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

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

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

ClientProtocol::~ClientProtocol(){}

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

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

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

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

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

    this->sendBuffer(buffer);
}

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

    this->sendIntBuffer(buffer, angle);

    this->sendBuffer(buffer);
}

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



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

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

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

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

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

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

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

        players_list.addPlayer(id, name);
    }
}

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

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

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

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

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

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

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

Jun 07, 18 20:36 **ClientProtocol.cpp** Page 3/3

```
130         }
131 }
```

Jun 07, 18 20:36 **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
12 class Player;
13 class WeaponList;
14
15 /* Clase que se encarga de enviar y recibir mensajes del socket
16  * con un formato determinado */
17 class ClientProtocol: public Protocol {
18     private:
19         Gtk::Window& window;
20
21     public:
22         /* Constructor */
23         ClientProtocol(Socket&& socket, Gtk::Window& window);
24
25         /* Constructor por movimiento */
26         ClientProtocol(ClientProtocol&& other);
27
28         /* Destructor */
29         ~ClientProtocol();
30
31         /* Envia un mensaje que indica una accion de movimiento */
32         void sendMoveAction(char action);
33
34         /* Envia un mensaje que indica una accion de cambio de arma
35          * con el nombre del arma */
36         void sendChangeWeapon(const std::string &weapon);
37
38         /* Envia un mensaje de accion de disparo, con el angulo, la potencia
39          * y el tiempo de explosion */
40         void sendWeaponShoot(int32_t angle, int32_t power, int32_t time);
41
42         /* Envia un mensaje de accion de disparo teledirigido con
43          * la posicion del disparo */
44         void sendWeaponSelfDirectedShoot(const Position &pos);
45
46         /* Envia un mesaje que indica el cambio del angulo del scope */
47         void updateScope(int angle);
48
49         /* Envia un mensaje de finalizacion de juego */
50         void sendEndGame();
51
52         /* Recibe el comienzo del juego */
53         void receiveStartGame();
54
55         /* Recibe y setea la imagen de fondo */
56         void receiveBackgroundImage(WorldView& world);
57
58         /* Recibe los datos del turno */
59         void receiveTurnData(Turn& turn);
60
61         /* Recibe los jugadores de la partida junto con su
62          * id y su nombre */
63         void receivePlayers(PlayersList& players_list);
64
65         /* Recibe la vigas presentes en el mapa junto con su tamaÃ±o,
66          * su posicion y su rotacion */
```

Jun 07, 18 20:36 **ClientProtocol.h** Page 2/2

```
67          void receiveGirders(ViewsList& viewsList);
68
69          /* Recibe las armas presentes en el juego junto con
70           * su municion */
71          void receiveWeaponsAmmo(WeaponList& weapon_list);
72
73          /* Envia el contenido del buffer */
74          void sendBuffer(Buffer &buffer) override;
75  };
76
77  #endif
```

Jun 07, 18 19:58 **DataReceiver.cpp** Page 1/2

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

Jun 07, 18 19:58 **DataReceiver.cpp** Page 2/2

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

Jun 06, 18 20:08 **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 enviados por el servidor */
10  class DataReceiver: public Thread{
11      private:
12          Player& player;
13          ClientProtocol& protocol;
14
15          /* Recibe los datos de la configuracion inicial */
16          void initialConfig();
17
18          /* Analiza los datos recibidos */
19          bool analizeReceivedData(Buffer buffer);
20
21      public:
22          /* Constructor */
23          DataReceiver(Player& player);
24
25          /* Destructor */
26          ~DataReceiver();
27
28          /* Comienza a recibir mensajes del protocolo */
29          void run() override;
30  };
31
32
33  #endif
```

May 30, 18 20:03 **main.cpp** Page 1/1

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

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

    window.set_title(CLIENT_WINDOW_NAME);

    window.set_icon_from_file(ICON_PATH);

    ServerMenu server_menu(window);

    app->run(window);

    return 0;
}
```

Jun 01, 18 13:12 **ButtonBuilder.cpp** Page 1/1

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

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

Jun 03, 18 12:56 **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 visualizaciÃ³n del label del boton */
9          static void buildButton(Gtk::Button* button);
10 };
11
12
13 #endif //WORMS_BUTTONBUILDER_H
```

Jun 07, 18 11:17 **CreateGameMenu.cpp** Page 1/1

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

Jun 07, 18 11:17 **CreateGameMenu.h** Page 1/1

```
#ifndef __CREATEGAMEMENU__
#define __CREATEGAMEMENU__

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

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

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

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

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

#endif
```

Jun 07, 18 21:10 **GameMenu.cpp** Page 1/2

```
#include "GameMenu.h"
#include "Path.h"
#include "CreateGameMenu.h"
#include "JoinGameMenu.h"
#include "ButtonBuilder.h"

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

GameMenu::GameMenu(Gtk::Window& window, ClientProtocol& protocol):
    MenuView(window, *this, protocol, PATH){

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

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

    this->addMenu();

    Gtk::Button *create_game, *join_game;

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

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

    create_game->signal_clicked().connect(sigc::mem_fun(*this, &GameMenu::create
ButtonPressed));
    join_game->signal_clicked().connect(sigc::mem_fun(*this, &GameMenu::joinButt
onPressed));
}

GameMenu::~GameMenu(){}

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

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

bool GameMenu::selectAction(char action){
    std::string name(this->player_name->get_text());
    if (name.empty()){
        this->error->set_label("Debe ingresar su nombre");
        return false;
```

Jun 07, 18 21:10 **GameMenu.cpp** Page 2/2

```
63      }
64      try{
65          this->protocol.sendChar(action);
66          this->protocol.sendString(name);
67          this->window.remove();
68          return true;
69      } catch (const SocketException& e){
70          this->showFatalError();
71          return false;
72      }
73  }
```

