

May 31, 18 12:08

## ExplosionView.cpp

Page 1/1

```

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

```

May 31, 18 12:08

## ExplosionView.h

Page 1/1

```

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

```

May 31, 18 12:08

**ExplosionViewList.cpp**

Page 1/1

```

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

```

May 31, 18 12:08

**ExplosionViewList.h**

Page 1/1

```

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

```

Jun 06, 18 20:08

## WalkingAnimation.cpp

Page 1/1

```

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

```

Jun 06, 18 20:08

## WalkingAnimation.h

Page 1/1

```

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

```

Jun 06, 18 20:08

## WeaponAnimation.cpp

Page 1/2

```

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

```

Jun 06, 18 20:08

## WeaponAnimation.cpp

Page 2/2

```

60     int width = this->scope_image->get_width() / 3;
61     int height = this->scope_image->get_height();
62     int pos_x = width + dir * width;
63     this->scope_vector.clear();
64     for (int i = 0; i < height / WORM_IMAGE_WIDTH; i++) {
65         this->scope_vector.push_back(Gdk::Pixbuf::create_subpixbuf(scope_image,
66 pos_x, i * WORM_IMAGE_WIDTH, width, WORM_IMAGE_WIDTH));
67     }
68     std::vector<Glib::RefPtr<Gdk::Pixbuf>>::iterator iter = this->scope_vector.begin();
69     sigc::slot<bool> my_slot = sigc::bind(sigc::mem_fun(*this, &WeaponAnimation::batHitCallback), iter, width, dir);
70     Glib::signal_timeout().connect(my_slot, 12);
71 }
72
73 void WeaponAnimation::changeAngle(int angle, char dir) {
74     this->angle = angle;
75     this->setWeaponImage(dir);
76 }
77
78 void WeaponAnimation::updateWormImage(Gtk::Image* worm_image) {
79     this->worm_image = worm_image;

```

Jun 06, 18 20:08

## WeaponAnimation.h

Page 1/1

```

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

```

May 30, 18 20:03

## main.cpp

Page 1/1

```

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

```

Jun 01, 18 13:12

## ButtonBuilder.cpp

Page 1/1

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

Jun 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 visualizaci3n del label del boton */
9         static void buildButton(Gtk::Button* button);
10 };
11
12
13 #endif //WORMS_BUTTONBUILDER_H
```

Jun 05, 18 15:19

## CreateGameMenu.cpp

Page 1/1

```

1  #include "CreateGameMenu.h"
2  #include <gtkmm/builder.h>
3  #include <glibmm/main.h>
4  #include "Path.h"
5  #include "GamePlayers.h"
6  #include "ButtonBuilder.h"
7
8  CreateGameMenu::CreateGameMenu(Gtk::Window& window, MenuView& first_menu, Client
Protocol& protocol, std::string&& name, int quantity):
9      SelectableListMenu(window, first_menu, protocol, std::move(name)){
10
11      Glib::RefPtr<Gtk::Builder> builder = Gtk::Builder::create_from_file(GLADE_PA
TH + "client_CreateGameMenu.glade");
12
13      builder->get_widget("error", this->error);
14      builder->get_widget("game_name", this->game_name);
15      builder->get_widget("players_number", this->players_number);
16      builder->get_widget("games", this->games);
17      builder->get_widget("quit_game", this->quit_game);
18
19      this->configure(quantity);
20
21      ButtonBuilder::buildButton(quit_game);
22
23      builder->get_widget("create_game_menu", this->menu);
24
25      this->addMenu();
26      this->quit_game->signal_clicked().connect(sigc::mem_fun(*this, &CreateGameMe
nu::quitButtonPressed));
27  }
28
29  CreateGameMenu::~CreateGameMenu() {}
30
31  void CreateGameMenu::selectButtonPressed(Glib::ustring map_name){
32      std::string name(this->game_name->get_text());
33      if (name.empty()){
34          this->error->set_label("Debe ingresar el nombre de la partida");
35          return;
36      }
37
38      size_t players = this->players_number->get_value_as_int();
39      if (players < min_players || players > max_players){
40          std::string message("El numero de jugadores debe estar entre ");
41          message += std::to_string(min_players) + std::string(" y ") + std::to_str
ing(max_players);
42          this->error->set_label(message);
43          return;
44      }
45
46      try{
47          this->protocol.sendString(map_name);
48          this->protocol.sendString(name);
49          this->protocol.sendLength(players);
50          bool result = this->protocol.receiveChar();
51          if (!result){
52              this->showErrorAndRestart("Ocurrio un error al crear la partida");
53          } else {
54              this->waitToPlayers();
55          }
56      } catch (const SocketException& e){
57          this->showFatalError();
58      }
59  }

```

Jun 01, 18 13:12

## CreateGameMenu.h

Page 1/1

```

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

```

Jun 03, 18 12:56

## GameMenu.cpp

Page 1/2

