpp

Page 1/1

```

1  #include "Menu.h"
2  #include <string>
3  #include "Path.h"
4  #include "ButtonBuilder.h"
5
6  Menu::Menu(const std::string& path, Gtk::Window& window) : window(window) {
7      this->builder = Gtk::Builder::create_from_file(path);
8
9      this->builder->get_widget("error", this->error);
10
11     this->builder->get_widget("quit_game", this->quit);
12
13     ButtonBuilder::buildButton(this->quit);
14
15     this->builder->get_widget("title", this->title);
16     this->title->set(TITLE_MENU_IMAGE);
17
18     this->builder->get_widget("background", this->background);
19     this->background->set(BACKGROUND_MENU_IMAGE);
20
21     this->builder->get_widget("warning1", this->warning1);
22     this->warning1->set(WARNING_IMAGE);
23
24     this->builder->get_widget("warning2", this->warning2);
25     this->warning2->set(WARNING_IMAGE);
26
27     this->quit->signal_clicked().connect(
28         sigc::mem_fun(*this, &Menu::quitButtonPressed));
29
30     this->builder->get_widget("warning_box", this->warning_box);
31 }
32
33 Menu::~Menu() {}
34
35 void Menu::quitButtonPressed() {
36     this->window.close();
37 }
38
39 void Menu::hideWarningBox() {
40     this->error->hide();
41     this->warning1->hide();
42     this->warning2->hide();
43 }
44
45 void Menu::showError(const std::string& error) {
46     if (error.empty()) {
47         this->hideWarningBox();
48     } else {
49         this->error->set_label(error);
50         this->warning_box->show_all();
51     }
52 }

```

jun 25, 18 19:28

## Menu.h

Page 1/1

```

1  #ifndef WORMS_MENU_H
2  #define WORMS_MENU_H
3
4  #include <gtkmm/button.h>
5  #include <gtkmm/label.h>
6  #include <gtkmm/window.h>
7  #include <gtkmm/image.h>
8  #include <gtkmm/builder.h>
9  #include <gtkmm/box.h>
10 #include <string>
11
12 class Menu {
13 protected:
14     Gtk::Label* error;
15     Gtk::Button* quit;
16     Gtk::Window& window;
17     Gtk::Image* title;
18     Gtk::Image* background;
19     Gtk::Image* warning1;
20     Gtk::Image* warning2;
21     Gtk::Box* warning_box;
22     Glib::RefPtr<Gtk::Builder> builder;
23
24     /* Handler del boton de salir */
25     void quitButtonPressed();
26
27     /* Muestra un mensaje de error */
28     void showError(const std::string& error);
29
30     /* Oculta el mensaje de error */
31     void hideWarningBox();
32
33 public:
34     /* Constructor */
35     Menu(const std::string& path, Gtk::Window& window);
36
37     /* Destructor */
38     ~Menu();
39 };
40
41
42 #endif //WORMS_MENU_H

```

jun 25, 18 19:28

## MenuView.cpp

Page 1/1

```

1  #include "MenuView.h"
2  #include <string>
3  #include "ServerFatalError.h"
4
5  MenuView::MenuView(Gtk::Window& window, MenuView& main_menu,
6                    ClientProtocol& protocol, const std::string& path) :
7      Menu(path, window), protocol(protocol), main_menu(main_menu) {}
8
9  MenuView::~MenuView() {
10     delete this->menu;
11 }
12
13 void MenuView::showFatalError() {
14     ServerFatalError error(this->window);
15 }
16
17 void MenuView::showErrorAndRestart(const std::string& error) {
18     this->window.remove();
19     this->main_menu.showError(error);
20     this->window.add(*this->main_menu.menu);
21 }
22
23 void MenuView::addMenu() {
24     this->window.add(*this->menu);
25     this->window.show_all();
26     this->hideWarningBox();
27 }

```

jun 25, 18 19:28

## MenuView.h

Page 1/1

```

1  #ifndef __MENUVIEW_H__
2  #define __MENUVIEW_H__
3
4  #include <gtkmm/container.h>
5  #include <memory>
6  #include <string>
7  #include "ClientProtocol.h"
8  #include "Menu.h"
9
10 class MenuView : public Menu {
11 protected:
12     std::unique_ptr<MenuView> next_menu;
13     ClientProtocol& protocol;
14     MenuView& main_menu;
15     Gtk::Container* menu;
16
17     /* Muestra un mensaje de error y cierra la aplicacion */
18     void showFatalError();
19
20     /* Muestra un mensaje de error y reinicia */
21     void showErrorAndRestart(const std::string& error);
22
23 public:
24     /* Constructor */
25     MenuView(Gtk::Window& window, MenuView& main_menu, ClientProtocol& protocol,
26             const std::string& path);
27
28     /* Destructor */
29     virtual ~MenuView();
30
31     /* Agrega el menu al container y el container al window */
32     void addMenu();
33 };
34
35 #endif

```

jun 25, 18 19:28

## SelectableListMenu.cpp

Page 1/2

```

1  #include "SelectableListMenu.h"
2  #include <string>
3  #include "ButtonBuilder.h"
4
5  SelectableListMenu::SelectableListMenu(Gtk::Window& window,
6                                         MenuView& first_menu,
7                                         ClientProtocol& protocol,
8                                         std::string&& name,
9                                         const std::string& path) :
10     MenuView(window, first_menu, protocol, path),
11     player_name(std::move(name)) {
12     this->builder->get_widget("turn_back", this->turn_back);
13     ButtonBuilder::buildButton(this->turn_back);
14     this->turn_back->signal_clicked().connect(
15         sigc::mem_fun(*this, &SelectableListMenu::turnBackButtonPressed));
16 }
17
18 SelectableListMenu::~SelectableListMenu() {}
19
20 void SelectableListMenu::turnBackButtonPressed() {
21     std::string string;
22     try {
23         this->protocol.sendString(string);
24         this->showErrorAndRestart(string);
25     } catch(const std::exception& e) {
26         this->showFatalError();
27     }
28 }
29
30 void SelectableListMenu::configure(int quantity) {
31     try {
32         for (int i = 0; i < quantity; i++) {
33             std::string field = this->protocol.receiveString();
34             this->addField(field);
35         }
36     } catch(const SocketException& e) {
37         this->showFatalError();
38     }
39
40     for (auto it = this->fields.begin(); it != this->fields.end(); ++it) {
41         this->games->pack_start(it->getContainer());
42     }
43     this->games->show();
44 }
45
46 void SelectableListMenu::addField(const std::string& field_name) {
47     GameMenuField field(field_name);
48     this->fields.push_back(std::move(field));
49     this->fields.back().getButton().signal_clicked().connect(
50         sigc::bind<Glib::ustring>(sigc::mem_fun(*this,
51         &SelectableListMenu::selectButtonPressed), field_name));
52 }
53
54 bool SelectableListMenu::createPlayer() {
55     try {
56         this->player = std::unique_ptr<Player>(
57             new Player(this->protocol, this->player_name, this->window,
58                 this->main_menu));
59     } catch(const std::exception& e) {
60         this->showFatalError();
61     }
62     return false;
63 }
64
65 void SelectableListMenu::waitToPlayers() {
66     this->window.remove();

```

jun 25, 18 19:28

## SelectableListMenu.cpp

Page 2/2

```

67     this->window.add(this->waiting_room.getWidget());
68     this->window.show_all();
69     sigc::slot<bool> my_slot = sigc::mem_fun(*this,
70         &SelectableListMenu::createPlayer);
71     Glib::signal_idle().connect(my_slot);
72 }

```

jun 25, 18 19:28

**SelectableListMenu.h**

Page 1/1

```

1  #ifndef __SELECTABLELISTMENU_H__
2  #define __SELECTABLELISTMENU_H__
3
4  #include <gtkmm/box.h>
5  #include <gtkmm/button.h>
6  #include <memory>
7  #include <string>
8  #include <vector>
9  #include "ClientProtocol.h"
10 #include "MenuView.h"
11 #include "WaitingRoom.h"
12 #include "Player.h"
13 #include "GameMenuField.h"
14
15 class SelectableListMenu : public MenuView {
16 protected:
17     Gtk::Box* games;
18     std::string player_name;
19     WaitingRoom waiting_room;
20     std::vector<GameMenuField> fields;
21     std::unique_ptr<Player> player;
22     Gtk::Button* turn_back;
23
24     /* Realiza la configuracion del juego */
25     void configure(int quantity);
26
27     /* Agrega un campo a la lista */
28     void addField(const std::string& field_name);
29
30     /* Crea un nuevo jugador */
31     bool createPlayer();
32
33     /* Handler del boton de seleccion */
34     virtual void selectButtonPressed(Glib::ustring field_name) = 0;
35
36     /* Handler del boton volver */
37     void turnBackButtonPressed();
38
39     /* Muestra el mensaje esperando jugadores */
40     void waitToPlayers();
41
42 public:
43     /* Constructor */
44     SelectableListMenu(Gtk::Window& window, MenuView& first_menu,
45                       ClientProtocol& protocol, std::string&& name,
46                       const std::string& path);
47
48     /* Destructor */
49     ~SelectableListMenu();
50 };
51
52 #endif

```

jun 12, 18 14:03

**ServerFatalError.cpp**

Page 1/1

```

1  #include "ServerFatalError.h"
2  #include <gtkmm/messagedialog.h>
3  #include <string>
4
5  const std::string ERROR_MSG("Ocurrio un error con la conexion del servidor");
6
7  ServerFatalError::ServerFatalError(Gtk::Window& window) {
8      Gtk::MessageDialog dialog(window, ERROR_MSG, false, Gtk::MESSAGE_ERROR,
9                                Gtk::BUTTONS_CLOSE, true);
10
11      dialog.run();
12      window.close();
13  }
14
15  ServerFatalError::~ServerFatalError() {}

```

jun 12, 18 14:03

## ServerFatalError.h

Page 1/1

```

1  #ifndef __SERVERFATALERROR_H__
2  #define __SERVERFATALERROR_H__
3
4  #include <gtkmm/window.h>
5
6  /* Clase que se encarga de mostrar un error fatal
7   * con la conexi3n entre el servidor y el cliente */
8  class ServerFatalError {
9  public:
10     /* Constructor */
11     explicit ServerFatalError(Gtk::Window& window);
12
13     /* Destructor */
14     ~ServerFatalError();
15 };
16
17 #endif

```

jun 25, 18 19:28

## ServerMenu.cpp

Page 1/1

```

1  #include "ServerMenu.h"
2  #include <string>
3  #include "Path.h"
4  #include "ButtonBuilder.h"
5
6  const std::string PATH = GLADE_PATH + "client_ServerMenu.glade";
7
8  ServerMenu::ServerMenu(Gtk::Window& window) : Menu(PATH, window) {
9      this->builder->get_widget("host", this->host);
10     this->builder->get_widget("service", this->service);
11     this->builder->get_widget("connect", this->connect);
12
13     ButtonBuilder::buildButton(this->connect);
14
15     this->builder->get_widget("server_menu", this->menu);
16
17     this->window.add(*this->menu);
18     this->window.show_all();
19
20     this->hideWarningBox();
21
22     this->connect->signal_clicked().connect(
23         sigc::mem_fun(*this, &ServerMenu::connectButtonPressed));
24 }
25
26 ServerMenu::~ServerMenu() {
27     delete this->menu;
28 }
29
30 void ServerMenu::connectButtonPressed() {
31     std::string host(this->host->get_text());
32     if (host.empty()) {
33         this->showError("Debe ingresar un host");
34         return;
35     }
36
37     std::string service(this->service->get_text());
38     if (service.empty()) {
39         this->showError("Debe ingresar un servicio");
40         return;
41     }
42
43     this->connectToServer(host, service);
44 }
45
46 void ServerMenu::connectToServer(const std::string& host,
47                                 const std::string& service) {
48     try {
49         Socket socket(Socket::Client(host.c_str(), service.c_str()));
50         this->protocol.reset(
51             new ClientProtocol(std::move(socket), this->window));
52         this->window.remove();
53         this->next_menu = std::unique_ptr<MenuView>(
54             new GameMenu(this->window, *this->protocol));
55     } catch (const SocketException& e) {
56         this->showError("No pudo conectarse al servidor");
57     }
58 }

```

jun 12, 18 14:03

## ServerMenu.h

Page 1/1

```

1  #ifndef __SERVERMENU__
2  #define __SERVERMENU__
3
4  #include <gtkmm/button.h>
5  #include <gtkmm/entry.h>
6  #include <string>
7  #include <memory>
8  #include "ClientProtocol.h"
9  #include "GameMenu.h"
10 #include "MenuView.h"
11 #include "Menu.h"
12
13 /* Menu de conexion con el servidor */
14 class ServerMenu : public Menu {
15 private:
16     Gtk::Entry* host;
17     Gtk::Entry* service;
18     Gtk::Button* connect;
19     Gtk::Container* menu;
20     std::unique_ptr<MenuView> next_menu;
21     std::unique_ptr<ClientProtocol> protocol;
22
23     /* Handler del boton de conexion */
24     void connectButtonPressed();
25
26     /* Intenta realizar una conexion con el servidor */
27     void connectToServer(const std::string& host, const std::string& service);
28
29 public:
30     /* Constructor */
31     explicit ServerMenu(Gtk::Window& window);
32
33     /* Destructor */
34     ~ServerMenu();
35 };
36
37 #endif

```

jun 25, 18 19:28

## WaitingRoom.cpp

Page 1/1

```

1  #include "WaitingRoom.h"
2  #include "Path.h"
3  #include <string>
4
5  const std::string begining("<span size='20000'>");
6  const std::string ending("</span>");
7
8  #define SPACING 30
9
10 WaitingRoom::WaitingRoom(): container(true, SPACING){
11     this->label.set_use_markup(true);
12     this->label.set_markup(begining + "Esperando jugadores..." + ending);
13     this->image.set(WAITING_ROOM_IMAGE);
14     this->container.pack_start(this->image, Gtk::PACK_SHRINK);
15     this->container.pack_start(this->label, Gtk::PACK_SHRINK);
16     this->container.set_halign(Gtk::ALIGN_CENTER);
17     this->container.set_valign(Gtk::ALIGN_CENTER);
18 }
19
20 WaitingRoom::~WaitingRoom() {}
21
22 Gtk::Widget& WaitingRoom::getWidget() {
23     return this->container;
24 }

```



jun 25, 18 19:28

## WaitingRoom.h

Page 1/1

```

1  #ifndef __WAITINGROOM_H__
2  #define __WAITINGROOM_H__
3
4  #include <gtkmm/label.h>
5  #include <gtkmm/image.h>
6  #include <gtkmm/hvbox.h>
7
8  /* Label de que indica la espera a otros jugadores */
9  class WaitingRoom {
10 private:
11     Gtk::VBox container;
12     Gtk::Label label;
13     Gtk::Image image;
14
15 public:
16     /* Constructor */
17     WaitingRoom();
18
19     /* Destructor */
20     ~WaitingRoom();
21
22     /* Devuelve el contenedor del mensaje */
23     Gtk::Widget& getWidget();
24 };
25
26 #endif

```

jun 12, 18 14:03

## Handlers.cpp

Page 1/3

```

1  #include "Handlers.h"
2  #include <gtkmm/adjustment.h>
3  #include <gdk/gdkkeysyms.h>
4  #include "Player.h"
5  #include "ViewPositionTransformer.h"
6  #include "WeaponNames.h"
7
8  const char SPACE = ' ';
9  const int WEAPONS_DEFAULT_TIME = 3;
10 const char ASCII_OFFSET = 48;
11 const char ASCII_1 = 49;
12 const char ASCII_5 = 53;
13 const int MAX_TIME = 3000;
14 const int ANGLE_STEP = 6;
15
16 Handlers::Handlers(Player& player, ViewsList& view_list, WeaponList& weapons,
17     WorldView& world) :
18     player(player), view_list(view_list), weapons(weapons), world(world),
19     scroll_handler(world.getWindow()), power_accumulator(*this, MAX_TIME) {
20     this->has_shoot = false;
21     this->current_angle = DEFAULT_ANGLE;
22     this->weapons_time = WEAPONS_DEFAULT_TIME;
23     this->enabled = false;
24 }
25
26 Handlers::~Handlers() {}
27
28 void Handlers::enableAll() {
29     this->weapons_time = WEAPONS_DEFAULT_TIME;
30     this->current_angle = DEFAULT_ANGLE;
31     this->has_shoot = false;
32     this->enabled = true;
33
34     this->player.getProtocol().updateScope(DEFAULT_ANGLE);
35
36     Gtk::Container* window = this->world.getWindow().get_parent()->get_parent();
37
38     window->set_can_focus(true);
39     window->grab_focus();
40
41     window->signal_key_press_event().connect(
42         sigc::mem_fun(*this, &Handlers::keyPressHandler));
43     window->signal_key_release_event().connect(
44         sigc::mem_fun(*this, &Handlers::keyReleaseHandler));
45     this->world.getWindow().signal_button_press_event().connect(
46         sigc::mem_fun(*this, &Handlers::onButtonPressEvent));
47 }
48
49 void Handlers::disableAll() {
50     this->enabled = false;
51 }
52
53 bool Handlers::isEnabled() const {
54     return this->enabled;
55 }
56
57 void Handlers::powerAccumStopped(int power) {
58     this->player.shoot(this->current_angle, power, this->weapons_time);
59 }
60
61 bool Handlers::keyPressHandler(GdkEventKey* key_event) {
62     if (!this->enabled) {
63         return true;
64     }
65
66     if (key_event->keyval == GDK_KEY_Left) {