Jun 07, 18 11:17 **GameMenuField.cpp** Page 1/1

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

May 28, 18 18:21 **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         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
34 #endif
```

Jun 07, 18 11:17 **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 07, 18 11:17 **JoinGameMenu.cpp** Page 1/1

```cpp
#include "JoinGameMenu.h"
#include "Path.h"
#include "WaitingLabel.h"

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

JoinGameMenu::JoinGameMenu(Gtk::Window& window, MenuView& first_menu, ClientProtocol& protocol, std::string&& name, int quantity):
    SelectableListMenu(window, first_menu, protocol, std::move(name), PATH){

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

    this->configure(quantity);

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

    this->addMenu();
}

JoinGameMenu::~JoinGameMenu(){}


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

Jun 07, 18 11:17 **JoinGameMenu.h** Page 1/1

```cpp
#ifndef __JOINGAMEMENU__
#define __JOINGAMEMENU__

#include "SelectableListMenu.h"

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

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

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

#endif
```

Jun 07, 18 11:34 **Menu.cpp** Page 1/1

```cpp
#include "Menu.h"
#include "Path.h"
#include "ButtonBuilder.h"

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

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

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

    ButtonBuilder::buildButton(this->quit);

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

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

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

Menu::~Menu() {}

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

Jun 07, 18 11:17 **Menu.h** Page 1/1

```cpp
#ifndef WORMS_MENU_H
#define WORMS_MENU_H

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

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

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

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

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


#endif //WORMS_MENU_H
```

Jun 07, 18 11:17 **MenuView.cpp** Page 1/1

```cpp
#include "MenuView.h"
#include "ServerFatalError.h"

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

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

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

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

void MenuView::showError(const std::string& error){
    this->error->set_label(error);
}

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

Jun 07, 18 20:10 **MenuView.h** Page 1/1

```cpp
#ifndef __MENUVIEW_H__
#define __MENUVIEW_H__

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

class MenuView : public Menu {
    private:
        /* Muestra un mensaje de error */
        void showError(const std::string& error);

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

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

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

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

        /* Destructor */
        virtual ~MenuView();

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

#endif
```

```cpp
#include "SelectableListMenu.h"
#include "ButtonBuilder.h"

SelectableListMenu::SelectableListMenu(Gtk::Window& window, MenuView& first_menu, ClientProtocol& protocol, std::string&& name, const std::string& path):
    MenuView(window, first_menu, protocol, path), player_name(std::move(name)) {

    this->builder->get_widget("turn_back", this->turn_back);
    ButtonBuilder::buildButton(this->turn_back);
    this->turn_back->signal_clicked().connect(sigc::mem_fun(*this, &SelectableListMenu::turnBackButtonPressed));
}

SelectableListMenu::~SelectableListMenu() {}

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

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

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

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

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

void SelectableListMenu::waitToPlayers() {
    this->window.remove();
    this->window.add(this->waiting_label.getWidget());
    this->window.show_all();
    sigc::slot<bool> my_slot = sigc::mem_fun(*this, &SelectableListMenu::createPlayer);
    Glib::signal_idle().connect(my_slot);
```



```cpp
}
```

Jun 07, 18 11:17 **SelectableListMenu.h** Page 1/1

```
#ifndef __SELECTABLELISTMENU_H__
#define __SELECTABLELISTMENU_H__

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

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

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

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

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

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

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

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

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

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

#endif
```

Jun 07, 18 11:17 **ServerFatalError.cpp** Page 1/1

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

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

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

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

Jun 01, 18 13:44 **ServerFatalError.h** Page 1/1

```h
#ifndef __SERVERFATALERROR_H__
#define __SERVERFATALERROR_H__

#include <gtkmm/window.h>

class ServerFatalError{
    public:
        ServerFatalError(Gtk::Window& window);

        ~ServerFatalError();
};

#endif
```

Jun 07, 18 11:17 **ServerMenu.cpp** Page 1/1

```cpp
#include "ServerMenu.h"
#include "Path.h"
#include "ButtonBuilder.h"

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

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

    ButtonBuilder::buildButton(this->connect);

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

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

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

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

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

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

    this->connectToServer(host, service);
}

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

Jun 07, 18 11:17 **ServerMenu.h** Page 1/1

```
#ifndef __SERVERMENU__
#define __SERVERMENU__

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

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

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

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

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

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

#endif
```

May 27, 18 22:46 **WaitingLabel.cpp** Page 1/1

```
#include "WaitingLabel.h"

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

WaitingLabel::WaitingLabel(){
    this->label.set_use_markup(true);
    this->label.set_markup(begining + "Esperando jugadores..." + ending);
    this->label.show();
}

WaitingLabel::~WaitingLabel(){}

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

Jun 03, 18 12:56 **WaitingLabel.h** Page 1/1

```
#ifndef __WAITINGLABEL_H__
#define __WAITINGLABEL_H__

#include <gtkmm/label.h>

/* Label de que indica la espera a otros jugadores */
class WaitingLabel{
    private:
        Gtk::Label label;

    public:
        /* Constructor */
        WaitingLabel();

        /* Destructor */
        ~WaitingLabel();

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


#endif
```

Jun 09, 18 14:13 **Handlers.cpp** Page 1/3

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

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

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

Handlers::~Handlers() {}

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

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

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

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

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

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

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

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

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

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

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



```
129 }
130
131 bool Handlers::onButtonPressEvent(GdkEventButton *event) {
132     if (!this->enabled){
133         return true;
134     }
135
136     if (!this->weapons.getCurrentWeapon().isSelfDirected()) {
137         return true;
138     }
139     if (!this->weapons.getCurrentWeapon().hasAmmo()) {
140         this->player.getMusicPlayer().playNoAmmo();
141         return true;
142     }
143     if (this->has_shoot) {
144         return true;
145     }
146     if ((event->type == GDK_BUTTON_PRESS) && (event->button == 1)) {
147         float x = event->x;
148         float y = event->y;
149         x += this->world.getWindow().get_hadjustment()->get_value();
150         y += this->world.getWindow().get_vadjustment()->get_value();
151         Position position(x, y);
152         Position newPosition = ViewPositionTransformer(this->world.getLayout()).
transformToPosition(position);
153         this->has_shoot = true;
154         this->player.shoot(newPosition);
155     }
156     return true;
157 }
158
159 int Handlers::getCurrentAngle() const{
160     return this->current_angle;
161 }
162
163 void Handlers::stop(){
164     this->scroll_handler.stop();
165 }
```