```

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

```

Jun 03, 18 12:56

## GameMenu.cpp

Page 2/2

```

61      std::string name(this->player_name->get_text());
62      if (name.empty()){
63          this->error->set_label("Debe ingresar su nombre");
64          return false;
65      }
66      try{
67          this->protocol.sendChar(action);
68          this->protocol.sendString(name);
69          this->window.remove();
70          return true;
71      } catch (const SocketException& e){
72          this->showFatalError();
73          return false;
74      }
75  }

```



Jun 02, 18 12:18

## 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->title.override_background_color(Gdk::RGBA("white"));
11     this->container.pack_start(this->title);
12     this->container.pack_end(this->button);
13
14     this->button.set_label("Seleccionar");
15     ButtonBuilder::buildButton(&this->button);
16 }
17
18 GameMenuField::~GameMenuField(){}
19
20 GameMenuField::GameMenuField(GameMenuField&& other): title(std::move(other.title
21 )),
22     button(std::move(other.button)), container(std::move(other.container)){}
23
24 Gtk::Container& GameMenuField::getContainer(){
25     return this->container;
26 }
27
28 Gtk::Button& GameMenuField::getButton(){
29     return this->button;
30 }

```

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     /* Devuelve el contenedor del menu */
26     Gtk::Container& getContainer();
27
28     /* Devuelve el boton del menu */
29     Gtk::Button& getButton();
30 };
31
32
33
34 #endif

```

May 31, 18 13:58

## GameMenu.h

Page 1/1

```

1  #ifndef __GAMEMENU__
2  #define __GAMEMENU__
3
4  #include <gtkmm/button.h>
5  #include <gtkmm/entry.h>
6  #include <gtkmm/window.h>
7  #include <string>
8  #include <memory>
9  #include "ClientProtocol.h"
10 #include "MenuView.h"
11
12 /* Clase que se encarga de controlar el menu del juego */
13 class GameMenu: public MenuView{
14     private:
15         Gtk::Entry* player_name;
16
17         /* Crea el boton de creacion de partida */
18         void createButtonPressed();
19
20         /* Crea el boton de unirse a partida */
21         void joinButtonPressed();
22
23         /* Envia la accion implementada */
24         bool selectAction(char action);
25
26     public:
27         /* Constructor */
28         GameMenu(Gtk::Window& window, ClientProtocol& protocol);
29
30         /* Destructor */
31         ~GameMenu();
32 };
33
34 #endif

```

Jun 03, 18 12:56

## JoinGameMenu.cpp

Page 1/1

```

1  #include "JoinGameMenu.h"
2  #include <gtkmm/builder.h>
3  #include <glibmm/main.h>
4  #include "Path.h"
5  #include "WaitingLabel.h"
6  #include "ButtonBuilder.h"
7
8  JoinGameMenu::JoinGameMenu(Gtk::Window& window, MenuView& first_menu, ClientProt
ocol& protocol, std::string&& name, int quantity):
9      SelectableListMenu(window, first_menu, protocol, std::move(name)){
10
11      Glib::RefPtr<Gtk::Builder> builder = Gtk::Builder::create_from_file(GLADE_PA
TH + "client_JoinGameMenu.glade");
12
13      builder->get_widget("error", this->error);
14      builder->get_widget("game", this->games);
15      builder->get_widget("quit_game", this->quit_game);
16
17      this->configure(quantity);
18
19      ButtonBuilder::buildButton(quit_game);
20
21      builder->get_widget("join_game_menu", this->menu);
22
23      this->addMenu();
24
25      quit_game->signal_clicked().connect(sigc::mem_fun(*this, &JoinGameMenu::quit
ButtonPressed));
26
27  }
28
29  JoinGameMenu::~JoinGameMenu(){}
30
31
32  void JoinGameMenu::selectButtonPressed(Glib::ustring game_name){
33      try{
34          this->protocol.sendString(game_name);
35          bool result = this->protocol.receiveChar();
36          if (!result){
37              this->showErrorAndRestart("Ocurrio un error al unirse a la partida");
38          } else {
39              this->waitToPlayers();
40          }
41      } catch (const SocketException& e){
42          this->showFatalError();
43      }
44  }

```

Jun 01, 18 13:12

## JoinGameMenu.h

Page 1/1

```

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

```

Jun 03, 18 12:56

## MenuView.cpp

Page 1/1