```

jun 12, 18 14:03

## Handlers.cpp

Page 2/3

```

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;

```

jun 12, 18 14:03

## Handlers.cpp

Page 3/3

```

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 }

```

jun 12, 18 14:03	Handlers.h	Page 1/2
1	<b>#ifndef</b> __HANDLERS_H__	
2	<b>#define</b> __HANDLERS_H__	
3		
4	<b>#include</b> <gdk/gdk.h>	
5	<b>#include</b> "WeaponPowerAccum.h"	
6	<b>#include</b> "ScrollHandler.h"	
7		
8	class Player;	
9		
10	class ViewsList;	
11		
12	class WeaponList;	
13		
14	class WorldView;	
15		
16	<i>/* Clase que se encarga de definir los handlers del teclado y</i>	
17	<i>del mouse. */</i>	
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	<i>/* Constructor */</i>	
36	Handlers(Player& player, ViewsList& view_list, WeaponList& weapons,	
37	WorldView& world);	
38		
39	<i>/* Destructor */</i>	
40	~Handlers();	
41		
42	<i>/* Handler completo para el presionado de teclas. Indica</i>	
43	<i>los pasos que se deben realizar al presionar una tecla</i>	
44	<i>especifica */</i>	
45	bool keyPressHandler(GdkEventKey* key_event);	
46		
47	<i>/* Handler completo para la liberaci3n de teclas. Indica</i>	
48	<i>los pasos que se deben realizar al liberar una tecla</i>	
49	<i>especifica */</i>	
50	bool keyReleaseHandler(GdkEventKey* key_event);	
51		
52	<i>/* Handler del mouse. Indica los pasos que se deben realizar</i>	
53	<i>al utilizar el mouse */</i>	
54	bool onButtonPressEvent(GdkEventButton* event);	
55		
56	<i>/* Habilita todos los handlers */</i>	
57	void enableAll();	
58		
59	<i>/* Deshabilita todos los handlers */</i>	
60	void disableAll();	
61		
62	<i>/* Devuelve true si los handlers estan habilitados */</i>	
63	bool isEnabled() <b>const</b> ;	
64		
65	<i>/* Realiza el shoot del player */</i>	
66	void powerAccumStopped(int power);	

jun 12, 18 14:03	Handlers.h	Page 2/2
67		
68	<i>/* Devuelve el angulo actual del scope */</i>	
69	int getCurrentAngle() <b>const</b> ;	
70		
71	<i>/* Detiene los handlers */</i>	
72	void stop();	
73	};	
74		
75	<b>#endif</b>	

jun 12, 18 14:03

## Player.cpp

Page 1/2

```

1  #include "Player.h"
2  #include <string>
3  #include "WeaponNames.h"
4
5  Player::Player(ClientProtocol& protocol, const std::string& name,
6               Gtk::Window& window, MenuView& main_menu) :
7      protocol(protocol), name(name),
8      screen(window, main_menu, *this, this->weapons),
9      turn(*this, this->screen.getTurnLabel()),
10     view_list(this->screen.getWorld(), *this, this->screen.getPlayersView(),
11             musicPlayer),
12     data_receiver(*this),
13     handlers(*this, this->view_list, this->weapons,
14             this->screen.getWorld()) {
15     this->musicPlayer.playMusic();
16     this->data_receiver.start();
17 }
18
19 Player::~Player() {
20     this->data_receiver.stop();
21     this->data_receiver.join();
22 }
23
24 void Player::startTurn(int worm_id, int player_id, float wind) {
25     this->view_list.setCurrentWorm(worm_id);
26     this->screen.getWindView().update(wind);
27     const std::string& current_player = this->screen.getPlayersView().getPlayer(
28         player_id);
29     if (current_player == this->name) {
30         //Es mi turno
31         this->musicPlayer.playStartTurnSound();
32         this->handlers.enableAll();
33         this->changeWeapon(this->weapons.getCurrentWeapon().getName());
34         this->screen.getTurnLabel().beginTurn();
35         this->turn.start();
36     } else {
37         this->screen.getTurnLabel().beginTurn(current_player);
38     }
39 }
40
41 void Player::endTurn() {
42     this->turn.stop();
43     this->screen.getTurnLabel().endTurn();
44     this->view_list.removeScopeVisibility();
45 }
46
47 void Player::endGame(const std::string& winner) {
48     this->data_receiver.stop();
49     this->screen.getTurnLabel().setEndGame();
50     this->view_list.setVictory();
51     this->protocol.sendEndGame();
52     this->handlers.stop();
53     this->screen.setWinner(winner, this->name == winner);
54 }
55
56 void Player::shootWeapon() {
57     this->turn.reduceTime();
58     this->weapons.getCurrentWeapon().shoot();
59 }
60
61 void Player::changeWeapon(std::string weapon) {
62     this->musicPlayer.playSelectWeaponSound();
63     this->weapons.changeWeapon(weapon);
64     if (this->handlers.isEnabled()) {
65         this->protocol.sendChangeWeapon(weapon);
66     }

```

jun 12, 18 14:03

## Player.cpp

Page 2/2

```

67 }
68
69 void Player::shoot(Position position) {
70     this->shootWeapon();
71     this->protocol.sendWeaponSelfDirectedShoot(position);
72     this->screen.getWeaponsView().updateAmmo(this->weapons.getCurrentWeapon());
73 }
74
75 void Player::playTickTime() {
76     this->musicPlayer.playTickSound();
77 }
78
79 void Player::shoot(int angle, int power, int time) {
80     this->shootWeapon();
81     if (!this->weapons.getCurrentWeapon().isTimed()) {
82         time = -1;
83     }
84     if (!this->weapons.getCurrentWeapon().hasScope()) {
85         angle = MAX_WEAPON_ANGLE * 8;
86     }
87     this->protocol.sendWeaponShoot(angle, power, time);
88     this->view_list.removeScopeVisibility();
89     this->screen.getWeaponsView().updateAmmo(this->weapons.getCurrentWeapon());
90 }
91
92 ViewsList& Player::getViewsList() {
93     return this->view_list;
94 }
95
96 ScreenView& Player::getScreen() {
97     return this->screen;
98 }
99
100 WeaponList& Player::getWeapons() {
101     return this->weapons;
102 }
103
104 ClientProtocol& Player::getProtocol() {
105     return this->protocol;
106 }
107
108 MusicPlayer& Player::getMusicPlayer() {
109     return this->musicPlayer;
110 }
111
112 Turn& Player::getTurn() {
113     return this->turn;
114 }

```

jun 12, 18 14:03

## Player.h

Page 1/2

```

1  #ifndef __CLIENTPLAYER_H__
2  #define __CLIENTPLAYER_H__
3
4  #include <memory>
5  #include <gtkmm/window.h>
6  #include <string>
7  #include "MenuView.h"
8  #include "ClientProtocol.h"
9  #include "Turn.h"
10 #include "Weapon.h"
11 #include "WeaponList.h"
12 #include "ScreenView.h"
13 #include "ViewsList.h"
14 #include "Position.h"
15 #include "DataReceiver.h"
16 #include "Handlers.h"
17 #include "MusicPlayer.h"
18
19 class Player {
20 private:
21     ClientProtocol& protocol;
22     std::string name;
23     WeaponList weapons;
24     ScreenView screen;
25     Turn turn;
26     ViewsList view_list;
27     DataReceiver data_receiver;
28     Handlers handlers;
29     MusicPlayer musicPlayer;
30
31     /* Reduce el tiempo del turno y actualiza la municion */
32     void shootWeapon();
33
34 public:
35     /* Constructor */
36     Player(ClientProtocol& protocol, const std::string& name,
37           Gtk::Window& window, MenuView& main_menu);
38
39     /* Destructor */
40     ~Player();
41
42
43     /* Comienza el turno. Si es el turno del jugador entonces,
44        habilita los handlers, sino muestra los movimientos realizados
45        por el otro jugador */
46     void startTurn(int worm_id, int player_id, float wind);
47
48     /* Finaliza el turno del jugador actual */
49     void endTurn();
50
51     /* Finaliza el juego */
52     void endGame(const std::string& winner);
53
54     /* Cambia el arma actual por la espeificada */
55     void changeWeapon(std::string weapon);
56
57     /* Realiza el disparo del arma con el angulo, potencia
58        y tiempo pasados */
59     void shoot(int angle, int power, int time);
60
61     /* Realiza el disparo del arma en la posicion pasada */
62     void shoot(Position position);
63
64     /* Reproduce el sonido de falta de tiempo */
65     void playTickTime();
66

```

jun 12, 18 14:03

## Player.h

Page 2/2

```

67     /* Devuelve la lista de los elementos presentes en la vista */
68     ViewsList& getViewsList();
69
70     /* Devuelve la vista */
71     ScreenView& getScreen();
72
73     /* Devuelve la lista de armas */
74     WeaponList& getWeapons();
75
76     /* Devuelve el protocolo */
77     ClientProtocol& getProtocol();
78
79     /* Devuelve el music player */
80     MusicPlayer& getMusicPlayer();
81
82     /* Devuelve el turno */
83     Turn& getTurn();
84 };
85
86 #endif

```

jun 12, 18 14:03

## Turn.cpp

Page 1/1

```

1  #include "Turn.h"
2  #include <glibmm/main.h>
3  #include "Player.h"
4
5  const int TIME_DEFAULT = 60;
6  const int REDUCTION_TIME_DEFAULT = 3;
7  const int LIMIT_TIME = 10;
8
9  Turn::Turn(Player& player, TurnLabel& time_label) :
10     actual_time(TIME_DEFAULT), player(player), time_label(time_label),
11     max_time(TIME_DEFAULT), reduction_time(REDUCTION_TIME_DEFAULT) {}
12
13  Turn::~Turn() {}
14
15  bool Turn::startCallBack() {
16     if (this->actual_time <= LIMIT_TIME) {
17         this->player.playTickTime();
18     }
19
20     this->actual_time--;
21     if (this->actual_time < 0) {
22         return false;
23     }
24     this->time_label.setTime(this->actual_time);
25     return true;
26 }
27
28 void Turn::start() {
29     this->actual_time = this->max_time;
30     this->my_connection = Glib::signal_timeout().connect(
31         sigc::mem_fun(*this, &Turn::startCallBack), 1000);
32 }
33
34 void Turn::reduceTime() {
35     this->actual_time = this->reduction_time;
36 }
37
38 void Turn::stop() {
39     if (this->my_connection.connected()) {
40         this->my_connection.disconnect();
41     }
42 }
43
44 void Turn::setTime(int time, int reduction_time) {
45     this->max_time = time;
46     this->reduction_time = reduction_time;
47 }

```

jun 12, 18 14:03

## Turn.h

Page 1/1

```

1  #ifndef __CLIENTTURN_H__
2  #define __CLIENTTURN_H__
3
4  #include "TurnLabel.h"
5
6  class Player;
7
8  /* Clase que se encarga de contar el tiempo del turno */
9  class Turn {
10 private:
11     int actual_time;
12     Player& player;
13     TurnLabel& time_label;
14     sigc::connection my_connection;
15     int max_time;
16     int reduction_time;
17
18     /* Callback de start */
19     bool startCallBack();
20
21 public:
22     /* Constructor */
23     Turn(Player& player, TurnLabel& time_label);
24
25     /* Destructor */
26     ~Turn();
27
28
29     /* Comienza la cuenta regresiva del turno actualizando el
30      * label que muestra el tiempo */
31     void start();
32
33     /* Reduce el tiempo restante del turno a 3 segundos */
34     void reduceTime();
35
36     /* Detiene el contador y finaliza el turno */
37     void stop();
38
39     /* Setea los tiempos */
40     void setTime(int time, int reduction_time);
41 };
42
43 #endif

```

jun 12, 18 14:03

**DistanceWeapon.cpp**

Page 1/1

```

1  #include "DistanceWeapon.h"
2  #include <string>
3
4  DistanceWeapon::DistanceWeapon(std::string name, int ammo, bool time) :
5      Weapon(name, ammo) {
6      this->has_Scope = true;
7      this->is_Timed = time;
8  }
9
10 DistanceWeapon::~DistanceWeapon() {}
11
12 DistanceWeapon::DistanceWeapon(DistanceWeapon&& other) : Weapon(
13     std::move(other)) {}
14
15 bool DistanceWeapon::hasVariablePower() const {
16     return true;
17 }
18

```

jun 12, 18 14:03

**DistanceWeapon.h**

Page 1/1

```

1  #ifndef __CLIENTDISTANCEWEAPON_H__
2  #define __CLIENTDISTANCEWEAPON_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  /* Clase que se encarga de representar a las armas de distancia */
8  class DistanceWeapon : public Weapon {
9  public:
10     /* Constructor */
11     DistanceWeapon(std::string name, int ammo, bool time = false);
12
13     /* Destructor */
14     ~DistanceWeapon();
15
16     /* Constructor por movimiento */
17     DistanceWeapon(DistanceWeapon&& other);
18
19
20     /* Devuelve true si el arma tiene potencia variable */
21     bool hasVariablePower() const override;
22 };
23
24 #endif

```

jun 12, 18 14:03

**MeleeWeapon.cpp**

Page 1/1

```

1  #include "MeleeWeapon.h"
2  #include <limits>
3  #include <string>
4
5  MeleeWeapon::MeleeWeapon(std::string name, int ammo, bool scope, bool time) :
6      Weapon(name, ammo) {
7      this->has_Scope = scope;
8      this->is_Timed = time;
9  }
10
11  MeleeWeapon::MeleeWeapon(MeleeWeapon&& other) : Weapon(std::move(other)) {}
12

```

jun 12, 18 14:03

**MeleeWeapon.h**

Page 1/1

```

1  #ifndef __CLIENTMELEEWEAPON_H__
2  #define __CLIENTMELEEWEAPON_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  /* Clase que se encarga de representar las armas de cuerpo a cuerpo */
8  class MeleeWeapon : public Weapon {
9  public:
10     /* Constructor */
11     MeleeWeapon(std::string name, int ammo, bool scope, bool time = false);
12
13     /* Destructor */
14     ~MeleeWeapon() {}
15
16     /* Constructor por movimiento */
17     MeleeWeapon(MeleeWeapon&& other);
18 };
19
20 #endif

```



jun 12, 18 14:03

## WeaponPowerAccum.cpp

Page 1/1

```

1  #include "WeaponPowerAccum.h"
2  #include "Handlers.h"
3
4  const int TIME_STEP = 50;
5  const int MINIMUM_POWER = 1000;
6  const int POWER_STEP = 15;
7
8  WeaponPowerAccum::WeaponPowerAccum(Handlers& handlers, int time) :
9      actual_time(0), max_time(time), handlers(handlers) {}
10
11  WeaponPowerAccum::~WeaponPowerAccum() {}
12
13  bool WeaponPowerAccum::startCallBack() {
14      this->actual_time += TIME_STEP;
15      this->power += POWER_STEP;
16
17      if (this->actual_time == this->max_time) {
18          this->handlers.powerAccumStopped(this->power);
19          return false;
20      }
21      return true;
22  }
23
24  void WeaponPowerAccum::start() {
25      this->actual_time = 0;
26      this->power = MINIMUM_POWER;
27      this->my_connection = Glib::signal_timeout().connect(
28          sigc::mem_fun(*this, &WeaponPowerAccum::startCallBack), TIME_STEP);
29  }
30
31  void WeaponPowerAccum::stop() {
32      if (this->my_connection.connected()) {
33          this->my_connection.disconnect();
34          this->handlers.powerAccumStopped(this->power);
35      }
36  }

```

jun 12, 18 14:03

## WeaponPowerAccum.h

Page 1/1

```

1  #ifndef __CLIENTTIMER_H__
2  #define __CLIENTTIMER_H__
3
4  #include <glibmm/main.h>
5
6  class Handlers;
7
8  /* Clase que simula a un contador */
9  class WeaponPowerAccum {
10 private:
11     int actual_time;
12     int max_time;
13     int power;
14     Handlers& handlers;
15     sigc::connection my_connection;
16
17     /* Callback de start */
18     bool startCallBack();
19
20 public:
21     /* Constructor */
22     WeaponPowerAccum(Handlers& handlers, int time);
23
24     /* Destructor */
25     ~WeaponPowerAccum();
26
27     /* Cuenta el tiempo transcurrido y llama al metodo timerStopped
28        de la clase Handler con este tiempo */
29     void start();
30
31     /* Detiene el contador */
32     void stop();
33 };
34
35 #endif