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



```
66
67         /* Detiene los handlers */
68         void stop();
69 };
70
71 #endif
```

Jun 09, 18 14:13 **Player.cpp** Page 1/2

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

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

    this->musicPlayer.playMusic();
    this->data_receiver.start();
}

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

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

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

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

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

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

Jun 09, 18 14:13 **Player.cpp** Page 2/2

```cpp

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

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

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

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

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

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

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

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

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

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



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

Jun 07, 18 21:03 **Turn.cpp** Page 1/1

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

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

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

Turn::~Turn() {}

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

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

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

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

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

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

Jun 07, 18 14:42 **Turn.h** Page 1/1

```cpp
#ifndef __CLIENTTURN_H__
#define __CLIENTTURN_H__

#include "TurnLabel.h"

class Player;

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

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

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

        /* Destructor */
        ~Turn();


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

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

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

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

#endif
```

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

```cpp
#include "DistanceWeapon.h"

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

DistanceWeapon::~DistanceWeapon() {}

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

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

```

May 27, 18 21:56 **DistanceWeapon.h** Page 1/1

```cpp
#ifndef __CLIENTDISTANCEWEAPON_H__
#define __CLIENTDISTANCEWEAPON_H__

#include "Weapon.h"

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

        /* Destructor */
        ~DistanceWeapon();

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


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

#endif
```

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

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

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

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

```

May 27, 18 21:56 **MeleeWeapon.h** Page 1/1

```cpp
#ifndef __CLIENTMELEEWEAPON_H__
#define __CLIENTMELEEWEAPON_H__

#include "Weapon.h"

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

        /* Destructor */
        ~MeleeWeapon() {}

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

#endif
```

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

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

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

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

WeaponPowerAccum::~WeaponPowerAccum() {}

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

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

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

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

May 31, 18 12:08 **WeaponPowerAccum.h** Page 1/1

```cpp
#ifndef __CLIENTTIMER_H__
#define __CLIENTTIMER_H__

#include <glibmm/main.h>

class Handlers;

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

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

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

        /* Destructor */
        ~WeaponPowerAccum();

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

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

#endif
```

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

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

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

AirAttack::~AirAttack() {}

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

```

May 27, 18 21:56 **AirAttack.h** Page 1/1

```cpp
#ifndef __CLIENTAIRATTACK_H__
#define __CLIENTAIRATTACK_H__

#include "SelfDirectedWeapon.h"

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

        /* Destructor */
        ~AirAttack();

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

#endif
```

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

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

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

Banana::~Banana() {}

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

```

May 27, 18 21:56 **Banana.h** Page 1/1

```cpp
#ifndef __CLIENTBANANA_H__
#define __CLIENTBANANA_H__

#include "DistanceWeapon.h"

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

        /* Destructor */
        ~Banana();

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

#endif
```

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

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

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

Bat::~Bat() {}

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

```

May 31, 18 12:08 **Bat.h** Page 1/1

```cpp
#ifndef __CLIENTBAT_H__
#define __CLIENTBAT_H__

#include "MeleeWeapon.h"

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

        /* Destructor */
        ~Bat();

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

#endif
```

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

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

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

Bazooka::~Bazooka(){}

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

```

May 27, 18 21:56 **Bazooka.h** Page 1/1

```cpp
#ifndef __CLIENTBAZOOKA_H__
#define __CLIENTBAZOOKA_H__

#include "DistanceWeapon.h"

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

        /* Destructor */
        ~Bazooka();

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

#endif
```

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

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

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

Dynamite::~Dynamite() {}

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

```

May 27, 18 21:56 **Dynamite.h** Page 1/1

```cpp
#ifndef __CLIENTDYNAMITE_H__
#define __CLIENTDYNAMITE_H__

#include "MeleeWeapon.h"

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

        /* Destructor */
        ~Dynamite();

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

#endif
```

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

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

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

GreenGrenade::~GreenGrenade(){}

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

```

May 31, 18 12:08 **GreenGrenade.h** Page 1/1

```cpp
#ifndef __CLIENTGREENGRENADE_H__
#define __CLIENTGREENGRENADE_H__

#include "DistanceWeapon.h"

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

        /* Destructor */
        ~GreenGrenade();

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

#endif
```

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

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

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

HolyGrenade::~HolyGrenade(){}

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

```

May 31, 18 12:08 **HolyGrenade.h** Page 1/1

```cpp
#ifndef __CLIENTHOLYGRENADE_H__
#define __CLIENTHOLYGRENADE_H__

#include "DistanceWeapon.h"

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

        /* Destructor */
        ~HolyGrenade();

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

#endif
```

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

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

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

Mortar::~Mortar() {}

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

```

May 27, 18 21:56 **Mortar.h** Page 1/1

```cpp
#ifndef __CLIENTMORTAR_H__
#define __CLIENTMORTAR_H__

#include "DistanceWeapon.h"

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

        /* Destructor */
        ~Mortar();

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

#endif
```

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

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

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

RedGrenade::~RedGrenade() {}

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

```

May 31, 18 12:08 **RedGrenade.h** Page 1/1

```cpp
#ifndef __CLIENTREDGRENADE_H__
#define __CLIENTREDGRENADE_H__

#include "DistanceWeapon.h"

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

        /* Destructor */
        ~RedGrenade();

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

#endif
```

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

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

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

Teleportation::~Teleportation(){}

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