```

1  #include "MenuView.h"
2  #include "ServerFatalError.h"
3
4  MenuView::MenuView(Gtk::Window& window, MenuView& main_menu, ClientProtocol& pro
5  tocol):
6     window(window), protocol(protocol), main_menu(main_menu) {
7
8     Glib::RefPtr<Gdk::Pixbuf> aux = Gdk::Pixbuf::create_from_file(BACKGROUND_MEN
9     U_IMAGE);
10     this->background.set(aux);
11     this->menu_container.add_overlay(this->background);
12 }
13
14 MenuView::~MenuView(){
15     delete this->menu;
16 }
17
18 void MenuView::showFatalError(){
19     ServerFatalError error(this->window);
20 }
21
22 void MenuView::showErrorAndRestart(const std::string& error){
23     this->window.remove();
24     this->main_menu.showError(error);
25     this->window.add(this->main_menu.menu_container);
26 }
27
28 void MenuView::showError(const std::string& error){
29     this->error->set_label(error);
30 }
31
32 void MenuView::quitButtonPressed() {
33     this->window.close();
34 }
35
36 void MenuView::addMenu() {
37     this->menu_container.add_overlay(*this->menu);
38     this->window.add(this->menu_container);
39     this->window.show_all();
40 }

```

Jun 05, 18 14:28

## MenuView.h

Page 1/1

```

1  #ifndef __MENUVIEW_H__
2  #define __MENUVIEW_H__
3
4  #include <gtkmm/hvbox.h>
5  #include <gtkmm/label.h>
6  #include <gtkmm/window.h>
7  #include <gtkmm/overlay.h>
8  #include <gtkmm/image.h>
9  #include <memory>
10 #include "ClientProtocol.h"
11
12 class MenuView{
13     private:
14         /* Muestra un mensaje de error */
15         void showError(const std::string& error);
16
17     protected:
18         Gtk::Window& window;
19         ClientProtocol& protocol;
20         Gtk::Label* error;
21         std::unique_ptr<MenuView> next_menu;
22         MenuView& main_menu;
23         Gtk::Container* menu;
24
25         Gtk::Overlay menu_container;
26         Gtk::Image background;
27
28         /* Muestra un mensaje de error y cierra la aplicacion*/
29         void showFatalError();
30
31         /* Muestra un mensaje de error y reinicia */
32         void showErrorAndRestart(const std::string& error);
33
34         /* Agrega el menu al world y el world al window */
35         void addMenu();
36
37         /* Handler del boton de salir */
38         void quitButtonPressed();
39
40     public:
41         /* Constructor */
42         MenuView(Gtk::Window& window, MenuView& main_menu, ClientProtocol& proto
43 col);
44
45         /* Destructor */
46         virtual ~MenuView();
47
48 };
49 #endif

```

Jun 01, 18 14:20

## SelectableListMenu.cpp

Page 1/1

```

1  #include "SelectableListMenu.h"
2
3  SelectableListMenu::SelectableListMenu(Gtk::Window& window, MenuView& first_menu
4  , ClientProtocol& protocol, std::string& name):
5      MenuView(window, first_menu, protocol), player_name(std::move(name)){}
6
7  SelectableListMenu::~SelectableListMenu(){}
8
9  void SelectableListMenu::configure(int quantity){
10     try{
11         for (int i = 0; i < quantity; i++){
12             std::string field = this->protocol.receiveString();
13             this->addField(field);
14         }
15     } catch (const SocketException& e){
16         this->showFatalError();
17     }
18
19     for (auto it = this->fields.begin(); it != this->fields.end(); ++it){
20         this->games->pack_start(it->getContainer());
21     }
22     this->games->show();
23 }
24
25 void SelectableListMenu::addField(const std::string& field_name){
26     GameMenuField field(field_name);
27     this->fields.push_back(std::move(field));
28     this->fields.back().getButton().signal_clicked().connect(sigc::bind<Glib::us
29 tring>(sigc::mem_fun(*this,
30
31                                     &SelectableListMenu::selectButtonPressed), field_name));
32
33 }
34
35 bool SelectableListMenu::createPlayer(){
36     try{
37         this->player = std::unique_ptr<Player>(new Player(std::move(this->protoc
38 ol), this->player_name, this->window));
39     } catch (const std::exception& e){
40         this->showFatalError();
41     }
42     return false;
43 }
44
45 void SelectableListMenu::waitToPlayers(){
46     this->window.remove();
47     this->window.add(this->waiting_label.getWidget());
48     this->window.show_all();
49     sigc::slot<bool> my_slot = sigc::mem_fun(*this, &SelectableListMenu::createP
50 layer);
51     Glib::signal_idle().connect(my_slot);
52 }

```

May 31, 18 13:56

## SelectableListMenu.h

Page 1/1

```

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

```

Jun 01, 18 13:45

## ServerFatalError.cpp

Page 1/1

```

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

```

Jun 01, 18 13:44

## ServerFatalError.h

Page 1/1

```

1  #ifndef __SERVERFATALERROR_H__
2  #define __SERVERFATALERROR_H__
3
4  #include <gtkmm/window.h>
5
6  class ServerFatalError{
7      public:
8          ServerFatalError(Gtk::Window& window);
9
10         ~ServerFatalError();
11 };
12
13 #endif

```

Jun 05, 18 14:27

## ServerMenu.cpp

Page 1/2