```

jun 12, 18 14:03

**AirAttack.cpp**

Page 1/1

```
1 #include "AirAttack.h"
2 #include "WeaponNames.h"
3
4 AirAttack::AirAttack(int ammo) : SelfDirectedWeapon(AIR_ATTACK_NAME, ammo) {}
5
6 AirAttack::~AirAttack() {}
7
8 AirAttack::AirAttack(AirAttack&& other) : SelfDirectedWeapon(
9     std::move(other)) {}
10
```

jun 12, 18 14:03

**AirAttack.h**

Page 1/1

```
1 #ifndef __CLIENTAIRATTACK_H__
2 #define __CLIENTAIRATTACK_H__
3
4 #include "SelfDirectedWeapon.h"
5
6 /* Clase que representa al arma AirStrike */
7 class AirAttack : public SelfDirectedWeapon {
8 public:
9     /* Constructor */
10    explicit AirAttack(int ammo);
11
12    /* Destructor */
13    ~AirAttack();
14
15    /* Constructor por movimiento */
16    AirAttack(AirAttack&& other);
17 };
18
19 #endif
```

jun 09, 18 19:03

**Banana.cpp**

Page 1/1

```

1  #include "Banana.h"
2  #include "WeaponNames.h"
3
4  Banana::Banana(int ammo) : DistanceWeapon(BANANA_NAME, ammo, true) {}
5
6  Banana::~Banana() {}
7
8  Banana::Banana(Banana&& other) : DistanceWeapon(std::move(other)) {}
9

```

jun 12, 18 14:03

**Banana.h**

Page 1/1

```

1  #ifndef __CLIENTBANANA_H__
2  #define __CLIENTBANANA_H__
3
4  #include "DistanceWeapon.h"
5
6  /* Clase que representa al arma Banana */
7  class Banana : public DistanceWeapon {
8  public:
9      /* Constructor */
10     explicit Banana(int ammo);
11
12     /* Destructor */
13     ~Banana();
14
15     /* Constructor por movimiento */
16     Banana(Banana&& other);
17 };
18
19 #endif

```

jun 12, 18 14:03

**Bat.cpp**

Page 1/1

```
1 #include "Bat.h"
2 #include "WeaponNames.h"
3
4 Bat::Bat(int ammo) : MeleeWeapon(BAT_NAME, ammo, true) {}
5
6 Bat::~Bat() {}
7
8 Bat::Bat(Bat&& other) : MeleeWeapon(std::move(other)) {}
9
```

jun 12, 18 14:03

**Bat.h**

Page 1/1

```
1 #ifndef __CLIENTBAT_H__
2 #define __CLIENTBAT_H__
3
4 #include "MeleeWeapon.h"
5
6 /* Clase que representa al arma Bat de baseball */
7 class Bat : public MeleeWeapon {
8 public:
9     /* Constructor */
10    explicit Bat(int ammo);
11
12    /* Destructor */
13    ~Bat();
14
15    /* Constructor por movimiento */
16    Bat(Bat&& other);
17 };
18
19 #endif
```

jun 12, 18 14:03

**Bazooka.cpp**

Page 1/1

```
1 #include "Bazooka.h"
2 #include "WeaponNames.h"
3
4 Bazooka::Bazooka(int ammo) : DistanceWeapon(BAZOOKA_NAME, ammo) {}
5
6 Bazooka::~Bazooka() {}
7
8 Bazooka::Bazooka(Bazooka&& other) : DistanceWeapon(std::move(other)) {}
9
```

jun 12, 18 14:03

**Bazooka.h**

Page 1/1

```
1 #ifndef __CLIENTBAZOOKA_H__
2 #define __CLIENTBAZOOKA_H__
3
4 #include "DistanceWeapon.h"
5
6 /* Clase que representa al arma Bazooka */
7 class Bazooka : public DistanceWeapon {
8 public:
9     /* Constructor */
10     explicit Bazooka(int ammo);
11
12     /* Destructor */
13     ~Bazooka();
14
15     /* Constructor por movimiento */
16     Bazooka(Bazooka&& other);
17 };
18
19 #endif
```

jun 12, 18 14:03

**Dynamite.cpp**

Page 1/1

```

1  #include "Dynamite.h"
2  #include "WeaponNames.h"
3
4  Dynamite::Dynamite(int ammo) : MeleeWeapon(DYNAMITE_NAME, ammo, false, true) {}
5
6  Dynamite::~Dynamite() {}
7
8  Dynamite::Dynamite(Dynamite&& other) : MeleeWeapon(std::move(other)) {}
9

```

jun 12, 18 14:03

**Dynamite.h**

Page 1/1

```

1  #ifndef __CLIENTDYNAMITE_H__
2  #define __CLIENTDYNAMITE_H__
3
4  #include "MeleeWeapon.h"
5
6  /* Clase que representa al arma Dinamita */
7  class Dynamite : public MeleeWeapon {
8  public:
9      /* Constructor */
10     explicit Dynamite(int ammo);
11
12     /* Destructor */
13     ~Dynamite();
14
15     /* Constructor por movimiento */
16     Dynamite(Dynamite&& other);
17 };
18
19 #endif

```

jun 12, 18 14:03

**GreenGrenade.cpp**

Page 1/1

```
1 #include "GreenGrenade.h"
2 #include "WeaponNames.h"
3
4 GreenGrenade::GreenGrenade(int ammo) :
5     DistanceWeapon(GREEN_GRENADE_NAME, ammo, true) {}
6
7 GreenGrenade::~GreenGrenade() {}
8
9 GreenGrenade::GreenGrenade(GreenGrenade&& other) : DistanceWeapon(
10     std::move(other)) {}
11
```

jun 12, 18 14:03

**GreenGrenade.h**

Page 1/1

```
1 #ifndef __CLIENTGREENGRENADE_H__
2 #define __CLIENTGREENGRENADE_H__
3
4 #include "DistanceWeapon.h"
5
6 /* Clase que representa al arma Granada verde */
7 class GreenGrenade : public DistanceWeapon {
8 public:
9     /* Constructor */
10     explicit GreenGrenade(int ammo);
11
12     /* Destructor */
13     ~GreenGrenade();
14
15     /* Constructor por movimiento */
16     GreenGrenade(GreenGrenade&& other);
17 };
18
19 #endif
```

jun 12, 18 14:03

**HolyGrenade.cpp**

Page 1/1

```

1  #include "HolyGrenade.h"
2  #include "WeaponNames.h"
3
4  HolyGrenade::HolyGrenade(int ammo) :
5      DistanceWeapon(HOLY_GRENADE_NAME, ammo, true) {}
6
7  HolyGrenade::~HolyGrenade() {}
8
9  HolyGrenade::HolyGrenade(HolyGrenade&& other) : DistanceWeapon(
10      std::move(other)) {}
11

```

jun 12, 18 14:03

**HolyGrenade.h**

Page 1/1

```

1  #ifndef __CLIENTHOLYGRENADE_H__
2  #define __CLIENTHOLYGRENADE_H__
3
4  #include "DistanceWeapon.h"
5
6  /* Clase que representa al arma Granada santa */
7  class HolyGrenade : public DistanceWeapon {
8  public:
9      /* Constructor */
10     explicit HolyGrenade(int ammo);
11
12     /* Destructor */
13     ~HolyGrenade();
14
15     /* Constructor por movimiento */
16     HolyGrenade(HolyGrenade&& other);
17 };
18
19 #endif

```



jun 12, 18 14:03

**Mortar.cpp**

Page 1/1

```
1 #include "Mortar.h"
2 #include "WeaponNames.h"
3
4 Mortar::Mortar(int ammo) : DistanceWeapon(MORTAR_NAME, ammo, false) {}
5
6 Mortar::~Mortar() {}
7
8 Mortar::Mortar(Mortar&& other) : DistanceWeapon(std::move(other)) {}
9
```

jun 12, 18 14:03

**Mortar.h**

Page 1/1

```
1 #ifndef __CLIENTMORTAR_H__
2 #define __CLIENTMORTAR_H__
3
4 #include "DistanceWeapon.h"
5
6 /* Clase que representa al arma Mortero */
7 class Mortar : public DistanceWeapon {
8 public:
9     /* Constructor */
10     explicit Mortar(int ammo);
11
12     /* Destructor */
13     ~Mortar();
14
15     /* Constructor por movimiento */
16     Mortar(Mortar&& other);
17 };
18
19 #endif
```

jun 12, 18 14:03

**RedGrenade.cpp**

Page 1/1

```
1 #include "RedGrenade.h"
2 #include "WeaponNames.h"
3
4 RedGrenade::RedGrenade(int ammo) :
5     DistanceWeapon(RED_GRENADE_NAME, ammo, true) {}
6
7 RedGrenade::~RedGrenade() {}
8
9 RedGrenade::RedGrenade(RedGrenade&& other) : DistanceWeapon(std::move(other)) {}
10
```

jun 12, 18 14:03

**RedGrenade.h**

Page 1/1

```
1 #ifndef __CLIENTREDGRENADE_H__
2 #define __CLIENTREDGRENADE_H__
3
4 #include "DistanceWeapon.h"
5
6 /* Clase que representa al arma Granada roja */
7 class RedGrenade : public DistanceWeapon {
8 public:
9     /* Constructor */
10     explicit RedGrenade(int ammo);
11
12     /* Destructor */
13     ~RedGrenade();
14
15     /* Constructor por movimiento */
16     RedGrenade(RedGrenade&& other);
17 };
18
19 #endif
```

jun 12, 18 14:03

**Teleportation.cpp**

Page 1/1

```

1  #include "Teleportation.h"
2  #include "WeaponNames.h"
3
4  Teleportation::Teleportation(int ammo) : SelfDirectedWeapon(TELEPORT_NAME,
5                                                    ammo) {}
6
7  Teleportation::~Teleportation() {}
8
9  Teleportation::Teleportation(Teleportation&& other) : SelfDirectedWeapon(
10      std::move(other)) {}
11

```

jun 12, 18 14:03

**Teleportation.h**

Page 1/1

```

1  #ifndef __CLIENTTELEPORTATION_H__
2  #define __CLIENTTELEPORTATION_H__
3
4  #include "SelfDirectedWeapon.h"
5
6  /* Clase que representa al arma Teletransportador */
7  class Teleportation : public SelfDirectedWeapon {
8  public:
9      /* Constructor */
10     explicit Teleportation(int ammo);
11
12     /* Destructor */
13     ~Teleportation();
14
15     /* Constructor por movimiento */
16     Teleportation(Teleportation&& other);
17 };
18
19 #endif

```

jun 12, 18 14:03

## SelfDirectedWeapon.cpp

Page 1/1

```

1  #include "SelfDirectedWeapon.h"
2  #include <string>
3
4  SelfDirectedWeapon::SelfDirectedWeapon(std::string name, int ammo) : Weapon(
5      name, ammo) {}
6
7  SelfDirectedWeapon::~SelfDirectedWeapon() {}
8
9  SelfDirectedWeapon::SelfDirectedWeapon(SelfDirectedWeapon&& other) : Weapon(
10     std::move(other)) {}
11
12 bool SelfDirectedWeapon::isSelfDirected() const {
13     return true;
14 }
15

```

jun 12, 18 14:03

## SelfDirectedWeapon.h

Page 1/1

```

1  #ifndef __SELFDIRECTEDWEAPON_H__
2  #define __SELFDIRECTEDWEAPON_H__
3
4  #include "Weapon.h"
5  #include <string>
6
7  /* Clase que representa las armas teledirigidas */
8  class SelfDirectedWeapon : public Weapon {
9  public:
10     /* Constructor */
11     SelfDirectedWeapon(std::string name, int ammo);
12
13     /* Destructor */
14     ~SelfDirectedWeapon();
15
16     /* Constructor por movimiento */
17     SelfDirectedWeapon(SelfDirectedWeapon&& other);
18
19     /* Devuelve true si es teledirigida */
20     bool isSelfDirected() const override;
21 };
22
23 #endif

```

jun 12, 18 14:03

## Weapon.cpp

Page 1/1

```

1  #include "Weapon.h"
2  #include <string>
3
4  Weapon::Weapon(std::string name, int ammo) :
5      name(name), ammo(ammo), has_Scope(false), is_Timed(false) {}
6
7  Weapon::~Weapon() {}
8
9  Weapon::Weapon(Weapon&& other) {
10     this->name = std::move(other.name);
11     this->ammo = std::move(other.ammo);
12     this->has_Scope = std::move(other.has_Scope);
13     this->is_Timed = std::move(other.is_Timed);
14 }
15
16 Weapon& Weapon::operator=(Weapon&& other) {
17     this->name = std::move(other.name);
18     this->ammo = std::move(other.ammo);
19     this->has_Scope = std::move(other.has_Scope);
20     this->is_Timed = std::move(other.is_Timed);
21     return *this;
22 }
23
24 bool Weapon::hasScope() const {
25     return this->has_Scope;
26 }
27
28 bool Weapon::isSelfDirected() const {
29     return false;
30 }
31
32 bool Weapon::isTimed() const {
33     return this->is_Timed;
34 }
35
36 bool Weapon::hasVariablePower() const {
37     return false;
38 }
39
40 const std::string& Weapon::getName() const {
41     return this->name;
42 }
43
44 void Weapon::shoot() {
45     if (this->ammo <= 100)
46         this->ammo--;
47 }
48
49 bool Weapon::hasAmmo() const {
50     return this->ammo > 0;
51 }
52
53 unsigned int Weapon::getAmmo() const {
54     return this->ammo;
55 }
56

```

jun 12, 18 14:03

## Weapon.h

Page 1/1

```

1  #ifndef __CLIENTWEAPON_H__
2  #define __CLIENTWEAPON_H__
3
4  #include <string>
5
6  /* Clase que se encarga de representar a las armas del juego */
7  class Weapon {
8  protected:
9      std::string name;
10     unsigned int ammo;
11     bool has_Scope;
12     bool is_Timed;
13
14 public:
15     /* Constructor */
16     Weapon(std::string name, int ammo);
17
18     /* Destructor */
19     ~Weapon();
20
21     /* Constructor por movimiento */
22     Weapon(Weapon&& other);
23
24     /* Operador = por movimiento */
25     Weapon& operator=(Weapon&& other);
26
27     /* Devuelve true si el arma tiene mira */
28     virtual bool hasScope() const;
29
30     /* Devuelve true si el arma es teledirigida */
31     virtual bool isSelfDirected() const;
32
33     /* Devuelve true si el arma es por tiempo */
34     virtual bool isTimed() const;
35
36     /* Devuelve true si el arma tiene potencia variable */
37     virtual bool hasVariablePower() const;
38
39     /* Devuelve el nombre del arma */
40     virtual const std::string& getName() const;
41
42     /* Disminuye la cantidad de municiones del arma */
43     virtual void shoot();
44
45     /* Devuelve true si el arma tiene balas */
46     virtual bool hasAmmo() const;
47
48     /* Devuelve la cantidad de balas */
49     unsigned int getAmmo() const;
50 };
51
52 #endif
53

```

jun 12, 18 14:03

## WeaponList.cpp

Page 1/1

```

1  #include "WeaponList.h"
2  #include <utility>
3  #include <string>
4  #include "WeaponNames.h"
5
6  WeaponList::WeaponList() : current_weapon(DEFAULT_WEAPON) {}
7
8  WeaponList::~WeaponList() {}
9
10 void WeaponList::add(std::string weapon, int ammo) {
11     WeaponsFactory factory;
12     this->weapons.insert(std::pair<std::string, weapon_ptr>(weapon, std::move(
13         factory.createWeapon(weapon, ammo))));
14 }
15
16 void WeaponList::changeWeapon(std::string weapon) {
17     this->current_weapon = weapon;
18 }
19
20 Weapon& WeaponList::getCurrentWeapon() {
21     return *this->weapons.at(this->current_weapon);
22 }
23
24 WeaponList::iterator WeaponList::begin() {
25     return this->weapons.begin();
26 }
27
28 WeaponList::iterator WeaponList::end() {
29     return this->weapons.end();
30 }
31