```

May 31, 18 12:08 **Teleportation.h** Page 1/1

```cpp
#ifndef __CLIENTTELEPORTATION_H__
#define __CLIENTTELEPORTATION_H__

#include "SelfDirectedWeapon.h"

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

        /* Destructor */
        ~Teleportation();

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

#endif
```

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

```cpp
#include "SelfDirectedWeapon.h"

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

SelfDirectedWeapon::~SelfDirectedWeapon() {}

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

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

```

May 31, 18 12:08 **SelfDirectedWeapon.h** Page 1/1

```cpp
#ifndef __SELFDIRECTEDWEAPON_H__
#define __SELFDIRECTEDWEAPON_H__

#include "Weapon.h"

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

        /* Destructor */
        ~SelfDirectedWeapon();

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

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

#endif
```

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

```cpp
#include "Weapon.h"

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

Weapon::~Weapon() {}

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

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

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

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

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

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

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

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

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

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

```

May 27, 18 21:56 **Weapon.h** Page 1/1

```cpp
#ifndef __CLIENTWEAPON_H__
#define __CLIENTWEAPON_H__

#include <string>

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

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

        /* Destructor */
        ~Weapon();

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

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


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

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

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

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

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

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

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

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

May 27, 18 21:56 **WeaponList.cpp** Page 1/1

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

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

WeaponList::~WeaponList() {}

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

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

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

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

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

```

May 31, 18 12:08 **WeaponList.h** Page 1/1

```cpp
#ifndef __CLIENTWEAPONLIST_H__
#define __CLIENTWEAPONLIST_H__

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

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

    public:
        /* Constructor */
        WeaponList();

        /* Destructor */
        ~WeaponList();


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

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

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

        typedef WeaponsList::iterator iterator;
        typedef WeaponsList::const_iterator const_iterator;
        iterator begin();
        iterator end();
};


#endif
```

May 27, 18 21:56 **WeaponsFactory.cpp** Page 1/1

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

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


WeaponsFactory::WeaponsFactory() {}

WeaponsFactory::~WeaponsFactory() {}

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

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

```cpp
#ifndef __CLIENTWEAPONSFACTORY_H__
#define __CLIENTWEAPONSFACTORY_H__

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

typedef std::unique_ptr<Weapon> weapon_ptr;

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

        /* Destructor */
        ~WeaponsFactory();


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


#endif
```

Jun 03, 18 12:56 **MusicPath.h** Page 1/1

```
#ifndef WORMS_MUSICPATH_H
#define WORMS_MUSICPATH_H

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

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

#endif //WORMS_MUSICPATH_H
```

Jun 03, 18 12:56 **MusicPlayer.cpp** Page 1/3

```
#include "MusicPlayer.h"
#include "MusicPlayerException.h"
#include "WeaponNames.h"
#include "Protocol.h"
#include "MusicPath.h"

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

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

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

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

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

void MusicPlayer::addEffect(const std::string& audio) {
    int channel;
    Mix_Chunk* effect = NULL;
    effect = Mix_LoadWAV(audio.c_str());
    if (effect == NULL) {
        return;
    }
    if ((channel = Mix_PlayChannel(-1, effect, 0)) == -1) {
        Mix_FreeChunk(effect);
```

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



```
133     }
134 }
135
136 void MusicPlayer::playSelectWeaponSound() {
137     this->addEffect(SELECT_WEAPON_SOUND);
138 }
```

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

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

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

MusicPlayerException::~MusicPlayerException(){}

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

May 22, 18 11:29 **MusicPlayerException.h** Page 1/1

```cpp
#ifndef __MUSICPLAYEREXCEPTION_H__
#define __MUSICPLAYEREXCEPTION_H__

#include <exception>
#include <string>

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

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

        //Destruye la excepcion
        virtual ~MusicPlayerException();

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

#endif
```

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



```
67
68
69 #endif
```

May 31, 18 12:08 **ExplosionView.cpp** Page 1/1

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

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

ExplosionView::~ExplosionView() {}

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

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

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

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

May 31, 18 12:08 **ExplosionView.h** Page 1/1

```cpp
#ifndef __CLIENTEXPLOSIONVIEW_H__
#define __CLIENTEXPLOSIONVIEW_H__

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

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

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

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

        /* Destructor */
        ~ExplosionView();

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


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

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


#endif
```

May 31, 18 12:08 **ExplosionViewList.cpp** Page 1/1

```cpp
#include "ExplosionViewList.h"

ExplosionViewList::ExplosionViewList() {}

ExplosionViewList::~ExplosionViewList() {}

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

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

May 31, 18 12:08 **ExplosionViewList.h** Page 1/1

```cpp
#ifndef WORMS_EXPLOSIONVIEWLIST_H
#define WORMS_EXPLOSIONVIEWLIST_H

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

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

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

    public:
        /* Constructor */
        ExplosionViewList();

        /* Destructor */
        ~ExplosionViewList();


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

};


#endif //WORMS_EXPLOSIONVIEWLIST_H
```

Jun 09, 18 14:13 **WalkingAnimation.cpp** Page 1/1

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

#define DIR_RIGHT 1
#define DIR_LEFT -1

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

WalkingAnimation::~WalkingAnimation() {}

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

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

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

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

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

Jun 09, 18 14:13 **WalkingAnimation.h** Page 1/1

```cpp
#ifndef WORMS_WALKINGANIMATION_H
#define WORMS_WALKINGANIMATION_H

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

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

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

        /* Destructor */
        ~WalkingAnimation();

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


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

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

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

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


#endif //WORMS_WALKINGANIMATION_H
```

Jun 06, 18 20:08 **WeaponAnimation.cpp** Page 1/2

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

#define DIR_RIGHT 1

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

WeaponAnimation::~WeaponAnimation() {}

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

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

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

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

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

void WeaponAnimation::weaponShootAnimation(const std::string &weapon, char dir) {
    if (weapon != BAT_NAME) {
        return;
    }
    this->scope_image = Gdk::Pixbuf::create_from_file(BAT_HIT_ANIMATION);
```