```

1  #include "ServerMenu.h"
2  #include <gtkmm/builder.h>
3  #include "Path.h"
4  #include "ButtonBuilder.h"
5
6  ServerMenu::ServerMenu(Gtk::Window& window): window(window) {
7      Glib::RefPtr<Gtk::Builder> builder = Gtk::Builder::create_from_file(GLADE_PA
8      TH + "client_ServerMenu.glade");
9
10     builder->get_widget("error", this->error);
11     builder->get_widget("host", this->host);
12     builder->get_widget("service", this->service);
13     builder->get_widget("connect", this->connect);
14     builder->get_widget("quit_game", this->quit);
15
16     ButtonBuilder::buildButton(this->quit);
17     ButtonBuilder::buildButton(this->connect);
18
19     builder->get_widget("server_menu", this->menu);
20
21     builder->get_widget("background", this->background);
22     Glib::RefPtr<Gdk::Pixbuf> aux = Gdk::Pixbuf::create_from_file(BACKGROUND_MEN
23     U_IMAGE);
24     this->background->set(aux);
25
26     this->window.add(*this->menu);
27     this->window.show_all();
28
29     this->connect->signal_clicked().connect(sigc::mem_fun(*this, &ServerMenu::co
30     nnectButtonPressed));
31     this->quit->signal_clicked().connect(sigc::mem_fun(*this, &ServerMenu::quitB
32     uttonPressed));
33 }
34
35 ServerMenu::~ServerMenu() {
36     delete this->menu;
37 }
38
39 void ServerMenu::connectButtonPressed() {
40     std::string host(this->host->get_text());
41     if (host.empty()) {
42         this->error->set_label("Debe ingresar un host");
43         return;
44     }
45
46     std::string service(this->service->get_text());
47     if (service.empty()) {
48         this->error->set_label("Debe ingresar un servicio");
49         return;
50     }
51
52     this->connectToServer(host, service);
53 }
54
55 void ServerMenu::quitButtonPressed() {
56     this->window.close();
57 }
58
59 void ServerMenu::connectToServer(const std::string &host, const std::string &ser
60     vice){
61     try{
62         Socket socket(Socket::Client(host.c_str(), service.c_str()));
63         this->protocol.reset(new ClientProtocol(std::move(socket), this->window)
64     );
65
66         this->window.remove();
67         this->next_menu = std::unique_ptr<MenuView>(new GameMenu(this->window, *

```

Jun 05, 18 14:27

## ServerMenu.cpp

Page 2/2

```

        this->protocol));
    } catch (const SocketException& e) {
    }
    this->error->set_label("No pudo conectarse al servidor");
}

```

Jun 05, 18 14:28

## ServerMenu.h

Page 1/1

```

1  #ifndef __SERVERMENU__
2  #define __SERVERMENU__
3
4  #include <gtkmm/application.h>
5  #include <gtkmm/hvbox.h>
6  #include <gtkmm/button.h>
7  #include <gtkmm/entry.h>
8  #include <gtkmm/label.h>
9  #include <gtkmm/window.h>
10 #include <gtkmm/overlay.h>
11 #include <gtkmm/image.h>
12 #include <string>
13 #include <memory>
14 #include "ClientProtocol.h"
15 #include "GameMenu.h"
16 #include "MenuView.h"
17
18 /* Menu de conexion con el servidor */
19 class ServerMenu{
20     private:
21         Gtk::Label* error;
22         Gtk::Entry* host;
23         Gtk::Entry* service;
24         Gtk::Button* connect;
25         Gtk::Button* quit;
26         Gtk::Window& window;
27         Gtk::Container* menu;
28         std::unique_ptr<MenuView> next_menu;
29         std::unique_ptr<ClientProtocol> protocol;
30         Gtk::Image* background;
31
32         /* Handler del boton de conexion */
33         void connectButtonPressed();
34
35         /* Handler del boton de salir */
36         void quitButtonPressed();
37
38         /* Intenta realizar una conexion con el servidor */
39         void connectToServer(const std::string &host, const std::string &service
40 );
41     public:
42         /* Constructor */
43         ServerMenu(Gtk::Window& window);
44
45         /* Destructor */
46         ~ServerMenu();
47 };
48
49 #endif

```

May 27, 18 22:46

**WaitingLabel.cpp**

Page 1/1

```

1  #include "WaitingLabel.h"
2
3  const std::string begining("<span size='20000'>");
4  const std::string ending("</span>");
5
6  WaitingLabel::WaitingLabel() {
7      this->label.set_use_markup(true);
8      this->label.set_markup(begining + "Esperando jugadores..." + ending);
9      this->label.show();
10 }
11
12 WaitingLabel::~WaitingLabel() {}
13
14 Gtk::Widget& WaitingLabel::getWidget() {
15     return this->label;
16 }

```

Jun 03, 18 12:56

**WaitingLabel.h**

Page 1/1

```

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

```



Jun 03, 18 12:56

## MusicPath.h

Page 1/1

```

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

```

Jun 03, 18 12:56

## MusicPlayer.cpp

Page 1/3