```

jun 12, 18 14:03

## WeaponList.h

Page 1/1

```

1  #ifndef __CLIENTWEAPONLIST_H__
2  #define __CLIENTWEAPONLIST_H__
3
4  #include <map>
5  #include <string>
6  #include "Weapon.h"
7  #include "WeaponsFactory.h"
8
9  /* Clase que se encarga de almacenar las armas del juego */
10 class WeaponList {
11 private:
12     typedef std::map<std::string, weapon_ptr> WeaponsList;
13     WeaponsList weapons;
14     std::string current_weapon;
15
16 public:
17     /* Constructor */
18     WeaponList();
19
20     /* Destructor */
21     ~WeaponList();
22
23
24     /* Agrega un arma a la lista */
25     void add(std::string weapon, int ammo);
26
27     /* Devuelve el arma actual */
28     Weapon& getCurrentWeapon();
29
30     /* Cambia el arma actual por la especificada */
31     void changeWeapon(std::string weapon);
32
33     typedef WeaponsList::iterator iterator;
34     typedef WeaponsList::const_iterator const_iterator;
35
36     iterator begin();
37
38     iterator end();
39 };
40
41
42 #endif

```

jun 12, 18 14:03

**WeaponsFactory.cpp**

Page 1/1

```

1  #include "WeaponsFactory.h"
2  #include "WeaponNames.h"
3  #include <string>
4  #include "AirAttack.h"
5  #include "Banana.h"
6  #include "Bat.h"
7  #include "Bazooka.h"
8  #include "Dynamite.h"
9  #include "GreenGrenade.h"
10 #include "HolyGrenade.h"
11 #include "Mortar.h"
12 #include "RedGrenade.h"
13 #include "Teleportation.h"
14
15 WeaponsFactory::WeaponsFactory() {}
16
17 WeaponsFactory::~WeaponsFactory() {}
18
19
20 weapon_ptr WeaponsFactory::createWeapon(std::string weapon, int ammo) {
21     if (weapon == AIR_ATTACK_NAME)
22         return weapon_ptr(new AirAttack(ammo));
23     else if (weapon == BANANA_NAME)
24         return weapon_ptr(new Banana(ammo));
25     else if (weapon == BAT_NAME)
26         return weapon_ptr(new Bat(ammo));
27     else if (weapon == BAZOOKA_NAME)
28         return weapon_ptr(new Bazooka(ammo));
29     else if (weapon == DYNAMITE_NAME)
30         return weapon_ptr(new Dynamite(ammo));
31     else if (weapon == GREEN_GRENADE_NAME)
32         return weapon_ptr(new GreenGrenade(ammo));
33     else if (weapon == HOLY_GRENADE_NAME)
34         return weapon_ptr(new HolyGrenade(ammo));
35     else if (weapon == MORTAR_NAME)
36         return weapon_ptr(new Mortar(ammo));
37     else if (weapon == RED_GRENADE_NAME)
38         return weapon_ptr(new RedGrenade(ammo));
39     return weapon_ptr(new Teleportation(ammo));
40 }

```

jun 12, 18 14:03

**WeaponsFactory.h**

Page 1/1

```

1  #ifndef __CLIENTWEAPONSFACTORY_H__
2  #define __CLIENTWEAPONSFACTORY_H__
3
4  #include <memory>
5  #include <string>
6  #include "Weapon.h"
7
8  typedef std::unique_ptr<Weapon> weapon_ptr;
9
10 /* Clase que se encarga de crear las armas del juego */
11 class WeaponsFactory {
12 public:
13     /* Constructor */
14     WeaponsFactory();
15
16     /* Destructor */
17     ~WeaponsFactory();
18
19
20     /* Crea el arma especificada con las municiones especificadas */
21     weapon_ptr createWeapon(std::string weapon, int ammo);
22 };
23
24
25 #endif

```

jun 12, 18 14:03

## MusicPath.h

Page 1/1

```

1  #ifndef WORMS_MUSICPATH_H
2  #define WORMS_MUSICPATH_H
3
4  #include <string>
5  #include "Path.h"
6
7  const std::string BACKGROUND_MUSIC = SOUNDS_PATH + "BackgroundMusic.mp3";
8  const std::string START_TURN_SOUND = SOUNDS_PATH + "Misc/StartRound.wav";
9  const std::string TICK_SOUND = SOUNDS_PATH + "Misc/TimerTick.wav";
10 const std::string RUN_AWAY_SOUND = SOUNDS_PATH + "Worms/RunAway.wav";
11 const std::string DEATH_SOUND = SOUNDS_PATH + "Worms/Death.wav";
12 const std::string DAMAGE_RECEIVE_SOUND =
13     SOUNDS_PATH + "Worms/DamageReceive.wav";
14 const std::string EXPLOSION_SOUND = SOUNDS_PATH + "Weapons/Explosion.wav";
15 const std::string TELEPORT_SOUND = SOUNDS_PATH + "Weapons/Teleportation.wav";
16 const std::string BAT_SOUND = SOUNDS_PATH + "Weapons/BaseballSound.wav";
17 const std::string HOLY_GRENADE_SOUND = SOUNDS_PATH + "Weapons/HolyGrenade.wav";
18 const std::string AIR_ATTACK_SOUND = SOUNDS_PATH + "Weapons/AirAttack.wav";
19 const std::string SHOOT_SOUND = SOUNDS_PATH + "Weapons/ShootWeapon.wav";
20 const std::string ROLLBACK_SOUND = SOUNDS_PATH + "Misc/RollBack.wav";
21 const std::string JUMP_SOUND = SOUNDS_PATH + "Misc/Jump.wav";
22 const std::string SELECT_WEAPON_SOUND = SOUNDS_PATH + "Misc/SelectWeapon.wav";
23 const std::string NO_AMMO_SOUND = SOUNDS_PATH + "Misc/NoAmmo.wav";
24 const std::string VICTORY_SOUND = SOUNDS_PATH + "Worms/Victory.WAV";
25
26 #endif //WORMS_MUSICPATH_H

```

jun 12, 18 14:03

## MusicPlayer.cpp

Page 1/3

```

1  #include "MusicPlayer.h"
2  #include <map>
3  #include <string>
4  #include "MusicPlayerException.h"
5  #include "WeaponNames.h"
6  #include "Protocol.h"
7  #include "MusicPath.h"
8
9  MusicPlayer::MusicPlayer() {
10     this->music = NULL;
11     // Initialize SDL.
12     if (SDL_Init(SDL_INIT_AUDIO) < 0) {
13         throw MusicPlayerException("Error al inicializar SDL");
14     }
15
16     //Initialize SDL_mixer
17     if (Mix_OpenAudio(22050, MIX_DEFAULT_FORMAT, 2, 4096) == -1) {
18         throw MusicPlayerException("Error al inicializar SDL mixer");
19     }
20
21     // Load background music
22     this->music = Mix_LoadMUS(BACKGROUND_MUSIC.c_str());
23     if (this->music == NULL) {
24     }
25 }
26
27 MusicPlayer::~MusicPlayer() {
28     Mix_HaltChannel(-1);
29     this->stop();
30     if (this->music != NULL) {
31         Mix_FreeMusic(this->music);
32     }
33     std::map<int, Mix_Chunk*>::iterator iter;
34     for (iter = this->effects.begin(); iter != this->effects.end(); iter++) {
35         Mix_FreeChunk(iter->second);
36     }
37     // quit SDL_mixer
38     Mix_CloseAudio();
39     Mix_Quit();
40     SDL_Quit();
41 }
42
43 void MusicPlayer::check(int channel) {
44     if (this->effects.find(channel) != this->effects.end()) {
45         // elimino el audio anterior de este canal
46         Mix_FreeChunk(this->effects.at(channel));
47         this->effects.erase(channel);
48     }
49     std::map<int, Mix_Chunk*>::iterator iter = this->effects.begin();
50     while (iter != this->effects.end()) {
51         if (!Mix_Playing(iter->first)) {
52             Mix_FreeChunk(iter->second);
53             iter = this->effects.erase(iter);
54         } else {
55             iter++;
56         }
57     }
58 }
59
60 void MusicPlayer::addEffect(const std::string& audio) {
61     int channel;
62     Mix_Chunk* effect = NULL;
63     effect = Mix_LoadWAV(audio.c_str());
64     if (effect == NULL) {
65         return;
66     }

```



jun 12, 18 14:03

**MusicPlayer.cpp**

Page 2/3

```

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);

```

jun 12, 18 14:03

**MusicPlayer.cpp**

Page 3/3

```

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 12, 18 14:03

**MusicPlayerException.cpp**

Page 1/1

```

1  #include "MusicPlayerException.h"
2  #include <string>
3
4  MusicPlayerException::MusicPlayerException(std::string msg) : msg(msg) {
5      this->msg.insert(0, "Error en Music Player: ");
6  }
7
8  MusicPlayerException::~MusicPlayerException() {}
9
10 const char* MusicPlayerException::what() const noexcept {
11     return this->msg.c_str();
12 }

```

jun 12, 18 14:03

**MusicPlayerException.h**

Page 1/1

```

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

```

jun 12, 18 14:03	MusicPlayer.h	Page 1/2
1	<b>#ifndef</b> __MUSICPLAYER_H__	
2	<b>#define</b> __MUSICPLAYER_H__	
3		
4	<b>#include</b> <SDL2/SDL.h>	
5	<b>#include</b> <SDL2/SDL_mixer.h>	
6	<b>#include</b> <map>	
7	<b>#include</b> <string>	
8		
9	<i>/* Clase que se encarga de reproducir musica y efectos</i>	
10	<i>* de sonido */</i>	
11	class MusicPlayer {	
12	private:	
13	Mix_Music* music; <i>// Musica de fondo</i>	
14	std::map<int, Mix_Chunk*> effects;	
15		
16	<i>/* Verifica si algunos efectos de la lista finalizaon y los</i>	
17	<i>* libera. Adem�s libera el efecto que se encuentre guardado</i>	
18	<i>* en la lista con clave channel */</i>	
19	void check(int channel);	
20		
21	<i>/* Agrega un nuevo efecto a la lista y lo reproduce */</i>	
22	void addEffect( <b>const</b> std::string& audio);	
23		
24	public:	
25	<i>/* Constructor */</i>	
26	MusicPlayer();	
27		
28	<i>/* Destructor */</i>	
29	~MusicPlayer();	
30		
31	<i>/* Reproduce la musica de fondo */</i>	
32	void playMusic();	
33		
34	<i>/* Reproduce el sonido de inicio de turno */</i>	
35	void playStartTurnSound();	
36		
37	<i>/* Reproduce el sonido de falta de tiempo */</i>	
38	void playTickSound();	
39		
40	<i>/* Reproduce el sonido de muerte de un worm */</i>	
41	void playDeathSound();	
42		
43	<i>/* Reproduce el sonido de da�o recibido */</i>	
44	void playDamageReceiveSound();	
45		
46	<i>/* Reproduce el sonido de la explosion */</i>	
47	void playExplosionSound( <b>const</b> std::string& weapon);	
48		
49	<i>/* Reproduce el sonido de arma disparada */</i>	
50	void playWeaponShotSound( <b>const</b> std::string& weapon);	
51		
52	<i>/* Reproduce el sonido de salto o rollback */</i>	
53	void playJumpSound(char action);	
54		
55	<i>/* Reproduce el sonido de arma seleccionada */</i>	
56	void playSelectWeaponSound();	
57		
58	<i>/* Reproduce el sonido de victoria */</i>	
59	void playVictory();	
60		
61	<i>/* Reproduce el sonido de arma descargada */</i>	
62	void playNoAmmo();	
63		
64	<i>/* Detiene la reproduccion de la musica de fondo */</i>	
65	void stop();	
66	};	

jun 12, 18 14:03	MusicPlayer.h	Page 2/2
67		
68		
69	<b>#endif</b>	

jun 12, 18 14:03

## ExplosionView.cpp

Page 1/1

```

1  #include "ExplosionView.h"
2  #include <gtkmm/image.h>
3  #include <glibmm/main.h>
4  #include "Path.h"
5
6  ExplosionView::ExplosionView(BulletView&& bullet) : bulletView(
7      std::move(bullet)) {
8      this->animation = Gdk::Pixbuf::create_from_file(EXPLOSION_ANIMATION);
9      int width = this->animation->get_width();
10     int height = this->animation->get_height();
11     for (int i = 0; i < height / width; i++) {
12         Glib::RefPtr<Gdk::Pixbuf> aux = Gdk::Pixbuf::create_subpixbuf(
13             this->animation, 0, i * width, width, width);
14         this->animation_vector.push_back(aux);
15     }
16     this->iter = this->animation_vector.begin();
17 }
18
19 ExplosionView::~ExplosionView() {}
20
21 ExplosionView::ExplosionView(ExplosionView&& other) :
22     bulletView(std::move(other.bulletView)) {
23     this->animation_vector = other.animation_vector;
24     this->animation = other.animation;
25     this->iter = this->animation_vector.begin();
26 }
27
28 bool ExplosionView::startCallBack() {
29     Gtk::Image& image = (Gtk::Image&) this->bulletView.getWidget();
30     image.set(*(this->iter));
31     this->iter++;
32     if (this->iter == this->animation_vector.end()) {
33         this->bulletView.removeFromWorld();
34         return false;
35     }
36     return true;
37 }
38
39 void ExplosionView::start() {
40     Glib::signal_timeout().connect(
41         sigc::mem_fun(*this, &ExplosionView::startCallBack), 40);
42 }
43
44 bool ExplosionView::hasFinished() {
45     return this->iter == this->animation_vector.end();
46 }

```

jun 12, 18 14:03

## ExplosionView.h

Page 1/1

```

1  #ifndef __CLIENTEXPLOSIONVIEW_H__
2  #define __CLIENTEXPLOSIONVIEW_H__
3
4  #include <vector>
5  #include <gdkmm/pixbuf.h>
6  #include "BulletView.h"
7
8  /* Clase que se encarga de reproducir la animacion de una explosion */
9  class ExplosionView {
10 private:
11     BulletView bulletView;
12     std::vector<Glib::RefPtr<Gdk::Pixbuf>> animation_vector;
13     Glib::RefPtr<Gdk::Pixbuf> animation;
14     std::vector<Glib::RefPtr<Gdk::Pixbuf>>::iterator iter;
15
16     /* Callback de start */
17     bool startCallBack();
18
19 public:
20     /* Constructor */
21     explicit ExplosionView(BulletView&& bullet);
22
23     /* Destructor */
24     ~ExplosionView();
25
26     /* Constructor por movimiento */
27     ExplosionView(ExplosionView&& other);
28
29     /* Realiza la animacion de la explosion */
30     void start();
31
32     /* Devuelve true si la animacion de la explosion finalizo */
33     bool hasFinished();
34 };
35
36
37
38 #endif

```

jun 12, 18 14:03

## ExplosionViewList.cpp

Page 1/1

```

1  #include "ExplosionViewList.h"
2  #include <list>
3
4  ExplosionViewList::ExplosionViewList() {}
5
6  ExplosionViewList::~ExplosionViewList() {}
7
8  void ExplosionViewList::check() {
9      std::list<ExplosionView>::iterator iter;
10     iter = this->animations.begin();
11     while (iter != this->animations.end()) {
12         if (iter->hasFinished()) {
13             iter = this->animations.erase(iter);
14         } else {
15             ++iter;
16         }
17     }
18 }
19
20 void ExplosionViewList::addAndStart(ExplosionView&& animation) {
21     this->check();
22     this->animations.push_back(std::move(animation));
23     this->animations.back().start();
24 }

```

jun 12, 18 14:03

## ExplosionViewList.h

Page 1/1

```

1  #ifndef WORMS_EXPLOSIONVIEWLIST_H
2  #define WORMS_EXPLOSIONVIEWLIST_H
3
4  #include <list>
5  #include "ExplosionView.h"
6
7  /* Clase que se encarga de almacenar animaciones de explosiones */
8  class ExplosionViewList {
9  private:
10     std::list<ExplosionView> animations;
11
12     /* Verifica si alguna animacion de la lista finalizo y las
13      * elimina de la lista */
14     void check();
15
16  public:
17     /* Constructor */
18     ExplosionViewList();
19
20     /* Destructor */
21     ~ExplosionViewList();
22
23
24     /* Agrega una animacion de explosion a la lista y la reproduce */
25     void addAndStart(ExplosionView&& animation);
26 };
27
28
29 #endif //WORMS_EXPLOSIONVIEWLIST_H

```

jun 12, 18 14:03

## WalkingAnimation.cpp

Page 1/1