Jun 06, 18 20:08 **WeaponAnimation.cpp** Page 2/2

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

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

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

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

```
#ifndef WORMS_WEAPONANIMATION_H
#define WORMS_WEAPONANIMATION_H

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

class WormView;

class WeaponAnimation {
    private:
        std::vector<Glib::RefPtr<Gdk::Pixbuf>> scope_vector;
        Glib::RefPtr<Gdk::Pixbuf> scope_image;
        Gtk::Image* worm_image;
        int angle;

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

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

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

        /* Destructor */
        ~WeaponAnimation();

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


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

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

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

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

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


#endif //WORMS_WEAPONANIMATION_H
```

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

```
#include "Scope.h"
#include "Path.h"
#include "WeaponNames.h"

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

Scope::~Scope(){}

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

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


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

Jun 02, 18 18:22 **Scope.h** Page 1/1

```h
#ifndef __SCOPE_H__
#define __SCOPE_H__

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

class Scope{
    private:
        Gtk::Image scope;
        WorldView& world;
        int angle;

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

        /* Destructor */
        ~Scope();

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

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

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

};

#endif
```

May 30, 18 20:03 **PlayerLifeLabel.cpp** Page 1/1

```cpp
#include "PlayerLifeLabel.h"
#include "GamePlayers.h"

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

PlayerLifeLabel::PlayerLifeLabel(): id(0), player_name(""), life(0){
    this->label.set_use_markup(true);
}

PlayerLifeLabel::~PlayerLifeLabel(){}

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

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

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

Gtk::Label& PlayerLifeLabel::getLabel(){
    return this->label;
}

void PlayerLifeLabel::updateLabel(){
    std::string message = begining + colors[this->id] + middle;
    message += std::to_string(this->id) + "- " + this->player_name;
    message += ": " + std::to_string(this->life) + ending;
    this->label.set_markup(message);
}
```

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

```
#ifndef __PLAYERLIFELABEL_H__
#define __PLAYERLIFELABEL_H__

#include <gtkmm/label.h>

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

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

    public:
        /* Constructor */
        PlayerLifeLabel();

        /* Destructor */
        ~PlayerLifeLabel();


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

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

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

        /* Devuelve el label del jugador */
        Gtk::Label& getLabel();
};


#endif
```

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

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

#define SPACING 20

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

PlayersList::~PlayersList(){}

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

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

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

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

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

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

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

```
#ifndef __PLAYERSLIST_H__
#define __PLAYERSLIST_H__

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

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

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

    public:
        /* Constructor */
        PlayersList();

        /* Destructor */
        ~PlayersList();

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

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

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

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

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

#endif
```

Jun 07, 18 20:07 **ScreenView.cpp** Page 1/2

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

#define PADDING 10
#define SPACING 30

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

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

ScreenView::~ScreenView() {}

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

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

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

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

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

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

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

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

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

```

Jun 07, 18 20:07 **ScreenView.cpp** Page 2/2

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

Jun 07, 18 20:06 **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 "MenuView.h"
8  #include "WorldView.h"
9  #include "WeaponView.h"
10 #include "TurnLabel.h"
11 #include "PlayersList.h"
12 #include "WindView.h"
13 #include "VictoryWindow.h"
14
15 /* Clase que se encarga de almacenar los contenedores principales
16  * de la vista y mostrar su contenido */
17 class ScreenView {
18     private:
19         Gtk::VBox screen;
20         Gtk::HBox world_box;
21         Gtk::VBox left_view;
22         Gtk::Window& window;
23
24         WorldView world;
25         WeaponView weapons_view;
26         TurnLabel turn_label;
27         PlayersList players;
28         WindView wind_view;
29
30         VictoryWindow victory_view;
31
32         /* CallBacks */
33         bool showCallBack();
34         bool closeCallBack();
35
36     public:
37         /* Constructor */
38         ScreenView(Gtk::Window& window, MenuView& main_menu, Player& player, Wea
   ponList& weapons);
39
40         /* Destructor */
41         ~ScreenView();
42
43         /* Muestra la pantalla en la ventana */
44         void show();
45
46         /* Cierra la ventana completamente */
47         void close();
48
49         /* Devuelve el WorldView */
50         WorldView& getWorld();
51
52         /* Devuelve el WeaponView */
53         WeaponView& getWeaponsView();
54
55         /* Devuelve el TurnLabel */
56         TurnLabel& getTurnLabel();
57
58         /* Devuelve el Players view */
59         PlayersList& getPlayersView();
60
61         /* Devuelve el wind view */
62         WindView& getWindView();
63
64         /* Muestra una ventana con el ganador */
65         void setWinner(const std::string& winner, bool i_win);
```

Jun 07, 18 20:06 **ScreenView.h** Page 2/2

```
66  };
67
68  #endif
```

Jun 07, 18 19:09 **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 07, 18 19:09 **TurnLabel.h** Page 1/1

```
#ifndef __TURNLABEL_H__
#define __TURNLABEL_H__

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

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

    public:
        /* Constructor */
        TurnLabel();

        /* Destructor */
        ~TurnLabel();


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

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

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

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

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

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


#endif
```

Jun 09, 18 14:13 **VictoryWindow.cpp** Page 1/1

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

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

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

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

    builder->get_widget("victory_msg", victory_msg);

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

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

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

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

VictoryWindow::~VictoryWindow() {}

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

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

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

void VictoryWindow::setWinner(const std::string& winner, bool i_win) {
    std::string winner_message;
    if (winner.empty()){
        winner_message = "Empate";
    } else if (i_win) {
        winner_message = "GANASTE!!!!";
    } else {
        winner_message = "Perdiste. El ganador fue: " + winner;
    }
    this->victory_msg->set_text(winner_message);
    this->my_window->set_modal(true);
    this->my_window->show_all();
}
```