```

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

```

Jun 03, 18 12:56

**MusicPlayer.cpp**

Page 2/3

```

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);

```

Jun 03, 18 12:56

**MusicPlayer.cpp**

Page 3/3

```

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

```

May 26, 18 12:13

**MusicPlayerException.cpp**

Page 1/1

```

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

```

May 22, 18 11:29

**MusicPlayerException.h**

Page 1/1

```

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

```

Jun 03, 18 12:56

## MusicPlayer.h

Page 1/2

```

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 encarga 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. Adem 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 da 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 };

```

Jun 03, 18 12:56

## MusicPlayer.h

Page 2/2

```

67
68
69 #endif

```

Jun 06, 18 20:08

Player.cpp

Page 1/2

```

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

```

Jun 06, 18 20:08

Player.cpp

Page 2/2

```

64     this->protocol.sendChangeWeapon(weapon);
65 }
66 }
67
68 void Player::shoot(Position position) {
69     this->shootWeapon();
70     this->protocol.sendWeaponSelfDirectedShoot(position);
71     this->screen.getWeaponsView().updateAmmo(this->weapons.getCurrentWeapon());
72 }
73
74 void Player::playTickTime() {
75     this->musicPlayer.playTickSound();
76 }
77
78 void Player::shoot(int angle, int power, int time) {
79     this->shootWeapon();
80     if (!this->weapons.getCurrentWeapon().isTimed()) {
81         time = -1;
82     }
83     if (!this->weapons.getCurrentWeapon().hasScope()) {
84         angle = MAX_WEAPON_ANGLE * 8;
85     }
86     this->protocol.sendWeaponShoot(angle, power, time);
87     this->view_list.removeScopeVisibility();
88     this->screen.getWeaponsView().updateAmmo(this->weapons.getCurrentWeapon());
89 }
90
91 ViewsList& Player::getViewsList() {
92     return this->view_list;
93 }
94
95 ScreenView& Player::getScreen() {
96     return this->screen;
97 }
98
99 WeaponList& Player::getWeapons() {
100     return this->weapons;
101 }
102
103 ClientProtocol& Player::getProtocol() {
104     return this->protocol;
105 }
106
107 MusicPlayer& Player::getMusicPlayer() {
108     return this->musicPlayer;
109 }

```

May 31, 18 12:08

## Player.h

Page 1/2

```

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

```

May 31, 18 12:08

## Player.h

Page 2/2

```

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

```

Jun 05, 18 14:07

## Scope.cpp

Page 1/1

```

1  #include "Scope.h"
2  #include "Path.h"
3  #include "WeaponNames.h"
4
5  Scope::Scope(WorldView& world): world(world){
6      this->scope.set(SCOPE_IMAGE);
7      this->angle = DEFAULT_ANGLE;
8      this->world.addElement(this->scope, Position(0,0), 0, 0);
9  }
10
11 Scope::~Scope(){}
12
13 void Scope::update(int angle, WormView& worm){
14     this->angle = angle;
15     char dir = worm.getDir();
16     if (dir == DIR_LEFT)
17         angle = 180 - angle;
18     this->world.moveScope(this->scope, worm.getWidget(), angle);
19     this->scope.show();
20     worm.updateScope(this->angle);
21 }
22
23 void Scope::update(WormView& worm){
24     this->update(this->angle, worm);
25 }
26
27
28 void Scope::hide(){
29     if (this->scope.is_visible()){
30         this->scope.hide();
31     }
32 }

```

Jun 02, 18 18:22

## Scope.h

Page 1/1

```

1  #ifndef __SCOPE_H__
2  #define __SCOPE_H__
3
4  #include <gtkmm/image.h>
5  #include "WorldView.h"
6  #include "WormView.h"
7
8  class Scope{
9      private:
10         Gtk::Image scope;
11         WorldView& world;
12         int angle;
13
14     public:
15         /* Constructor */
16         Scope(WorldView& world);
17
18         /* Destructor */
19         ~Scope();
20
21         /* Actualiza la posicion del scope */
22         void update(int angle, WormView& worm);
23
24         /* Actualiza la posicion del scope */
25         void update(WormView& worm);
26
27         /* Esconde el scope */
28         void hide();
29
30     };
31
32 #endif

```

Jun 06, 18 20:08

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         if (id == this->current_worm_id){
23             this->player.endTurnEarly();
24         }
25         it->second.removeFromWorld();
26         this->worms.erase(it);
27         this->musicPlayer.playDeathSound();
28         this->checkMovingWorms();
29     }
30 }
31
32 void ViewsList::removeWeapon(int id){
33     auto it = this->weapons.find(id);
34     if (it != this->weapons.end()) {
35         if(it->second.getName() != BAT_NAME) {
36             this->musicPlayer.playExplosionSound(it->second.getName());
37             ExplosionView explosion(std::move(it->second));
38             this->animation.addAndStart(std::move(explosion));
39         }
40         this->weapons.erase(it);
41
42         if (this->weapon_focused == id){
43             this->weapon_focused = -2;
44             this->checkMovingWorms();
45         }
46     }
47 }
48
49 void ViewsList::updateWormData(int id, int player_id, float pos_x, float pos_y,
int life, char dir, bool colliding){
50     auto it = this->worms.find(id);
51     Position pos(pos_x / UNIT_TO_SEND, pos_y / UNIT_TO_SEND);
52     if (it == this->worms.end()){
53         //Worm no existe
54         WormView worm(this->world, life, dir, pos, player_id);
55         this->worms.insert(std::make_pair(id, std::move(worm)));
56         this->players_list.addPlayerLife(player_id, life);
57     } else {
58         //Worm existe
59         int current_life = it->second.getLife();
60         if (current_life != life){
61             this->players_list.reducePlayerLife(player_id, current_life - life);
62             if (id == this->current_worm_id){