```

1  #include "WalkingAnimation.h"
2  #include "Path.h"
3  #include "ObjectSizes.h"
4
5  #define DIR_RIGHT 1
6  #define DIR_LEFT -1
7
8  WalkingAnimation::WalkingAnimation(Gtk::Image* worm_image) :
9      worm_image(worm_image), dir(DIR_RIGHT) {
10      this->walk_image = Gdk::Pixbuf::create_from_file(WORMS_PATH + "walk.png");
11      int width = this->walk_image->get_width();
12      int height = this->walk_image->get_height();
13      for (int i = 0; i < height / WORM_IMAGE_WIDTH; i++) {
14          walk_queue.push(Gdk::Pixbuf::create_subpixbuf(this->walk_image, 0,
15                                                         i * WORM_IMAGE_WIDTH, width, WORM_IMAGE_WIDTH));
16      }
17  }
18
19  WalkingAnimation::~WalkingAnimation() {}
20
21  WalkingAnimation::WalkingAnimation(WalkingAnimation&& other) :
22      walk_queue(std::move(other.walk_queue)),
23      walk_image(std::move(other.walk_image)),
24      worm_image(other.worm_image), dir(other.dir) {}
25
26  void WalkingAnimation::setMovementImage(char new_dir) {
27      if (new_dir == this->dir) {
28          this->walk_queue.push(std::move(this->walk_queue.front()));
29          this->walk_queue.pop();
30      }
31      this->setStaticImage(new_dir);
32  }
33
34  void WalkingAnimation::setStaticImage(char new_dir) {
35      this->dir = new_dir;
36      this->worm_image->set(Gdk::Pixbuf::create_subpixbuf(this->walk_queue.back(),
37                                                         WORM_IMAGE_WIDTH + this->dir * WORM_IMAGE_WIDTH, 0,
38                                                         WORM_IMAGE_WIDTH, WORM_IMAGE_WIDTH));
39  }
40
41  void WalkingAnimation::updateWormImage(Gtk::Image* worm_image) {
42      this->worm_image = worm_image;
43  }
44
45  char WalkingAnimation::getDir() const {
46      return this->dir;
47  }

```

jun 12, 18 14:03

## WalkingAnimation.h

Page 1/1

```

1  #ifndef WORMS_WALKINGANIMATION_H
2  #define WORMS_WALKINGANIMATION_H
3
4  #include <gtkmm/image.h>
5  #include <gdkmm/pixbuf.h>
6  #include <queue>
7
8  /* Clase que se encarga de actualizar la imagen del worm al
9   * moverse obteniendo una animacion del worm caminando */
10 class WalkingAnimation {
11 private:
12     std::queue<Glib::RefPtr<Gdk::Pixbuf>> walk_queue;
13     Glib::RefPtr<Gdk::Pixbuf> walk_image;
14     Gtk::Image* worm_image;
15     char dir;
16
17 public:
18     /* Constructor */
19     explicit WalkingAnimation(Gtk::Image* worm_image);
20
21     /* Destructor */
22     ~WalkingAnimation();
23
24     /* Constructor por movimiento */
25     WalkingAnimation(WalkingAnimation&& other);
26
27
28     /* Actualiza la imagen del worm por la siguiente
29      * imagen del worm caminando */
30     void setMovementImage(char new_dir);
31
32     /* Setea la imagen del worm por la imagen actual del
33      * worm caminando */
34     void setStaticImage(char new_dir);
35
36     /* Devuelve la direccion del worm */
37     char getDir() const;
38
39     /* Actualiza el puntero de la imagen del worm */
40     void updateWormImage(Gtk::Image* worm_image);
41 };
42
43
44 #endif //WORMS_WALKINGANIMATION_H

```

jun 26, 18 13:37

## WeaponAnimation.cpp

Page 1/2

```

1  #include "WeaponAnimation.h"
2  #include <glibmm/main.h>
3  #include <string>
4  #include <vector>
5  #include "WormView.h"
6  #include "Path.h"
7  #include "ObjectSizes.h"
8  #include "WeaponNames.h"
9
10 #define DIR_RIGHT 1
11
12 WeaponAnimation::WeaponAnimation(const std::string& weapon,
13                                Gtk::Image* worm_image) :
14     worm_image(worm_image), angle(DEFAULT_ANGLE) {
15     this->updateWeaponImage(weapon);
16 }
17
18 WeaponAnimation::~WeaponAnimation() {
19     this->stopWeaponShootAnimation();
20 }
21
22 WeaponAnimation::WeaponAnimation(WeaponAnimation&& other) :
23     scope_vector(std::move(other.scope_vector)),
24     scope_image(std::move(other.scope_image)),
25     worm_image(other.worm_image),
26     angle(other.angle) {}
27
28 void WeaponAnimation::updateWeaponImage(const std::string& weapon) {
29     this->scope_vector.clear();
30     this->scope_image = Gdk::Pixbuf::create_from_file(
31         WORMS_PATH + weapon + "_scope.png");
32     int width = this->scope_image->get_width();
33     int height = this->scope_image->get_height();
34     for (int i = 0; i < height / WORM_IMAGE_WIDTH; i++) {
35         this->scope_vector.push_back(
36             Gdk::Pixbuf::create_subpixbuf(scope_image, 0,
37                                           i * WORM_IMAGE_WIDTH, width,
38                                           WORM_IMAGE_WIDTH));
39     }
40 }
41
42 void WeaponAnimation::changeWeapon(const std::string& weapon, char dir) {
43     this->stopWeaponShootAnimation();
44     this->updateWeaponImage(weapon);
45     this->setWeaponImage(dir);
46 }
47
48 void WeaponAnimation::setWeaponImage(char dir) {
49     int width = this->scope_vector[(90 + this->angle) / 6]->get_width() / 3;
50     int height = this->scope_vector[(90 + this->angle) / 6]->get_height();
51     this->worm_image->set(Gdk::Pixbuf::create_subpixbuf(
52         this->scope_vector[(90 + this->angle) / 6], width + dir * width, 0,
53         width, height));
54 }
55
56 bool WeaponAnimation::batHitCallBack(
57     std::vector<Glib::RefPtr<Gdk::Pixbuf>>::iterator& iter, const int width,
58     char dir) {
59     this->worm_image->set(Gdk::Pixbuf::create_subpixbuf(*iter, 0, 0, width,
60                                                         WORM_IMAGE_WIDTH));
61     ++iter;
62     if (iter == this->scope_vector.end()) {
63         this->updateWeaponImage(BAT_NAME);
64         this->setWeaponImage(dir);
65         return false;
66     }

```

jun 26, 18 13:37

## WeaponAnimation.cpp

Page 2/2

```

67     return true;
68 }
69
70 void WeaponAnimation::weaponShootAnimation(const std::string& weapon, char dir){
71     if (weapon != BAT_NAME) {
72         return;
73     }
74     this->scope_image = Gdk::Pixbuf::create_from_file(BAT_HIT_ANIMATION);
75     int width = this->scope_image->get_width() / 3;
76     int height = this->scope_image->get_height();
77     int pos_x = width + dir * width;
78     this->scope_vector.clear();
79     for (int i = 0; i < height / WORM_IMAGE_WIDTH; i++) {
80         this->scope_vector.push_back(
81             Gdk::Pixbuf::create_subpixbuf(scope_image, pos_x,
82                                           i * WORM_IMAGE_WIDTH, width, WORM_IMAGE_WIDTH));
83     }
84     std::vector<Glib::RefPtr<Gdk::Pixbuf>>::iterator iter;
85     iter = this->scope_vector.begin();
86     sigc::slot<bool> my_slot = sigc::bind(
87         sigc::mem_fun(*this, &WeaponAnimation::batHitCallBack), iter, width,
88         dir);
89     this->my_connection = Glib::signal_timeout().connect(my_slot, 12);
90 }
91
92 void WeaponAnimation::changeAngle(int angle, char dir) {
93     this->angle = angle;
94     this->setWeaponImage(dir);
95 }
96
97 void WeaponAnimation::updateWormImage(Gtk::Image* worm_image) {
98     this->worm_image = worm_image;
99 }
100
101 void WeaponAnimation::stopWeaponShootAnimation() {
102     if (this->my_connection.connected()) {
103         this->my_connection.disconnect();
104         this->updateWeaponImage(BAT_NAME);
105     }
106 }

```

jun 26, 18 13:06

## WeaponAnimation.h

Page 1/1

```

1  #ifndef WORMS_WEAPONANIMATION_H
2  #define WORMS_WEAPONANIMATION_H
3
4  #include <gtkmm/image.h>
5  #include <gdkmm/pixbuf.h>
6  #include <vector>
7  #include <string>
8
9  class WormView;
10
11  /* Clase que se encarga de controlar las animaciones
12   * de las armas */
13  class WeaponAnimation {
14  private:
15      std::vector<Glib::RefPtr<Gdk::Pixbuf>> scope_vector;
16      Glib::RefPtr<Gdk::Pixbuf> scope_image;
17      Gtk::Image* worm_image;
18      int angle;
19      sigc::connection my_connection;
20
21      /* Actualiza las imagenes por las imagenes del arma nueva */
22      void updateWeaponImage(const std::string& weapon);
23
24      /* Callback */
25      bool batHitCallBack(std::vector<Glib::RefPtr<Gdk::Pixbuf>>::iterator& iter,
26                          const int width, char dir);
27
28  public:
29      /* Constructor */
30      WeaponAnimation(const std::string& weapon, Gtk::Image* worm_image);
31
32      /* Destructor */
33      ~WeaponAnimation();
34
35      /* Constructor por movimiento */
36      WeaponAnimation(WeaponAnimation&& other);
37
38
39      /* Cambia la imagen del worm con el arma actual por una imagen
40       * del worm con la nueva arma */
41      void changeWeapon(const std::string& weapon, char dir);
42
43      /* Setea la imagen del worm con el arma actual apuntando
44       * con el angulo especifico */
45      void setWeaponImage(char dir);
46
47      /* Realiza la animacion del disparo del arma */
48      void weaponShootAnimation(const std::string& weapon, char dir);
49
50      /* Actualiza el angulo, cambiando la imagen del arma
51       * por la correspondiente */
52      void changeAngle(int angle, char dir);
53
54      /* Actualiza el puntero de la imagen del worm */
55      void updateWormImage(Gtk::Image* worm_image);
56
57      /* Detiene la animacion del disparo del arma */
58      void stopWeaponShootAnimation();
59  };
60
61
62  #endif //WORMS_WEAPONANIMATION_H

```

jun 25, 18 19:28

## PlayerLifeLabel.cpp

Page 1/1

```

1  #include "PlayerLifeLabel.h"
2  #include <string>
3  #include "GamePlayers.h"
4
5  const std::string begining("<span color='";
6  const std::string middle("<span font_family='monospace'><b>");
7  const std::string ending("</b></span>");
8
9  PlayerLifeLabel::PlayerLifeLabel() : id(0), player_name(""), life(0) {
10      this->info_label.set_use_markup(true);
11      this->id_label.set_use_markup(true);
12      this->container.pack_start(this->id_label, Gtk::PACK_SHRINK);
13      this->container.pack_start(this->info_label, Gtk::PACK_SHRINK);
14  }
15
16  PlayerLifeLabel::~PlayerLifeLabel() {}
17
18  void PlayerLifeLabel::setPlayerName(int id, const std::string& player_name) {
19      this->id = id;
20      this->player_name = player_name;
21      this->updateLabel();
22  }
23
24  void PlayerLifeLabel::addLife(int life) {
25      this->life += life;
26      this->updateLabel();
27  }
28
29  void PlayerLifeLabel::reduceLife(int life) {
30      this->life -= life;
31      this->updateLabel();
32  }
33
34  Gtk::HBox& PlayerLifeLabel::getContainer() {
35      return this->container;
36  }
37
38  void PlayerLifeLabel::updateLabel() {
39      std::string id_message = begining + "white" + middle;
40      id_message += std::to_string(this->id) + " " + ending;
41      this->id_label.set_markup(id_message);
42      this->id_label.override_background_color(Gdk::RGBA(colors[this->id]));
43      std::string message = begining + "black" + middle + this->player_name;
44      message += ": " + std::to_string(this->life) + ending;
45      this->info_label.set_markup(message);
46  }

```



jun 25, 18 19:28

## PlayerLifeLabel.h

Page 1/1

```

1  #ifndef __PLAYERLIFELABEL_H__
2  #define __PLAYERLIFELABEL_H__
3
4  #include <gtkmm/label.h>
5  #include <gtkmm/hvbox.h>
6  #include <string>
7
8  /* Clase que se encarga de controlar el indicador de vida del jugador */
9  class PlayerLifeLabel {
10 private:
11     int id;
12     std::string player_name;
13     int life;
14     Gtk::HBox container;
15     Gtk::Label info_label;
16     Gtk::Label id_label;
17
18     /* Actualiza la informacion del label */
19     void updateLabel();
20
21 public:
22     /* Constructor */
23     PlayerLifeLabel();
24
25     /* Destructor */
26     ~PlayerLifeLabel();
27
28
29     /* Establece el nombre del jugador */
30     void setPlayerName(int id, const std::string& player_name);
31
32     /* Agrega la vida al label */
33     void addLife(int life);
34
35     /* Disminuye la vida y actualiza la vista del label */
36     void reduceLife(int life);
37
38     /* Devuelve el contenedor con la informacion del jugador */
39     Gtk::HBox& getContainer();
40 };
41
42
43 #endif

```

jun 25, 18 19:28

## PlayersList.cpp

Page 1/1

```

1  #include "PlayersList.h"
2  #include <glibmm/main.h>
3  #include <string>
4
5  #define SPACING 20
6
7  PlayersList::PlayersList() : container(false, SPACING) {
8      this->title.set_use_markup(true);
9      this->title.set_markup("<span><b><u>Jugadores</u></b></span>");
10     this->container.pack_start(this->title, Gtk::PACK_SHRINK);
11 }
12
13 PlayersList::~PlayersList() {}
14
15 void PlayersList::addPlayer(int id, const std::string& name) {
16     sigc::slot<bool> my_slot = sigc::bind(
17         sigc::mem_fun(*this, &PlayersList::addPlayerCallBack), id, name);
18     Glib::signal_idle().connect(my_slot);
19 }
20
21 bool PlayersList::addPlayerCallBack(int id, std::string name) {
22     this->players[id] = name;
23     this->labels[id].setPlayerName(id, name);
24     this->container.pack_start(this->labels[id].getContainer(), Gtk::PACK_SHRINK);
25     return false;
26 }
27
28 const std::string& PlayersList::getPlayer(int id) const {
29     return this->players.at(id);
30 }
31
32 Gtk::Container& PlayersList::getWindow() {
33     return this->container;
34 }
35
36 void PlayersList::addPlayerLife(int player_id, int life) {
37     this->labels[player_id].addLife(life);
38 }
39
40 void PlayersList::reducePlayerLife(int player_id, int life) {
41     this->labels[player_id].reduceLife(life);
42 }

```

jun 12, 18 14:03

## PlayersList.h

Page 1/1

```

1  #ifndef __PLAYERSLIST_H__
2  #define __PLAYERSLIST_H__
3
4  #include <map>
5  #include <string>
6  #include <gtkmm/hvbox.h>
7  #include <gtkmm/label.h>
8  #include "PlayerLifeLabel.h"
9
10 /* Clase que se encarga de almacenar los nombres y las vidas
11 * de todos los jugadores */
12 class PlayersList {
13 private:
14     std::map<int, std::string> players;
15     std::map<int, PlayerLifeLabel> labels;
16     Gtk::VBox container;
17     Gtk::Label title;
18
19     bool addPlayerCallBack(int id, std::string name);
20
21 public:
22     /* Constructor */
23     PlayersList();
24
25     /* Destructor */
26     ~PlayersList();
27
28     /* Agrega al jugador a la lista de jugadores y agrega su
29     * informacion a la vista */
30     void addPlayer(int id, const std::string& name);
31
32     /* Devuelve el nombre del jugador */
33     const std::string& getPlayer(int id) const;
34
35     /* Devuelve el contenedor de los jugadores */
36     Gtk::Container& getWindow();
37
38     /* Agrega la informacion de la vida del jugador a la vista */
39     void addPlayerLife(int player_id, int life);
40
41     /* Reduce la vida del jugador y actualiza la vista */
42     void reducePlayerLife(int player_id, int life);
43 };
44
45 #endif

```

jun 12, 18 14:03

## ScreenView.cpp

Page 1/2

```

1  #include "ScreenView.h"
2  #include "ServerFatalError.h"
3  #include <glibmm/main.h>
4  #include <string>
5
6  #define PADDING 10
7  #define SPACING 30
8
9  ScreenView::ScreenView(Gtk::Window& window, MenuView& main_menu, Player& player,
10                        WeaponList& weapons) :
11      left_view(false, SPACING), window(window),
12      weapons_view(weapons, player),
13      victory_view(window, main_menu) {
14      this->left_view.pack_start(this->wind_view.getWindow(), Gtk::PACK_SHRINK);
15      this->left_view.pack_start(this->players.getWindow(), Gtk::PACK_SHRINK);
16      this->world_box.pack_start(this->left_view, Gtk::PACK_SHRINK, PADDING);
17      this->world_box.pack_start(this->world.getContainer());
18      this->world_box.pack_end(this->weapons_view.getWindow(), Gtk::PACK_SHRINK);
19
20      this->screen.pack_start(this->turn_label.getWindow(), Gtk::PACK_SHRINK);
21      this->screen.pack_end(this->world_box);
22  }
23
24  ScreenView::~ScreenView() {}
25
26  void ScreenView::show() {
27      sigc::slot<bool> my_slot = sigc::mem_fun(*this, &ScreenView::showCallBack);
28      Glib::signal_idle().connect(my_slot);
29  }
30
31  bool ScreenView::showCallBack() {
32      this->weapons_view.update();
33      this->window.remove();
34      this->window.add(this->screen);
35      this->window.show_all();
36      return false;
37  }
38
39  void ScreenView::close() {
40      sigc::slot<bool> my_slot = sigc::mem_fun(*this, &ScreenView::closeCallBack);
41      Glib::signal_idle().connect(my_slot);
42  }
43
44  bool ScreenView::closeCallBack() {
45      ServerFatalError error(this->window);
46      return false;
47  }
48
49  WorldView& ScreenView::getWorld() {
50      return this->world;
51  }
52
53  WeaponView& ScreenView::getWeaponsView() {
54      return this->weapons_view;
55  }
56
57  TurnLabel& ScreenView::getTurnLabel() {
58      return this->turn_label;
59  }
60
61  PlayersList& ScreenView::getPlayersView() {
62      return this->players;
63  }
64
65  WindView& ScreenView::getWindView() {
66      return this->wind_view;

```

jun 12, 18 14:03

## 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 12, 18 14:03

## 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 12, 18 14:03

## 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 12, 18 14:03

## 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 12, 18 14:03

## TurnLabel.h

Page 1/1