Jun 09, 18 14:13 **VictoryWindow.h** Page 1/1

```
#ifndef WORMS_VICTORYWINDOW_H
#define WORMS_VICTORYWINDOW_H

#include <gtkmm/window.h>
#include <gtkmm/button.h>
#include <gtkmm/label.h>
#include <string>
#include "MenuView.h"

class VictoryWindow {
    private:
        Gtk::Window* my_window;
        Gtk::Window& window;
        Gtk::Button* return_menu;
        Gtk::Button* quit;
        Gtk::Label* victory_msg;
        MenuView& main_menu;
        bool was_closed;

        bool on_delete_event(GdkEventAny* any_event);

        void returnMenuButtonPressed();

        void quitButtonPressed();

    public:
        VictoryWindow(Gtk::Window& window, MenuView& main_menu);

        ~VictoryWindow();

        void setWinner(const std::string& winner, bool i_win);
};


#endif //WORMS_VICTORYWINDOW_H
```

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

```
#include "WeaponButton.h"
#include "Player.h"
#include "Path.h"

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

WeaponButton::~WeaponButton() {}

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

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

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

```

Jun 05, 18 15:28 **WeaponButton.h** Page 1/1

```h
#ifndef __CLIENTWEAPONBUTTON_H__
#define __CLIENTWEAPONBUTTON_H__

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

class Player;

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

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

        /* Destructor */
        ~WeaponButton();

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

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

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


#endif
```

Jun 03, 18 12:56 **WeaponView.cpp** Page 1/1

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

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

WeaponView::~WeaponView() {}

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

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

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

Jun 03, 18 12:56 **WeaponView.h** Page 1/1

```h
#ifndef __CLIENTWEAPONVIEW_H__
#define __CLIENTWEAPONVIEW_H__

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

class Player;
class WeaponList;
class WeaponButton;
class Weapon;

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

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

        /* Destructor */
        ~WeaponView();


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

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

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

#endif
```

Jun 02, 18 13:59 **WindView.cpp** Page 1/1

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

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

WindView::~WindView(){}

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

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

May 30, 18 22:01 **WindView.h** Page 1/1

```h
#ifndef __WINDVIEW_H__
#define __WINDVIEW_H__

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

class WindView{
    private:
        Gtk::VBox container;
        Gtk::Label velocity;
        Gtk::Image direction;

    public:
        WindView();
        ~WindView();

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

        Gtk::VBox& getWindow();
};

#endif

```

Jun 06, 18 20:39 **WorldView.cpp** Page 1/2

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

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

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

WorldView::~WorldView() {}

void WorldView::moveElement(Gtk::Widget& element, const Position& position, floa
t width, float height, bool focus){
    Position newPosition = ViewPositionTransformer(this->world).transformToScree
nAndMove(position, width, height);
    this->world.move(element, newPosition.getX(), newPosition.getY());
    if (focus){
        this->setFocus(element);
    }
}

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

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

void WorldView::addElement(Gtk::Widget& element, const Position& position, float
 width, float height, bool focus){
    Position newPosition = ViewPositionTransformer(this->world).transformToScree
nAndMove(position, width, height);
    this->world.put(element, newPosition.getX(), newPosition.getY());
    element.show_all();
    if (focus){
        this->setFocus(element);
    }
}

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

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

Jun 06, 18 20:39 **WorldView.cpp** Page 2/2

```
62
63  void WorldView::setFocus(Gtk::Widget& element){
64      this->window.get_hadjustment()->set_value(element.get_allocation().get_x() -
     this->window.get_hadjustment()->get_page_size() / 2);
65      this->window.get_vadjustment()->set_value(element.get_allocation().get_y() -
     this->window.get_vadjustment()->get_page_size() / 2);
66  }
67
68  void WorldView::setBackgroundImage(const Buffer& image){
69      sigc::slot<bool> my_slot = sigc::bind(sigc::mem_fun(*this, &WorldView::setBa
    ckgroundImageCallBack), image);
70      Glib::signal_idle().connect(my_slot);
71  }
72
73  bool WorldView::setBackgroundImageCallBack(Buffer image){
74      auto screen = this->container.get_screen();
75      size_t screen_width = screen->get_width();
76      size_t screen_height = screen->get_height();
77      auto pixbuf = Gio::MemoryInputStream::create();
78      pixbuf->add_data(image.getPointer(), image.getMaxSize());
79      auto aux = Gdk::Pixbuf::create_from_stream (pixbuf);
80      size_t img_width = aux->get_width();
81      size_t img_height = aux->get_height();
82      for (size_t x = 0; x < screen_width; x += img_width) {
83          for (size_t y = 0; y < screen_height; y += img_height) {
84              Gtk::Image background_image(aux);
85              background_image.show();
86              this->background.put(background_image, x, y);
87              this->background_images.push_back(std::move(background_image));
88          }
89      }
90      return false;
91  }
92
93  Gtk::Container& WorldView::getContainer(){
94      return this->container;
95  }
```

Jun 06, 18 20:17 **WorldView.h** Page 1/2

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

Jun 06, 18 20:17 **WorldView.h** Page 2/2

```
65
66  #endif
```

May 31, 18 12:08 **BulletView.cpp** Page 1/1

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

May 31, 18 12:08 **BulletView.h** Page 1/1

```
#ifndef __CLIENTBULLETVIEW_H__
#define __CLIENTBULLETVIEW_H__

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

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

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

        /* Destructor */
        ~BulletView();

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

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

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

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


#endif
```

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

```
#include "GirderView.h"
#include "GirderSize.h"

GirderView::GirderView(WorldView& worldView, size_t size, Position pos, int rotation):
    Viewable(worldView), size(size), rotation(rotation){

    std::string path(GIRDER_PATH);
    path += std::to_string(size);
    path += "_";
    path += std::to_string(rotation);
    path += ".png";
    this->image.set(path);
    float width = GirderSize::getGirderWidthMeters(size, rotation);
    float height = GirderSize::getGirderHeightMeters(size, rotation);
    this->addToWorld(pos, width, height);
}

GirderView::~GirderView(){}

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

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