```

Jun 06, 18 20:08

ViewsList.cpp

Page 2/3

```

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

```



Jun 06, 18 20:08

## ViewsList.cpp

Page 3/3

```

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

```

Jun 06, 18 20:08

## ViewsList.h

Page 1/2

```

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

```

Jun 06, 18 20:08

## ViewsList.h

Page 2/2

```

63
64     /* Actualiza la posicion del scope */
65     void updateScope(int angle);
66
67     /* Esconde la vista del scope */
68     void removeScopeVisibility();
69
70     /* Agrega una viga a la vista en la posicion indicada y
71      * con la rotacion indicada */
72     void addGirder(size_t size, float pos_x, float pos_y, int rotation);
73
74     /* Actualiza el worm actual y hace focus en este */
75     void setCurrentWorm(int id);
76
77     /* Actualiza la imagen de los worms ganadores por la animacion
78      * de los worms festejando */
79     void setVictory();
80
81     /* Chequea si el gusano actual se esta moviendo, caso contrario
82      * le da el focus a otro */
83     void checkMovingWorms();
84
85     /* Realiza la animacion del disparo del arma */
86     void shoot(const std::string& weapon);
87 };
88
89
90 #endif

```

Jun 05, 18 14:07

## WeaponPowerAccum.cpp

Page 1/1

```

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

```

May 31, 18 12:08

## WeaponPowerAccum.h

Page 1/1

```

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

```

Jun 06, 18 20:16

## ClientProtocol.cpp

Page 1/2

```

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

```

Jun 06, 18 20:16

## ClientProtocol.cpp

Page 2/2

```

64     this->sendBuffer(buffer);
65 }
66
67 void ClientProtocol::receiveStartGame(){
68     Buffer buffer = std::move(this->receiveBuffer());
69 }
70
71 void ClientProtocol::receiveBackgroundImage(WorldView& world){
72     Buffer buffer = std::move(this->receiveBuffer());
73     world.setBackgroundImage(buffer);
74 }
75
76 void ClientProtocol::receivePlayers(PlayersList& players_list){
77     Buffer buffer = std::move(this->receiveBuffer());
78     int quantity = this->receiveIntBuffer(buffer);
79
80     for (int i = 0; i < quantity; i++){
81         Buffer buffer = std::move(this->receiveBuffer());
82
83         int id = this->receiveIntBuffer(buffer);
84         std::string name = this->receiveStringBuffer(buffer);
85
86         players_list.addPlayer(id, name);
87     }
88 }
89
90 void ClientProtocol::receiveGirders(ViewsList& viewsList){
91     Buffer buffer = std::move(this->receiveBuffer());
92     int quantity = this->receiveIntBuffer(buffer);
93
94     for (int i = 0; i < quantity; i++){
95         Buffer buffer = std::move(this->receiveBuffer());
96
97         int size = this->receiveIntBuffer(buffer);
98         float pos_x = this->receiveIntBuffer(buffer) / UNIT_TO_SEND;
99         float pos_y = this->receiveIntBuffer(buffer) / UNIT_TO_SEND;
100         int rotation = this->receiveIntBuffer(buffer);
101         viewsList.addGirder(size, pos_x, pos_y, rotation);
102     }
103 }
104
105 void ClientProtocol::receiveWeaponsAmmo(WeaponList& weapon_list){
106     Buffer buffer = std::move(this->receiveBuffer());
107     int quantity = this->receiveIntBuffer(buffer);
108
109     for (int i = 0; i < quantity; i++){
110         Buffer buffer = std::move(this->receiveBuffer());
111
112         std::string name = this->receiveStringBuffer(buffer);
113         int ammo = this->receiveIntBuffer(buffer);
114         weapon_list.add(name, ammo);
115     }
116 }
117
118 void ClientProtocol::sendBuffer(Buffer &buffer){
119     try{
120         Protocol::sendBuffer(buffer);
121     } catch (const std::exception& e){
122         ServerFatalError error(this->window);
123     }
124 }

```

May 31, 18 16:52

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

```

May 31, 18 16:52

## ClientProtocol.h

Page 2/2