```

1  #ifndef __TURNLABEL_H__
2  #define __TURNLABEL_H__
3
4  #include <gtkmm/hvbox.h>
5  #include <gtkmm/label.h>
6  #include <string>
7
8  /* Clase que se encarga de controlar los labels que indican
9   * el estado del turno */
10 class TurnLabel {
11 private:
12     Gtk::Label message;
13     Gtk::Label time;
14     Gtk::HBox label;
15
16 public:
17     /* Constructor */
18     TurnLabel();
19
20     /* Destructor */
21     ~TurnLabel();
22
23
24     /* Cambia el label indicando que es el turno del jugador */
25     void beginTurn();
26
27     /* Cambia el label indicando que es el turno del jugador
28      * con nombre pasado por parametro */
29     void beginTurn(const std::string& player_name);
30
31     /* Cambia el label indicando que finalizo el turno del jugador */
32     void endTurn();
33
34     /* Cambia el label mostrando al ganador */
35     void setEndGame();
36
37     /* Cambia el label de tiempo al tiempo pasado por parametro */
38     void setTime(int time);
39
40     /* Devuelve el contenedor de la vista */
41     Gtk::Container& getWindow();
42 };
43
44
45 #endif

```

jun 25, 18 19:28

## VictoryWindow.cpp

Page 1/2

```

1  #include "VictoryWindow.h"
2  #include <gtkmm/builder.h>
3  #include <string>
4  #include "Path.h"
5  #include "ButtonBuilder.h"
6
7  const std::string begining("<span color='black'>");
8  const std::string medium(" font_family='monospace' size='large'><b>");
9  const std::string end("</b></span>");
10
11 VictoryWindow::VictoryWindow(Gtk::Window& window, MenuView& main_menu) :
12     window(window), main_menu(main_menu), was_closed(true) {
13     Glib::RefPtr<Gtk::Builder> builder = Gtk::Builder::create_from_file(
14         GLADE_PATH + "victory_window.glade");
15
16     builder->get_widget("Menu", this->my_window);
17
18     this->my_window->set_title(CLIENT_WINDOW_NAME);
19     this->my_window->set_icon_from_file(ICON_PATH);
20
21     builder->get_widget("victory_msg", victory_msg);
22     this->victory_msg->set_use_markup(true);
23     builder->get_widget("winner", winner);
24     this->winner->set_use_markup(true);
25
26     builder->get_widget("image", this->image);
27
28     builder->get_widget("Return_menu", this->return_menu);
29     builder->get_widget("quit", this->quit);
30
31     ButtonBuilder::buildButton(this->quit);
32     ButtonBuilder::buildButton(this->return_menu);
33
34     this->return_menu->signal_clicked().connect(
35         sigc::mem_fun(*this, &VictoryWindow::returnMenuButtonPressed));
36
37     this->quit->signal_clicked().connect(
38         sigc::mem_fun(*this, &VictoryWindow::quitButtonPressed));
39
40     this->my_window->signal_delete_event().connect(
41         sigc::mem_fun(*this, &VictoryWindow::on_delete_event));
42 }
43
44 VictoryWindow::~VictoryWindow() {}
45
46 bool VictoryWindow::on_delete_event(GdkEventAny* any_event) {
47     gtk_widget_destroy((GtkWidget*) this->my_window->gobj());
48     if (this->was_closed) {
49         // Si se apreto el botÃ³n salir o el botÃ³n de cerrar
50         this->window.close();
51     }
52     return true;
53 }
54
55 void VictoryWindow::returnMenuButtonPressed() {
56     this->was_closed = false;
57     this->my_window->close();
58     this->window.remove();
59     this->main_menu.addMenu();
60 }
61
62 void VictoryWindow::quitButtonPressed() {
63     this->my_window->close();
64 }
65
66 void VictoryWindow::setWinner(const std::string& winner, bool i_win) {

```

jun 25, 18 19:28

## 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 25, 18 19:28

## 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 12, 18 14:03

## WeaponButton.cpp

Page 1/1

```

1  #include "WeaponButton.h"
2  #include <string>
3  #include "Player.h"
4  #include "Path.h"
5
6  WeaponButton::WeaponButton(const std::string& weapon_name, unsigned int ammo,
7                             Player& player) :
8      weapon_name(weapon_name), player(player) {
9      this->setLabel(ammo);
10     std::string path = WEAPONS_PATH;
11     path += weapon_name + ".png";
12     this->image.set(path);
13     this->button.set_image(this->image);
14     this->button.set_always_show_image(true);
15     this->button.signal_clicked().connect(
16         sigc::mem_fun(*this, &WeaponButton::onClickedButton));
17 }
18
19 WeaponButton::~WeaponButton() {}
20
21 void WeaponButton::onClickedButton() {
22     this->player.changeWeapon(weapon_name);
23 }
24
25 Gtk::Widget& WeaponButton::getButton() {
26     return this->button;
27 }
28
29 void WeaponButton::setLabel(unsigned int ammo) {
30     std::string label = "Ammo:\n ";
31     if (!ammo) {
32         label += "0";
33         button.set_sensitive(false);
34     } else if (ammo > 100) {
35         label += "âM-^HM-^^";
36     } else {
37         label += std::to_string(ammo);
38     }
39     this->button.set_label(label);
40 }
41

```

jun 12, 18 14:03

## WeaponButton.h

Page 1/1

```

1  #ifndef __CLIENTWEAPONBUTTON_H__
2  #define __CLIENTWEAPONBUTTON_H__
3
4  #include <gtkmm/togglebutton.h>
5  #include <gtkmm/image.h>
6  #include <string>
7
8  class Player;
9
10 /* Clase que se encarga de mostrar el boton de un arma
11  * junto con la informacion correspondiente a esa arma */
12 class WeaponButton {
13 private:
14     std::string weapon_name;
15     Player& player;
16     Gtk::Button button;
17     Gtk::Image image;
18
19 public:
20     /* Constructor */
21     WeaponButton(const std::string& weapon_name, unsigned int ammo,
22                 Player& player);
23
24     /* Destructor */
25     ~WeaponButton();
26
27     /* Devuelve el wiget del boton */
28     Gtk::Widget& getButton();
29
30     /* Setea el label del boton */
31     void setLabel(unsigned int ammo);
32
33     /* Handler del boton al ser clickeado */
34     void onClickedButton();
35 };
36
37
38 #endif

```

jun 12, 18 14:03

## WeaponView.cpp

Page 1/1

```

1  #include "WeaponView.h"
2  #include <glibmm/main.h>
3  #include <string>
4  #include <utility>
5  #include "Player.h"
6  #include "WeaponList.h"
7  #include "WeaponButton.h"
8
9  WeaponView::WeaponView(WeaponList& weapons, Player& player) :
10     weapons(weapons), player(player) {}
11
12  WeaponView::~WeaponView() {}
13
14  void WeaponView::update() {
15     WeaponList::iterator iter;
16     int row = 1, column = 1;
17     for (iter = this->weapons.begin(); iter != this->weapons.end(); iter++) {
18         std::unique_ptr<WeaponButton> p(
19             new WeaponButton(iter->second->getName(),
20                             iter->second->getAmmo(), this->player));
21         this->buttons.insert(
22             std::pair<std::string, std::unique_ptr<WeaponButton>>(
23                 iter->second->getName(), std::move(p)));
24         this->window.attach(
25             this->buttons.at(iter->second->getName())->getButton(), column,
26             row, 1, 1);
27         row++;
28     }
29 }
30
31 Gtk::Grid& WeaponView::getWindow() {
32     return this->window;
33 }
34
35 void WeaponView::updateAmmo(const Weapon& weapon) {
36     this->buttons[weapon.getName()]>setLabel(weapon.getAmmo());
37 }

```

jun 12, 18 14:03

## WeaponView.h

Page 1/1

```

1  #ifndef __CLIENTWEAPONVIEW_H__
2  #define __CLIENTWEAPONVIEW_H__
3
4  #include <gtkmm/grid.h>
5  #include <unordered_map>
6  #include <memory>
7  #include <string>
8
9  class Player;
10
11  class WeaponList;
12
13  class WeaponButton;
14
15  class Weapon;
16
17  /* Clase que se encarga de mostrar los datos de las armas del juego
18   * y de almacenar todos los botones de las armas */
19  class WeaponView {
20  private:
21     WeaponList& weapons;
22     Gtk::Grid window;
23     Player& player;
24     std::unordered_map<std::string, std::unique_ptr<WeaponButton>> buttons;
25
26  public:
27     /* Constructor */
28     WeaponView(WeaponList& weapons, Player& player);
29
30     /* Destructor */
31     ~WeaponView();
32
33     /* Actualiza la informacion de todos los botones */
34     void update();
35
36     /* Actualiza la informacion de la municion del arma especifica */
37     void updateAmmo(const Weapon& weapon);
38
39     /* Devuelve el contenedor de la vista */
40     Gtk::Grid& getWindow();
41 };
42
43
44 #endif

```



jun 12, 18 14:03

## WindView.cpp

Page 1/1

```

1  #include "WindView.h"
2  #include <string>
3  #include "Path.h"
4
5  WindView::WindView() : container(false, 7) {
6      this->container.pack_start(this->velocity, Gtk::PACK_SHRINK);
7      this->container.pack_start(this->direction, Gtk::PACK_SHRINK);
8      this->velocity.set_use_markup(true);
9  }
10
11 WindView::~WindView() {}
12
13 void WindView::update(float wind) {
14     wind *= 10;
15     std::string message = "<span><b><u>Viento</u></b>\n\n";
16     std::string direction = "right";
17     if (wind == 0) {
18         direction = "no";
19     } else if (wind < 0) {
20         wind *= -1;
21         direction = "left";
22     }
23     std::string velocity = std::to_string(wind);
24     message += velocity.substr(0, 4) + "</span>";
25     this->velocity.set_markup(message);
26     this->direction.set(IMAGES_PATH + "arrow_" + direction + ".png");
27 }
28
29 Gtk::VBox& WindView::getWindow() {
30     return this->container;
31 }

```

jun 12, 18 14:03

## WindView.h

Page 1/1

```

1  #ifndef __WINDVIEW_H__
2  #define __WINDVIEW_H__
3
4  #include <gtkmm/hvbox.h>
5  #include <gtkmm/label.h>
6  #include <gtkmm/image.h>
7
8
9  /* Clase que se encarga de mostrar y actualizar
10   * la informacion del viento */
11 class WindView {
12 private:
13     Gtk::VBox container;
14     Gtk::Label velocity;
15     Gtk::Image direction;
16
17 public:
18     /* Constructor */
19     WindView();
20
21     /* Destructor */
22     ~WindView();
23
24     /* Actualiza la vista del viento */
25     void update(float wind);
26
27     /* Devuelve el contenedor del viento */
28     Gtk::VBox& getWindow();
29 };
30
31 #endif
32

```

jun 25, 18 19:28

## WorldView.cpp

Page 1/2

```

1  #include "WorldView.h"
2  #include <gtkmm/adjustment.h>
3  #include <glibmm/main.h>
4  #include <giomm/memoryinputstream.h>
5  #include "ViewPositionTransformer.h"
6  #include "Player.h"
7  #include "Math.h"
8  #include "Path.h"
9  #include "ObjectSizes.h"
10
11 WorldView::WorldView() {
12     this->container.add_overlay(this->background);
13     this->world.set_size(map_width, map_height);
14     this->window.add_events(Gdk::BUTTON_PRESS_MASK);
15     this->window.add(this->world);
16     this->container.add_overlay(this->window);
17
18     this->water.show(this->world);
19     this->window.get_hadjustment()->set_value(map_width / 2);
20     this->window.get_vadjustment()->set_value(map_height);
21 }
22
23 WorldView::~WorldView() {}
24
25 void WorldView::moveElement(Viewable& element, const Position& position) {
26     float width = element.getWidth();
27     float height = element.getHeight();
28     Gtk::Widget& widget = element.getWidget();
29     Position newPosition = ViewPositionTransformer(
30         this->world).transformToScreenAndMove(position, width, height);
31     this->world.move(widget, newPosition.getX(), newPosition.getY());
32     if (element.hasFocus()) {
33         this->setFocus(widget);
34     }
35 }
36
37 void WorldView::moveScope(Gtk::Widget& scope, Gtk::Widget& worm, int angle) {
38     float pos_x = this->world.child_property_x(worm).get_value();
39     float pos_y = this->world.child_property_y(worm).get_value();
40     pos_x += 50 * Math::cosDegrees(angle);
41     pos_y -= 50 * Math::sinDegrees(angle);
42     // Para que quede referenciado a la mitad de la imagen
43     pos_x -= worm.get_width() / 2;
44     this->world.move(scope, pos_x, pos_y);
45 }
46
47 void WorldView::removeElement(Gtk::Widget& element) {
48     this->world.remove(element);
49 }
50
51 void WorldView::addElement(Viewable& element, const Position& position) {
52     float width = element.getWidth();
53     float height = element.getHeight();
54     Gtk::Widget& widget = element.getWidget();
55     Position newPosition = ViewPositionTransformer(
56         this->world).transformToScreenAndMove(position, width, height);
57     this->world.put(widget, newPosition.getX(), newPosition.getY());
58     widget.show_all();
59     if (element.hasFocus()) {
60         this->setFocus(widget);
61     }
62 }
63
64 Gtk::ScrolledWindow& WorldView::getWindow() {
65     return this->window;
66 }

```

jun 25, 18 19:28

## WorldView.cpp

Page 2/2

```

67
68 Gtk::Layout& WorldView::getLayout() {
69     return this->world;
70 }
71
72 void WorldView::setFocus(Gtk::Widget& element) {
73     this->window.get_hadjustment()->set_value(element.get_allocation().get_x() -
74         this->window.get_hadjustment()->get_page_size() / 2);
75     this->window.get_vadjustment()->set_value(element.get_allocation().get_y() -
76         this->window.get_vadjustment()->get_page_size() / 2);
77 }
78
79 void WorldView::setBackgroundImage(const Buffer& image) {
80     sigc::slot<bool> my_slot = sigc::bind(
81         sigc::mem_fun(*this, &WorldView::setBackgroundImageCallBack),
82         image);
83     Glib::signal_idle().connect(my_slot);
84 }
85
86 bool WorldView::setBackgroundImageCallBack(Buffer image) {
87     auto screen = this->container.get_screen();
88     size_t screen_width = screen->get_width();
89     size_t screen_height = screen->get_height();
90     auto pixbuf = Gio::MemoryInputStream::create();
91     pixbuf->add_data(image.get_pointer(), image.get_max_size());
92     auto aux = Gdk::Pixbuf::create_from_stream(pixbuf);
93     size_t img_width = aux->get_width();
94     size_t img_height = aux->get_height();
95     for (size_t x = 0; x < screen_width; x += img_width) {
96         for (size_t y = 0; y < screen_height; y += img_height) {
97             Gtk::Image background_image(aux);
98             background_image.show();
99             this->background.put(background_image, x, y);
100             this->background_images.push_back(std::move(background_image));
101         }
102     }
103     return false;
104 }
105
106 Gtk::Container& WorldView::getContainer() {
107     return this->container;
108 }

```

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

jun 25, 18 19:28	WorldView.h	Page 2/2
67		
68		
69	<b>#endif</b>	

jun 25, 18 19:28

## BulletView.cpp

Page 1/1