```

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

```h
#ifndef __GIRDERVIEW_H__
#define __GIRDERVIEW_H__

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

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

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

        /* Destructor */
        ~GirderView();

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

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


#endif
```

May 28, 18 18:21 **Viewable.cpp** Page 1/1

```cpp
#include "Viewable.h"

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

Viewable::~Viewable(){}

void Viewable::move(const Position& pos, float width, float height){
    this->worldView.moveElement(this->getWidget(), pos, width, height, this->has_focus);
}

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

void Viewable::addToWorld(const Position& pos, float width, float height){
    this->worldView.addElement(this->getWidget(), pos, width, height, this->has_focus);
}

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

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

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

May 31, 18 12:08 **Viewable.h** Page 1/1

```h
#ifndef __VIEWABLE_H__
#define __VIEWABLE_H__

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

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

    protected:
        /* Agrega al objeto visual a la vista */
        void addToWorld(const Position& pos, float width, float height);

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

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

        /* Destructor */
        virtual ~Viewable();

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

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

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

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

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

#endif
```

Jun 07, 18 11:38 **WormLifeView.cpp** Page 1/1

```cpp
#include "WormLifeView.h"

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

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

WormLifeView::~WormLifeView(){}

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

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

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

May 27, 18 21:56 **WormLifeView.h** Page 1/1

```
#ifndef __WORMLIFEVIEW_H__
#define __WORMLIFEVIEW_H__

#include <gtkmm/label.h>

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

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

        /* Destructor */
        ~WormLifeView();

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

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

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


#endif
```

Jun 09, 18 14:13 **WormView.cpp** Page 1/2

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

WormView::WormView(WorldView& worldView, int life, char dir, Position pos, int player_id):
    Viewable(worldView), player_id(player_id), life(life), is_moving(false),
    last_position(Position(-1, -1)), label(life, colors[player_id]),
    walkingAnimation(&this->image), weaponAnimation(DEFAULT_WEAPON, &this->image) {
        this->worm.attach(this->label.getWidget(), 0, 0, 1, 1);
        this->worm.attach(this->image, 0, 1, 1, 1);
        this->walkingAnimation.setStaticImage(DIR_RIGHT);
        this->addToWorld(pos, worm_size, worm_size + 0.5);
}

WormView::~WormView(){}

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

void WormView::updateData(int new_life, char new_dir, const Position& new_pos, bool colliding, bool is_current_worm, bool has_shot) {
    if (new_life != this->life){
        this->label.updateLife(new_life);
    }
    this->life = new_life;
    this->is_moving = !(this->last_position == new_pos);
    this->last_position = new_pos;
    this->setNewImage(new_dir, colliding, is_current_worm, has_shot);
    this->move(new_pos, worm_size, worm_size + 0.5);
}

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

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

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

Gtk::Widget& WormView::getWidget(){
```

Jun 09, 18 14:13 **WormView.cpp** Page 2/2

```
61      return this->worm;
62  }
63
64  Gtk::Image& WormView::getImage() {
65      return this->image;
66  }
67
68  int WormView::getLife() const{
69      return this->life;
70  }
71
72  char WormView::getDir() const {
73      return this->walkingAnimation.getDir();
74  }
75
76  int WormView::getPlayerId() const{
77      return this->player_id;
78  }
79
80  bool WormView::isMoving() const{
81      return this->is_moving;
82  }
83
84  void WormView::setVictory() {
85      this->image.set(VICTORY_ANIMATION);
86  }
87
88  void WormView::weaponShoot(const std::string& weapon) {
89      this->weaponAnimation.weaponShootAnimation(weapon, this->getDir());
90  }
91
92  void WormView::resetFocus(){
93      this->is_moving = false;
94      this->setFocus(false);
95      this->walkingAnimation.setStaticImage(this->getDir());
96  }
```

Jun 06, 18 20:08 **WormView.h** Page 1/2

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

Jun 06, 18 20:08 **WormView.h** Page 2/2

```
63          /* Devuelve la direccion del worm */
64          char getDir() const;
65
66          /* Elimina la imagen del arma del worm */
67          void removeWeaponImage();
68
69          /* Devuelve la vida del worm */
70          int getLife() const;
71
72          /* Devuelve el id del player que controla al worm */
73          int getPlayerId() const;
74
75          /* Devuelve el contenedor donde se encuentra la vista del worm */
76          Gtk::Widget& getWidget() override;
77
78          /* Devuelve la imagen que contiene al worm */
79          Gtk::Image& getImage();
80
81          /* Cambia la imagen del worm por la animacion del worm
82           * festejando la victoria */
83          void setVictory();
84
85          /* Devuelve true si el gusano se esta moviendo */
86          bool isMoving() const;
87
88          /* Realiza la animacion del disparo del arma */
89          void weaponShoot(const std::string& weapon);
90
91          /* Resetea el focus del gusano */
92          void resetFocus();
93  };
94
95
96  #endif
```

Jun 07, 18 14:06 **ViewsList.cpp** Page 1/3

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