```

67     void receiveWeaponsAmmo(WeaponList& weapon_list);
68
69     /* Envia el contenido del buffer */
70     void sendBuffer(Buffer &buffer) override;
71 };
72
73 #endif

```

Jun 06, 18 20:08

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

```

Jun 06, 18 20:08

## DataReceiver.cpp

Page 2/2

```

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

```

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 analyzeReceivedData(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

```

Jun 06, 18 20:08

## Handlers.cpp

Page 1/3

```

1  #include "Handlers.h"
2  #include <gtkmm/adjustment.h>
3  #include <gdk/gdkkeysyms.h>
4  #include "Player.h"
5  #include "ViewPositionTransformer.h"
6  #include "WeaponNames.h"
7
8  const char SPACE = ' ';
9  const int WEAPONS_DEFAULT_TIME = 3;
10 const char ASCII_OFFSET = 48;
11 const char ASCII_1 = 49;
12 const char ASCII_5 = 53;
13 const int MAX_TIME = 3000;
14 const int ANGLE_STEP = 6;
15
16 Handlers::Handlers(Player& player, ViewsList& view_list, WeaponList& weapons, Wo
rldView& world):
17     player(player), view_list(view_list), weapons(weapons), world(world),
18     scroll_handler(world.getWindow()), power_accumulator(*this, MAX_TIME){
19     this->has_shoot = false;
20     this->current_angle = DEFAULT_ANGLE;
21     this->weapons_time = WEAPONS_DEFAULT_TIME;
22     this->enabled = false;
23 }
24
25 Handlers::~Handlers() {}
26
27 void Handlers::enableAll(){
28     this->weapons_time = WEAPONS_DEFAULT_TIME;
29     this->current_angle = DEFAULT_ANGLE;
30     this->has_shoot = false;
31
32     this->player.getProtocol().updateScope(DEFAULT_ANGLE);
33
34     this->world.getWindow().get_parent()->get_parent()->set_can_focus(true);
35     this->world.getWindow().get_parent()->get_parent()->grab_focus();
36
37     this->world.getWindow().get_parent()->get_parent()->signal_key_press_event()
38     .connect(sigc::mem_fun(*this,
39                             &Handlers::keyPressHandler));
40
41     this->world.getWindow().get_parent()->get_parent()->signal_key_release_event
42     ().connect(sigc::mem_fun(*this,
43                             &Handlers::keyReleaseHandler));
44
45     this->world.getWindow().signal_button_press_event().connect(sigc::mem_fun(*t
his, &Handlers::onButtonPressEvent));
46
47     this->enabled = true;
48 }
49
50 void Handlers::disableAll() {
51     this->world.getWindow().get_parent()->get_parent()->signal_key_press_event()
52     .connect(sigc::mem_fun(*this,
53                             &Handlers::inactiveKeyHandler));
54
55     this->world.getWindow().get_parent()->get_parent()->signal_key_release_event
56     ().connect(sigc::mem_fun(*this,
57                             &Handlers::inactiveKeyHandler));
58
59     this->world.getWindow().signal_button_press_event().connect(sigc::mem_fun(*t
his, &Handlers::inactiveButtonHandler));
60
61     this->enabled = false;
62 }

```

Jun 06, 18 20:08

## Handlers.cpp

Page 2/3

```

56 void Handlers::powerAccumStopped(int power){
57     this->player.shoot(this->current_angle, power, this->weapons_time);
58 }
59
60 bool Handlers::keyPressHandler(GdkEventKey *key_event) {
61     if (key_event->keyval == GDK_KEY_Left) {
62         this->player.getProtocol().sendMoveAction(MOVE_LEFT);
63     } else if (key_event->keyval == GDK_KEY_Right) {
64         this->player.getProtocol().sendMoveAction(MOVE_RIGHT);
65     } else if (key_event->keyval == GDK_KEY_Return) {
66         this->player.getProtocol().sendMoveAction(JUMP);
67     } else if (key_event->keyval == GDK_KEY_BackSpace) {
68         this->player.getProtocol().sendMoveAction(ROLLBACK);
69     } else if (key_event->keyval == GDK_KEY_Up) {
70         if (!this->weapons.getCurrentWeapon().hasScope()) {
71             return true;
72         }
73         if (this->current_angle < MAX_WEAPON_ANGLE) {
74             this->current_angle += ANGLE_STEP;
75
76             this->player.getProtocol().updateScope(this->current_angle);
77         } else if (key_event->keyval == GDK_KEY_Down) {
78             if (!this->weapons.getCurrentWeapon().hasScope()) {
79                 return true;
80             }
81             if (this->current_angle > MIN_WEAPON_ANGLE) {
82                 this->current_angle -= ANGLE_STEP;
83
84                 this->player.getProtocol().updateScope(this->current_angle);
85             } else if (key_event->keyval >= ASCII_1 && key_event->keyval <= ASCII_5) {
86                 this->weapons_time = key_event->keyval - ASCII_OFFSET;
87             } else if (key_event->keyval == SPACE && key_event->type == GDK_KEY_PRESS) {
88                 if (this->weapons.getCurrentWeapon().isSelfDirected()) {
89                     return true;
90                 }
91                 if (!this->weapons.getCurrentWeapon().hasAmmo()) {
92                     return true;
93                 }
94                 if (this->has_shoot) {
95                     return true;
96                 }
97                 this->has_shoot = true;
98                 if (!this->weapons.getCurrentWeapon().hasVariablePower()) {
99                     this->player.shoot(this->current_angle, -1, this->weapons_time);
100                 } else {
101                     this->power_accumulator.start();
102                 }
103             }
104             return true;
105         }
106     }
107
108 bool Handlers::keyReleaseHandler(GdkEventKey *key_event) {
109     if (key_event->type == GDK_KEY_RELEASE) {
110         if (key_event->keyval == SPACE) {
111             if (this->weapons.getCurrentWeapon().isSelfDirected()) {
112                 return true;
113             }
114             if (!this->weapons.getCurrentWeapon().hasVariablePower()) {
115                 return true;
116             }
117             if (!this->weapons.getCurrentWeapon().hasAmmo()) {
118                 this->player.getMusicPlayer().playNoAmmo();
119                 return true;
120             }
121             this->power_accumulator.stop();
122         }

```

Jun 06, 18 20:08

## Handlers.cpp

Page 3/3