```

1  #include "BulletView.h"
2  #include <string>
3  #include "ObjectSizes.h"
4
5  BulletView::BulletView(WorldView& worldView, std::string weapon, Position pos) :
6      Viewable(worldView), weapon_name(std::move(weapon)) {
7      std::string path(BULLETS_PATH);
8      path += this->weapon_name;
9      path += ".png";
10     this->image.set(path);
11     this->addToWorld(pos);
12 }
13
14 BulletView::~BulletView() {}
15
16 BulletView::BulletView(BulletView&& other) :
17     Viewable(std::move(other)), image(std::move(other.image)),
18     weapon_name(std::move(other.weapon_name)) {}
19
20 void BulletView::updateData(const Position& new_pos) {
21     this->move(new_pos);
22 }
23
24 Gtk::Widget& BulletView::getWidget() {
25     return this->image;
26 }
27
28 float BulletView::getWidth() const {
29     return weapon_size;
30 }
31
32 float BulletView::getHeight() const {
33     return weapon_size;
34 }
35
36 std::string BulletView::getName() {
37     return this->weapon_name;
38 }

```

jun 25, 18 19:28

## BulletView.h

Page 1/1

```

1  #ifndef __CLIENTBULLETVIEW_H__
2  #define __CLIENTBULLETVIEW_H__
3
4  #include <gtkmm/widget.h>
5  #include <gtkmm/image.h>
6  #include <string>
7  #include "Viewable.h"
8
9  /* Clase que se encarga de controlar la vista de las balas */
10 class BulletView : public Viewable {
11 private:
12     Gtk::Image image;
13     std::string weapon_name;
14
15 public:
16     /* Constructor */
17     BulletView(WorldView& worldView, std::string weapon, Position pos);
18
19     /* Destructor */
20     ~BulletView();
21
22     /* Constructor por movimient */
23     BulletView(BulletView&& other);
24
25     /* Actualiza la posicion de la bala en la vista */
26     void updateData(const Position& new_pos);
27
28     /* Devuelve el contenedor de la bala */
29     Gtk::Widget& getWidget() override;
30
31     /* Devuelve el ancho de la bala */
32     float getWidth() const override;
33
34     /* Devuelve el alto de la bala */
35     float getHeight() const override;
36
37     /* Devuelve el nombre del arma de la bala */
38     std::string getName();
39 };
40
41
42 #endif

```

jun 25, 18 19:28

**GirderView.cpp**

Page 1/1

```

1  #include "GirderView.h"
2  #include <string>
3  #include "GirderSize.h"
4
5  GirderView::GirderView(WorldView& worldView, size_t size, Position pos,
6                        int rotation) :
7      Viewable(worldView), size(size), rotation(rotation) {
8      std::string path(GIRDER_PATH);
9      path += std::to_string(size);
10     path += "_";
11     path += std::to_string(rotation);
12     path += ".png";
13     this->image.set(path);
14     this->addToWorld(pos);
15 }
16
17 GirderView::~GirderView() {}
18
19 GirderView::GirderView(GirderView&& other) : Viewable(std::move(other)),
20     image(std::move(other.image)), size(other.size),
21     rotation(other.rotation) {}
22
23 Gtk::Widget& GirderView::getWidget() {
24     return this->image;
25 }
26
27 float GirderView::getWidth() const {
28     return GirderSize::getGirderWidthMeters(this->size, this->rotation);
29 }
30
31 float GirderView::getHeight() const {
32     return GirderSize::getGirderHeightMeters(this->size, this->rotation);
33 }

```

jun 25, 18 19:28

**GirderView.h**

Page 1/1

```

1  #ifndef __GIRDERVIEW_H__
2  #define __GIRDERVIEW_H__
3
4  #include <gtkmm/widget.h>
5  #include <gtkmm/image.h>
6  #include <string>
7  #include "Viewable.h"
8
9  /* Clase que se encarga de controlar la vista de las vigas */
10 class GirderView : public Viewable {
11 private:
12     Gtk::Image image;
13     int size;
14     int rotation;
15
16 public:
17     /* Constructor */
18     GirderView(WorldView& worldView, size_t size, Position pos, int rotation);
19
20     /* Destructor */
21     ~GirderView();
22
23     /* Constructor por movimiento */
24     GirderView(GirderView&& other);
25
26     /* Devuelve el contenedor de la viga */
27     Gtk::Widget& getWidget() override;
28
29     /* Devuelve el ancho de la viga */
30     float getWidth() const override;
31
32     /* Devuelve el alto de la viga */
33     float getHeight() const override;
34 };
35
36
37 #endif

```

jun 25, 18 19:28

## Scope.cpp

Page 1/1

```

1  #include "Scope.h"
2  #include "Path.h"
3  #include "WorldView.h"
4  #include "WeaponNames.h"
5  #include "ObjectSizes.h"
6
7  Scope::Scope(WorldView& world) : Viewable(world) {
8      this->scope.set(SCOPE_IMAGE);
9      this->angle = DEFAULT_ANGLE;
10     this->addToWorld(Position(0, 0));
11 }
12
13 Scope::~Scope() {}
14
15 void Scope::update(int angle, WormView& worm) {
16     this->angle = angle;
17     char dir = worm.getDir();
18     if (dir == DIR_LEFT)
19         angle = 180 - angle;
20     this->worldView.moveScope(this->scope, worm.getWidget(), angle);
21     this->scope.show();
22     worm.updateScope(this->angle);
23 }
24
25 void Scope::update(WormView& worm) {
26     this->update(this->angle, worm);
27 }
28
29 void Scope::hide() {
30     if (this->scope.is_visible()) {
31         this->scope.hide();
32     }
33 }
34
35 Gtk::Widget& Scope::getWidget() {
36     return this->scope;
37 }
38
39 float Scope::getWidth() const {
40     return scope_size;
41 }
42
43 float Scope::getHeight() const {
44     return scope_size;
45 }

```

jun 25, 18 19:28

## Scope.h

Page 1/1

```

1  #ifndef __SCOPE_H__
2  #define __SCOPE_H__
3
4  #include <gtkmm/image.h>
5  #include "Viewable.h"
6  #include "WormView.h"
7
8  /* Clase que se encarga de controlar la imagen
9   * de la mira del arma */
10 class Scope : public Viewable {
11 private:
12     Gtk::Image scope;
13     int angle;
14
15 public:
16     /* Constructor */
17     explicit Scope(WorldView& world);
18
19     /* Destructor */
20     ~Scope();
21
22     /* Actualiza la posicion del scope */
23     void update(int angle, WormView& worm);
24
25     /* Actualiza la posicion del scope */
26     void update(WormView& worm);
27
28     /* Esconde el scope */
29     void hide();
30
31     /* Devuelve el contenedor del scope */
32     Gtk::Widget& getWidget() override;
33
34     /* Devuelve el ancho del scope */
35     float getWidth() const override;
36
37     /* Devuelve el alto del scope */
38     float getHeight() const override;
39 };
40
41 #endif

```

jun 25, 18 19:28

## Viewable.cpp

Page 1/1

```

1  #include "Viewable.h"
2  #include "WorldView.h"
3
4
5  Viewable::Viewable(WorldView& worldView) : has_focus(false),
6                                             worldView(worldView) {}
7
8  Viewable::~Viewable() {}
9
10 Viewable::Viewable(Viewable&& other) : has_focus(other.has_focus),
11                                       worldView(other.worldView) {}
12
13 void Viewable::move(const Position& pos) {
14     this->worldView.moveElement(*this, pos);
15 }
16
17 void Viewable::removeFromWorld() {
18     this->worldView.removeElement(this->getWidget());
19 }
20
21 void Viewable::addToWorld(const Position& pos) {
22     this->worldView.addElement(*this, pos);
23 }
24
25 void Viewable::setFocus(bool focus) {
26     this->has_focus = focus;
27 }
28
29 bool Viewable::hasFocus() const {
30     return this->has_focus;
31 }

```

jun 25, 18 19:28

## Viewable.h

Page 1/1

```

1  #ifndef __VIEWABLE_H__
2  #define __VIEWABLE_H__
3
4  #include <gtkmm/widget.h>
5  #include "Position.h"
6  #include "Path.h"
7
8  class WorldView;
9
10 /* Clase que se encarga de controlar los objetos visuales */
11 class Viewable {
12 private:
13     bool has_focus;
14
15 protected:
16     WorldView& worldView;
17
18     /* Agrega al objeto visual a la vista */
19     void addToWorld(const Position& pos);
20
21     /* Mueve al objeto visual a la posicion especificada */
22     void move(const Position& pos);
23
24 public:
25     /* Constructor */
26     explicit Viewable(WorldView& worldView);
27
28     /* Destructor */
29     virtual ~Viewable();
30
31     /* Constructor por movimiento */
32     Viewable(Viewable&& other);
33
34     /* Devuelve el contenedor del objeto visual */
35     virtual Gtk::Widget& getWidget() = 0;
36
37     /* Remueve al objeto visual de la vista */
38     void removeFromWorld();
39
40     /* Establece si al objeto visual se le puede hacer focus o no */
41     void setFocus(bool focus);
42
43     /* Devuelve true si el objeto visual es focuseable */
44     bool hasFocus() const;
45
46     /* Devuelve el ancho del viewable */
47     virtual float getWidth() const = 0;
48
49     /* Devuelve el alto del viewable */
50     virtual float getHeight() const = 0;
51 };
52
53 #endif

```

jun 12, 18 14:03

## WormLifeView.cpp

Page 1/1

```

1  #include "WormLifeView.h"
2  #include <string>
3
4  const std::string begining("<span color='white'><b>");
5  const std::string ending("</b></span>");
6
7  WormLifeView::WormLifeView(int life, const std::string& color) : color(color) {
8      this->label.set_use_markup(true);
9      this->updateLife(life);
10 }
11
12 WormLifeView::~WormLifeView() {}
13
14 WormLifeView::WormLifeView(WormLifeView&& other) :
15     label(std::move(other.label)), color(std::move(other.color)) {}
16
17 void WormLifeView::updateLife(int life) {
18     this->label.override_background_color(Gdk::RGBA(this->color));
19     this->label.set_markup(begining + std::to_string(life) + ending);
20 }
21
22 Gtk::Widget& WormLifeView::getWidget() {
23     return this->label;
24 }

```

jun 12, 18 14:03

## WormLifeView.h

Page 1/1

```

1  #ifndef __WORMLIFEVIEW_H__
2  #define __WORMLIFEVIEW_H__
3
4  #include <gtkmm/label.h>
5  #include <string>
6
7  /* Clase que se encarga de controlar el label de la vida
8   * del worm */
9  class WormLifeView {
10 private:
11     Gtk::Label label;
12     std::string color;
13
14 public:
15     /* Constructor */
16     WormLifeView(int life, const std::string& color);
17
18     /* Destructor */
19     ~WormLifeView();
20
21     /* Constructor por movimiento */
22     WormLifeView(WormLifeView&& other);
23
24     /* Actualiza el label de vida del worm */
25     void updateLife(int life);
26
27     /* Devuelve el contenedor de la vida */
28     Gtk::Widget& getWidget();
29 };
30
31
32 #endif

```



jun 26, 18 13:37

## WormView.cpp

Page 1/2

```

1  #include "WormView.h"
2  #include <string>
3  #include <glibmm/main.h>
4  #include "ObjectSizes.h"
5  #include "WeaponNames.h"
6  #include "GamePlayers.h"
7
8  WormView::WormView(WorldView& worldView, int life, char dir, Position pos,
9                      int player_id) :
10      Viewable(worldView), player_id(player_id), life(life), is_moving(false),
11      last_position(Position(-1, -1)), label(life, colors[player_id]),
12      walkingAnimation(&this->image),
13      weaponAnimation(DEFAULT_WEAPON, &this->image) {
14      this->worm.attach(this->label.getWidget(), 0, 0, 1, 1);
15      this->worm.attach(this->image, 0, 1, 1, 1);
16      this->walkingAnimation.setStaticImage(DIR_RIGHT);
17      this->addToWorld(pos);
18  }
19
20  WormView::~WormView() {}
21
22  WormView::WormView(WormView&& other) : Viewable(std::move(other)),
23      player_id(other.player_id), life(other.life),
24      is_moving(other.is_moving), last_position(other.last_position),
25      label(std::move(other.label)), image(std::move(other.image)),
26      worm(std::move(other.worm)),
27      walkingAnimation(std::move(other.walkingAnimation)),
28      weaponAnimation(std::move(other.weaponAnimation)) {
29      this->weaponAnimation.updateWormImage(&this->image);
30      this->walkingAnimation.updateWormImage(&this->image);
31  }
32
33  void WormView::updateData(int new_life, char new_dir, const Position& new_pos,
34                           bool colliding, bool is_current_worm, bool has_shot) {
35      if (new_life != this->life) {
36          this->label.updateLife(new_life);
37      }
38      this->life = new_life;
39      this->is_moving = !(this->last_position == new_pos);
40      this->last_position = new_pos;
41      this->setNewImage(new_dir, colliding, is_current_worm, has_shot);
42      this->move(new_pos);
43  }
44
45  void WormView::updateScope(int angle) {
46      this->weaponAnimation.stopWeaponShootAnimation();
47      this->weaponAnimation.changeAngle(angle, this->getDir());
48  }
49
50  void WormView::changeWeapon(const std::string& weapon) {
51      this->weaponAnimation.changeWeapon(weapon, this->getDir());
52  }
53
54  void WormView::setNewImage(char dir, bool colliding, bool is_current_worm,
55                           bool has_shot) {
56      //this->weaponAnimation.stopWeaponShootAnimation();
57      this->walkingAnimation.setStaticImage(dir);
58      if (is_current_worm) {
59          if (!this->is_moving && !has_shot && colliding) {
60              this->weaponAnimation.setWeaponImage(dir);
61          } else if (colliding) {
62              this->walkingAnimation.setMovementImage(dir);
63          }
64      }
65  }
66

```

jun 26, 18 13:37

## WormView.cpp

Page 2/2

```

67  Gtk::Widget& WormView::getWidget() {
68      return this->worm;
69  }
70
71  float WormView::getWidth() const {
72      return worm_size;
73  }
74
75  float WormView::getHeight() const {
76      return worm_size + 0.5;
77  }
78
79  Gtk::Image& WormView::getImage() {
80      return this->image;
81  }
82
83  int WormView::getLife() const {
84      return this->life;
85  }
86
87  char WormView::getDir() const {
88      return this->walkingAnimation.getDir();
89  }
90
91  int WormView::getPlayerId() const {
92      return this->player_id;
93  }
94
95  bool WormView::isMoving() const {
96      return this->is_moving;
97  }
98
99  void WormView::setVictory() {
100      this->image.set(VICTORY_ANIMATION);
101  }
102
103  void WormView::weaponShoot(const std::string& weapon) {
104      this->weaponAnimation.weaponShootAnimation(weapon, this->getDir());
105  }
106
107  void WormView::resetFocus() {
108      this->is_moving = false;
109      this->setFocus(false);
110      this->walkingAnimation.setStaticImage(this->getDir());
111  }

```

jun 25, 18 19:28	WormView.h	Page 1/2
1	<b>#ifndef</b> __WORMVIEW_H__	
2	<b>#define</b> __WORMVIEW_H__	
3		
4	<b>#include</b> <gtkmm/widget.h>	
5	<b>#include</b> <gtkmm/image.h>	
6	<b>#include</b> <gtkmm/grid.h>	
7	<b>#include</b> <gdkmm/pixbuf.h>	
8	<b>#include</b> <vector>	
9	<b>#include</b> <string>	
10	<b>#include</b> "Viewable.h"	
11	<b>#include</b> "WormLifeView.h"	
12	<b>#include</b> "WalkingAnimation.h"	
13	<b>#include</b> "WeaponAnimation.h"	
14		
15	<b>#define</b> DIR_RIGHT 1	
16	<b>#define</b> DIR_LEFT -1	
17		
18	<i>/* Clase que se encarga de controlar la vista de los worms */</i>	
19	class WormView : public Viewable {	
20	private:	
21	int player_id;	
22	int life;	
23	bool is_moving;	
24	Position last_position;	
25	WormLifeView label;	
26	Gtk::Image image;	
27	Gtk::Grid worm;	
28	WalkingAnimation walkingAnimation;	
29	WeaponAnimation weaponAnimation;	
30		
31	<i>/* Actualiza la imagen del worm a la correspondiente segun las</i>	
32	<i>* condiciones en las que se encuentra este */</i>	
33	void	
34	setNewImage(char dir, bool colliding, bool is_current_worm, bool has_shot);	
35		
36	public:	
37	<i>/* Constructor */</i>	
38	WormView(WorldView& worldView, int life, char dir, Position pos,	
39	int player_id);	
40		
41	<i>/* Destructor */</i>	
42	~WormView();	
43		
44	<i>/* Constructor por movimiento */</i>	
45	WormView(WormView&& other);	
46		
47	<i>/* Actualiza la posicion y vida del worm */</i>	
48	void updateData(int new_life, char new_dir, <b>const</b> Position& new_pos,	
49	bool colliding, bool is_current_worm, bool has_shot);	
50		
51		
52	<i>/* Actualiza la imagen del arma con el angulo actual */</i>	
53	void updateScope(int angle);	
54		
55	<i>/* Actualiza el arma del worm y cambia la imagen */</i>	
56	void changeWeapon( <b>const</b> std::string& weapon);	
57		
58	<i>/* Devuelve la direccion del worm */</i>	
59	char getDir() <b>const</b> ;	
60		
61	<i>/* Elimina la imagen del arma del worm */</i>	
62	void removeWeaponImage();	
63		
64	<i>/* Devuelve la vida del worm */</i>	
65	int getLife() <b>const</b> ;	
66		

jun 25, 18 19:28	WormView.h	Page 2/2
67	<i>/* Devuelve el id del player que controla al worm */</i>	
68	int getPlayerId() <b>const</b> ;	
69		
70	<i>/* Devuelve el contenedor donde se encuentra la vista del worm */</i>	
71	Gtk::Widget& getWidget() override;	
72		
73	<i>/* Devuelve el ancho del worm */</i>	
74	float getWidth() <b>const</b> override;	
75		
76	<i>/* Devuelve el alto del worm */</i>	
77	float getHeight() <b>const</b> override;	
78		
79	<i>/* Devuelve la imagen que contiene al worm */</i>	
80	Gtk::Image& getImage();	
81		
82	<i>/* Cambia la imagen del worm por la animacion del worm</i>	
83	<i>* festejando la victoria */</i>	
84	void setVictory();	
85		
86	<i>/* Devuelve true si el gusano se esta moviendo */</i>	
87	bool isMoving() <b>const</b> ;	
88		
89	<i>/* Realiza la animacion del disparo del arma */</i>	
90	void weaponShoot( <b>const</b> std::string& weapon);	
91		
92	<i>/* Resetea el focus del gusano */</i>	
93	void resetFocus();	
94	};	
95		
96		
97	<b>#endif</b>	

jun 26, 18 13:50

## ViewsList.cpp

Page 1/3