```
63          it->second.updateData(life, dir, pos, colliding, id == this->current_wor
    m_id, this->weapon_focused != -1);
64          this->checkMovingWorms();
65      }
66  }
67
68  void ViewsList::updateWeaponData(int id, const std::string& weapon_name, float p
    os_x, float pos_y){
69      auto it = this->weapons.find(id);
70      Position pos(pos_x / UNIT_TO_SEND, pos_y / UNIT_TO_SEND);
71      if (it == this->weapons.end()){
72          //Weapon no existe
73          BulletView weapon(this->world, weapon_name, pos);
74          if (this->weapon_focused < 0){
75              weapon.setFocus(true);
76              this->weapon_focused = id;
77              this->removeWormFocus();
78          }
79          this->weapons.insert(std::make_pair(id, std::move(weapon)));
80      } else {
81          //Weapon existe
82          it->second.updateData(pos);
83      }
84  }
85
86  void ViewsList::changeWeapon(const std::string& weapon_name) {
87      auto it = this->worms.find(this->current_worm_id);
88      it->second.changeWeapon(weapon_name);
89      if (WeaponsFactory().createWeapon(weapon_name, 1)->hasScope()) {
90          this->scope.update(it->second);
91      }
92  }
93
94  void ViewsList::updateScope(int angle) {
95      auto it = this->worms.find(this->current_worm_id);
96      if (it == this->worms.end()) {
97          return;
98      }
99      this->scope.update(angle, it->second);
100 }
101
102 void ViewsList::removeScopeVisibility() {
103     this->scope.hide();
104 }
105
106 bool ViewsList::addGirderCallBack(size_t size, Position pos, int rotation){
107     GirderView girder(this->world, size, pos, rotation);
108     this->girders.push_back(std::move(girder));
109     return false;
110 }
111
112 void ViewsList::addGirder(size_t size, float pos_x, float pos_y, int rotation){
113     sigc::slot<bool> my_slot = sigc::bind(sigc::mem_fun(*this, &ViewsList::addGi
    rderCallBack), size, Position(pos_x, pos_y), rotation);
114     Glib::signal_idle().connect(my_slot);
115 }
116
117 void ViewsList::setCurrentWorm(int id){
118     this->removeWormFocus();
119     for (auto it = this->worms.begin(); it != this->worms.end(); ++it){
120         it->second.resetFocus();
121     }
122     this->current_worm_id = id;
123     this->worm_focused = id;
124     this->weapon_focused = -1;
125     WormView& worm = this->worms.at(id);
```



```
126     this->world.setFocus(worm.getWidget());
127     worm.setFocus(true);
128 }
129
130 void ViewsList::removeWormFocus(){
131     auto it = this->worms.find(this->worm_focused);
132     if (it != this->worms.end()){
133         it->second.resetFocus();
134     }
135     this->worm_focused = -1;
136 }
137
138 void ViewsList::checkMovingWorms(){
139     if (this->weapon_focused != -2){
140         return;
141     }
142
143     auto it = this->worms.find(this->worm_focused);
144     if (it == this->worms.end() || !it->second.isMoving()){
145         this->removeWormFocus();
146         for (auto it2 = this->worms.begin(); it2 != this->worms.end(); ++it2){
147             if (it2->second.isMoving()){
148                 this->worm_focused = it2->first;
149                 it2->second.setFocus(true);
150                 this->world.setFocus(it2->second.getWidget());
151                 return;
152             }
153         }
154     }
155
156 }
157
158 void ViewsList::setVictory() {
159     if (this->worms.empty()){
160         return;
161     }
162     for (auto iter = this->worms.begin(); iter != this->worms.end(); iter++) {
163         this->musicPlayer.playVictory();
164         iter->second.setVictory();
165         this->world.setFocus(iter->second.getWidget());
166     }
167 }
168
169 void ViewsList::shoot(const std::string& weapon) {
170     this->worms.at(this->current_worm_id).weaponShoot(weapon);
171 }
```

```
#ifndef __VIEWSLIST_H__
#define __VIEWSLIST_H__

#include <unordered_map>
#include <vector>
#include <string>
#include "WorldView.h"
#include "WormView.h"
#include "BulletView.h"
#include "GirderView.h"
#include "PlayersList.h"
#include "ExplosionView.h"
#include "ExplosionViewList.h"
#include "MusicPlayer.h"
#include "Scope.h"

/* Clase que se encarga de almacenar los objetos visibles */
class ViewsList{
    private:
        WorldView& world;
        Player& player;
        PlayersList& players_list;
        std::unordered_map<int, WormView> worms;
        std::unordered_map<int, BulletView> weapons;
        std::vector<GirderView> girders;
        int current_worm_id;
        int weapon_focused;
        int worm_focused;
        ExplosionViewList animation;
        Scope scope;
        MusicPlayer& musicPlayer;

        /* Elimina el focus sobre el worm */
        void removeWormFocus();

        /* CallBacks */
        bool addGirderCallBack(size_t size, Position pos, int rotation);

    public:
        /* Constructor */
        ViewsList(WorldView& world, Player& player, PlayersList& players_list, M
usicPlayer& musicPlayer);

        /* Destructor */
        ~ViewsList();

        /* Elimina al worm de la vista actualizando la vida del player */
        void removeWorm(int id);

        /* Elimina la vista del arma y la reemplaza por la animacion de la explo
sion */
        void removeWeapon(int id);

        /* Actualiza la posicion y la vida del worm */
        void updateWormData(int id, int player_id, float pos_x, float pos_y, int
 life, char dir, bool colliding);

        /* Actualiza la posicion del arma */
        void updateWeaponData(int id, const std::string& weapon_name, float pos_
x, float pos_y);

        /* CallBack de changeWeapon */
        bool changeWeaponCallBack(const std::string &weapon_name);

        /* Actualiza la vista del worm con el arma nueva */
        void changeWeapon(const std::string &weapon_name);
```



```

        /* Actualiza la posicion del scope */
        void updateScope(int angle);

        /* Esconde la vista del scope */
        void removeScopeVisibility();

        /* Agrega una viga a la vista en la posicion indicada y
         * con la rotacion indicada */
        void addGirder(size_t size, float pos_x, float pos_y, int rotation);

        /* Actualiza el worm actual y hace focus en este */
        void setCurrentWorm(int id);

        /* Actualiza la imagen de los worms ganadores por la animacion
         * de los worms festejando */
        void setVictory();

        /* Chequea si el gusano actual se esta moviendo, caso contario
        le da el focus a otro */
        void checkMovingWorms();

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


#endif
```

Jun 09, 18 18:50 **Table of Content** Page 1/2

Table of Contents

Jun 09, 18 18:50 **Table of Content** Page 2/2