```

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

```

jun 26, 18 13:50

## ViewsList.cpp

Page 2/3

```

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

```

jun 26, 18 13:50

## ViewsList.cpp

Page 3/3

```

131     Glib::signal_idle().connect(my_slot);
132 }
133
134 void ViewsList::setCurrentWorm(int id) {
135     this->removeWormFocus();
136     std::unordered_map<int, WormView>::iterator it;
137     for (it = this->worms.begin(); it != this->worms.end(); ++it) {
138         it->second.resetFocus();
139     }
140     this->current_worm_id = id;
141     this->worm_focused = id;
142     this->weapon_focused = NO_FOCUS;
143     WormView& worm = this->worms.at(id);
144     this->world.setFocus(worm.getWidget());
145     worm.setFocus(true);
146 }
147
148 void ViewsList::removeWormFocus() {
149     std::unordered_map<int, WormView>::iterator it = this->worms.find(this->worm
_focused);
150     if (it != this->worms.end()) {
151         it->second.resetFocus();
152     }
153     this->worm_focused = NO_FOCUS;
154 }
155
156 void ViewsList::checkMovingWorms() {
157     if (this->weapon_focused != WEAPON_SHOT_AND_EXPLODED) {
158         return;
159     }
160
161     std::unordered_map<int, WormView>::iterator it = this->worms.find(this->worm
_focused);
162     if (it == this->worms.end() || !it->second.isMoving()) {
163         this->removeWormFocus();
164         for (auto it2 = this->worms.begin(); it2 != this->worms.end(); ++it2) {
165             if (it2->second.isMoving()) {
166                 this->worm_focused = it2->first;
167                 it2->second.setFocus(true);
168                 this->world.setFocus(it2->second.getWidget());
169                 return;
170             }
171         }
172     }
173 }
174
175 void ViewsList::setVictory() {
176     if (this->worms.empty()) {
177         return;
178     }
179     std::unordered_map<int, WormView>::iterator iter;
180     for (iter = this->worms.begin(); iter != this->worms.end(); iter++) {
181         this->musicPlayer.playVictory();
182         iter->second.setVictory();
183         this->world.setFocus(iter->second.getWidget());
184     }
185 }
186
187 void ViewsList::shoot(const std::string& weapon) {
188     this->worms.at(this->current_worm_id).weaponShoot(weapon);
189     if (this->weapon_focused == NO_FOCUS) {
190         this->weapon_focused = WEAPON_SHOT_AND_EXPLODED;
191     }
192 }

```

jun 12, 18 14:03

## 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 12, 18 14:03

## 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 contrario
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 26, 18 13:59

## Table of Content

Page 1/2

Table of Contents			
1	ClientProtocol.cpp..	sheets 1 to 2 ( 2) pages	1- 3 135 lines
2	ClientProtocol.h....	sheets 2 to 3 ( 2) pages	4- 5 80 lines
3	DataReceiver.cpp....	sheets 3 to 4 ( 2) pages	6- 7 102 lines
4	DataReceiver.h.....	sheets 4 to 4 ( 1) pages	8- 8 35 lines
5	main.cpp.....	sheets 5 to 5 ( 1) pages	9- 9 21 lines
6	ButtonBuilder.cpp...	sheets 5 to 5 ( 1) pages	10- 10 12 lines
7	ButtonBuilder.h.....	sheets 6 to 6 ( 1) pages	11- 11 14 lines
8	CreateGameMenu.cpp..	sheets 6 to 6 ( 1) pages	12- 12 54 lines
9	CreateGameMenu.h....	sheets 7 to 7 ( 1) pages	13- 13 29 lines
10	GameMenu.cpp.....	sheets 7 to 8 ( 2) pages	14- 15 80 lines
11	GameMenuField.cpp...	sheets 8 to 8 ( 1) pages	16- 16 31 lines
12	GameMenuField.h.....	sheets 9 to 9 ( 1) pages	17- 17 35 lines
13	GameMenu.h.....	sheets 9 to 9 ( 1) pages	18- 18 33 lines
14	JoinGameMenu.cpp....	sheets 10 to 10 ( 1) pages	19- 19 37 lines
15	JoinGameMenu.h.....	sheets 10 to 10 ( 1) pages	20- 20 24 lines
16	Menu.cpp.....	sheets 11 to 11 ( 1) pages	21- 21 53 lines
17	Menu.h.....	sheets 11 to 11 ( 1) pages	22- 22 43 lines
18	MenuView.cpp.....	sheets 12 to 12 ( 1) pages	23- 23 28 lines
19	MenuView.h.....	sheets 12 to 12 ( 1) pages	24- 24 36 lines
20	SelectableListMenu.cpp	sheets 13 to 13 ( 1) pages	25- 26 73 lines
21	SelectableListMenu.h	sheets 14 to 14 ( 1) pages	27- 27 53 lines
22	ServerFatalError.cpp	sheets 14 to 14 ( 1) pages	28- 28 15 lines
23	ServerFatalError.h..	sheets 15 to 15 ( 1) pages	29- 29 18 lines
24	ServerMenu.cpp.....	sheets 15 to 15 ( 1) pages	30- 30 59 lines
25	ServerMenu.h.....	sheets 16 to 16 ( 1) pages	31- 31 38 lines
26	WaitingRoom.cpp.....	sheets 16 to 16 ( 1) pages	32- 32 25 lines
27	WaitingRoom.h.....	sheets 17 to 17 ( 1) pages	33- 33 28 lines
28	Handlers.cpp.....	sheets 17 to 18 ( 2) pages	34- 36 171 lines
29	Handlers.h.....	sheets 19 to 19 ( 1) pages	37- 38 76 lines
30	Player.cpp.....	sheets 20 to 20 ( 1) pages	39- 40 115 lines
31	Player.h.....	sheets 21 to 21 ( 1) pages	41- 42 87 lines
32	Turn.cpp.....	sheets 22 to 22 ( 1) pages	43- 43 48 lines
33	Turn.h.....	sheets 22 to 22 ( 1) pages	44- 44 44 lines
34	DistanceWeapon.cpp..	sheets 23 to 23 ( 1) pages	45- 45 19 lines
35	DistanceWeapon.h....	sheets 23 to 23 ( 1) pages	46- 46 25 lines
36	MeleeWeapon.cpp.....	sheets 24 to 24 ( 1) pages	47- 47 13 lines
37	MeleeWeapon.h.....	sheets 24 to 24 ( 1) pages	48- 48 21 lines
38	WeaponPowerAccum.cpp	sheets 25 to 25 ( 1) pages	49- 49 37 lines
39	WeaponPowerAccum.h..	sheets 25 to 25 ( 1) pages	50- 50 36 lines
40	AirAttack.cpp.....	sheets 26 to 26 ( 1) pages	51- 51 11 lines
41	AirAttack.h.....	sheets 26 to 26 ( 1) pages	52- 52 20 lines
42	Banana.cpp.....	sheets 27 to 27 ( 1) pages	53- 53 10 lines
43	Banana.h.....	sheets 27 to 27 ( 1) pages	54- 54 20 lines
44	Bat.cpp.....	sheets 28 to 28 ( 1) pages	55- 55 10 lines
45	Bat.h.....	sheets 28 to 28 ( 1) pages	56- 56 20 lines
46	Bazooka.cpp.....	sheets 29 to 29 ( 1) pages	57- 57 10 lines
47	Bazooka.h.....	sheets 29 to 29 ( 1) pages	58- 58 20 lines
48	Dynamite.cpp.....	sheets 30 to 30 ( 1) pages	59- 59 10 lines
49	Dynamite.h.....	sheets 30 to 30 ( 1) pages	60- 60 20 lines
50	GreenGrenade.cpp....	sheets 31 to 31 ( 1) pages	61- 61 12 lines
51	GreenGrenade.h.....	sheets 31 to 31 ( 1) pages	62- 62 20 lines
52	HolyGrenade.cpp.....	sheets 32 to 32 ( 1) pages	63- 63 12 lines
53	HolyGrenade.h.....	sheets 32 to 32 ( 1) pages	64- 64 20 lines
54	Mortar.cpp.....	sheets 33 to 33 ( 1) pages	65- 65 10 lines
55	Mortar.h.....	sheets 33 to 33 ( 1) pages	66- 66 20 lines
56	RedGrenade.cpp.....	sheets 34 to 34 ( 1) pages	67- 67 11 lines
57	RedGrenade.h.....	sheets 34 to 34 ( 1) pages	68- 68 20 lines
58	Teleportation.cpp...	sheets 35 to 35 ( 1) pages	69- 69 12 lines
59	Teleportation.h.....	sheets 35 to 35 ( 1) pages	70- 70 20 lines
60	SelfDirectedWeapon.cpp	sheets 36 to 36 ( 1) pages	71- 71 16 lines
61	SelfDirectedWeapon.h	sheets 36 to 36 ( 1) pages	72- 72 24 lines
62	Weapon.cpp.....	sheets 37 to 37 ( 1) pages	73- 73 57 lines
63	Weapon.h.....	sheets 37 to 37 ( 1) pages	74- 74 54 lines
64	WeaponList.cpp.....	sheets 38 to 38 ( 1) pages	75- 75 32 lines
65	WeaponList.h.....	sheets 38 to 38 ( 1) pages	76- 76 43 lines

jun 26, 18 13:59

## Table of Content

Page 2/2

67	66	<i>WeaponsFactory.cpp</i> ..	sheets	39 to	39 ( 1)	pages	77- 77	41 lines
68	67	<i>WeaponsFactory.h</i> ....	sheets	39 to	39 ( 1)	pages	78- 78	26 lines
69	68	<i>MusicPath.h</i> .....	sheets	40 to	40 ( 1)	pages	79- 79	27 lines
70	69	<i>MusicPlayer.cpp</i> .....	sheets	40 to	41 ( 2)	pages	80- 82	141 lines
71	70	<i>MusicPlayerException.cpp</i>	sheets	42 to	42 ( 1)	pages	83- 83	13 lines
72	71	<i>MusicPlayerException.h</i>	sheets	42 to	42 ( 1)	pages	84- 84	23 lines
73	72	<i>MusicPlayer.h</i> .....	sheets	43 to	43 ( 1)	pages	85- 86	70 lines
74	73	<i>ExplosionView.cpp</i> ...	sheets	44 to	44 ( 1)	pages	87- 87	47 lines
75	74	<i>ExplosionView.h</i> ....	sheets	44 to	44 ( 1)	pages	88- 88	39 lines
76	75	<i>ExplosionViewList.cpp</i>	sheets	45 to	45 ( 1)	pages	89- 89	25 lines
77	76	<i>ExplosionViewList.h</i> .	sheets	45 to	45 ( 1)	pages	90- 90	30 lines
78	77	<i>WalkingAnimation.cpp</i>	sheets	46 to	46 ( 1)	pages	91- 91	48 lines
79	78	<i>WalkingAnimation.h</i> ..	sheets	46 to	46 ( 1)	pages	92- 92	45 lines
80	79	<i>WeaponAnimation.cpp</i> .	sheets	47 to	47 ( 1)	pages	93- 94	107 lines
81	80	<i>WeaponAnimation.h</i> ...	sheets	48 to	48 ( 1)	pages	95- 95	63 lines
82	81	<i>PlayerLifeLabel.cpp</i> .	sheets	48 to	48 ( 1)	pages	96- 96	47 lines
83	82	<i>PlayerLifeLabel.h</i> ...	sheets	49 to	49 ( 1)	pages	97- 97	44 lines
84	83	<i>PlayersList.cpp</i> .....	sheets	49 to	49 ( 1)	pages	98- 98	43 lines
85	84	<i>PlayersList.h</i> .....	sheets	50 to	50 ( 1)	pages	99- 99	46 lines
86	85	<i>ScreenView.cpp</i> .....	sheets	50 to	51 ( 2)	pages	100-101	72 lines
87	86	<i>ScreenView.h</i> .....	sheets	51 to	52 ( 2)	pages	102-103	72 lines
88	87	<i>TurnLabel.cpp</i> .....	sheets	52 to	52 ( 1)	pages	104-104	42 lines
89	88	<i>TurnLabel.h</i> .....	sheets	53 to	53 ( 1)	pages	105-105	46 lines
90	89	<i>VictoryWindow.cpp</i> ...	sheets	53 to	54 ( 2)	pages	106-107	87 lines
91	90	<i>VictoryWindow.h</i> ....	sheets	54 to	54 ( 1)	pages	108-108	50 lines
92	91	<i>WeaponButton.cpp</i> ....	sheets	55 to	55 ( 1)	pages	109-109	42 lines
93	92	<i>WeaponButton.h</i> .....	sheets	55 to	55 ( 1)	pages	110-110	39 lines
94	93	<i>WeaponView.cpp</i> .....	sheets	56 to	56 ( 1)	pages	111-111	38 lines
95	94	<i>WeaponView.h</i> .....	sheets	56 to	56 ( 1)	pages	112-112	45 lines
96	95	<i>WindView.cpp</i> .....	sheets	57 to	57 ( 1)	pages	113-113	32 lines
97	96	<i>WindView.h</i> .....	sheets	57 to	57 ( 1)	pages	114-114	33 lines
98	97	<i>WorldView.cpp</i> .....	sheets	58 to	58 ( 1)	pages	115-116	109 lines
99	98	<i>WorldView.h</i> .....	sheets	59 to	59 ( 1)	pages	117-118	70 lines
100	99	<i>BulletView.cpp</i> .....	sheets	60 to	60 ( 1)	pages	119-119	39 lines
101	100	<i>BulletView.h</i> .....	sheets	60 to	60 ( 1)	pages	120-120	43 lines
102	101	<i>GirderView.cpp</i> .....	sheets	61 to	61 ( 1)	pages	121-121	34 lines
103	102	<i>GirderView.h</i> .....	sheets	61 to	61 ( 1)	pages	122-122	38 lines
104	103	<i>Scope.cpp</i> .....	sheets	62 to	62 ( 1)	pages	123-123	46 lines
105	104	<i>Scope.h</i> .....	sheets	62 to	62 ( 1)	pages	124-124	42 lines
106	105	<i>Viewable.cpp</i> .....	sheets	63 to	63 ( 1)	pages	125-125	32 lines
107	106	<i>Viewable.h</i> .....	sheets	63 to	63 ( 1)	pages	126-126	54 lines
108	107	<i>WormLifeView.cpp</i> ....	sheets	64 to	64 ( 1)	pages	127-127	25 lines
109	108	<i>WormLifeView.h</i> .....	sheets	64 to	64 ( 1)	pages	128-128	33 lines
110	109	<i>WormView.cpp</i> .....	sheets	65 to	65 ( 1)	pages	129-130	112 lines
111	110	<i>WormView.h</i> .....	sheets	66 to	66 ( 1)	pages	131-132	98 lines
112	111	<i>ViewsList.cpp</i> .....	sheets	67 to	68 ( 2)	pages	133-135	193 lines
113	112	<i>ViewsList.h</i> .....	sheets	68 to	69 ( 2)	pages	136-137	95 lines