```

122     }
123     return true;
124 }
125
126 bool Handlers::onButtonPressEvent (GdkEventButton *event) {
127     if (!this->weapons.getCurrentWeapon().isSelfDirected()) {
128         return true;
129     }
130     if (!this->weapons.getCurrentWeapon().hasAmmo()) {
131         this->player.getMusicPlayer().playNoAmmo();
132         return true;
133     }
134     if (this->has_shoot) {
135         return true;
136     }
137     if ((event->type == GDK_BUTTON_PRESS) && (event->button == 1)) {
138         float x = event->x;
139         float y = event->y;
140         x += this->world.getWindow().get_hadjustment()->get_value();
141         y += this->world.getWindow().get_vadjustment()->get_value();
142         Position position(x, y);
143         Position newPosition = ViewPositionTransformer(this->world.getLayout()).
transformToPosition(position);
144         this->has_shoot = true;
145         this->player.shoot(newPosition);
146     }
147     return true;
148 }
149
150 bool Handlers::inactiveKeyHandler (GdkEventKey *key_event) {
151     return true;
152 }
153
154 bool Handlers::inactiveButtonHandler (GdkEventButton *event) {
155     return true;
156 }
157
158 int Handlers::getCurrentAngle() const{
159     return this->current_angle;
160 }
161
162 bool Handlers::isEnabled(){
163     return this->enabled;
164 }

```

Jun 05, 18 14:07

## Handlers.h

Page 1/2

```

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     public:
31         /* Constructor */
32         Handlers(Player& player, ViewsList& view_list, WeaponList& weapons, Worl
dView& world);
33
34         /* Destructor */
35         ~Handlers();
36
37         /* Handler completo para el presionado de teclas. Indica
38            los pasos que se deben realizar al presionar una tecla
39            especifica */
40         bool keyPressHandler(GdkEventKey *key_event);
41
42         /* Handler completo para la liberaci3n de teclas. Indica
43            los pasos que se deben realizar al liberar una tecla
44            especifica */
45         bool keyReleaseHandler(GdkEventKey *key_event);
46
47         /* Handler del mouse. Indica los pasos que se deben realizar
48            al utilizar el mouse */
49         bool onButtonPressEvent (GdkEventButton *event);
50
51         /* Handler inactivo de las teclas. Indica que no se debe
52            realizar ninguna accion al producirse un evento */
53         bool inactiveKeyHandler(GdkEventKey *key_event);
54
55         /* Handler inactivo del mouse. Indica que no se debe
56            realizar ninguna accion al producirse un evento */
57         bool inactiveButtonHandler(GdkEventButton *event);
58
59         /* Habilita todos los handlers */
60         void enableAll();
61
62         /* Deshabilita todos los handlers */
63         void disableAll();
64
65

```



Jun 05, 18 14:07

**Handlers.h**

Page 2/2

```

66      /* Realiza el shoot del player */
67      void powerAccumStopped(int power);
68
69      /* Devuelve el angulo actual del scope */
70      int getCurrentAngle() const;
71
72      /* Devuelve true si estan habilitados los handlers */
73      bool isEnabled();
74  };
75
76  #endif

```

May 30, 18 20:03

**PlayerLifeLabel.cpp**

Page 1/1

```

1  #include "PlayerLifeLabel.h"
2  #include "GamePlayers.h"
3
4  const std::string begining("<span color='");
5  const std::string middle(">");
6  const std::string ending("</span>");
7
8  PlayerLifeLabel::PlayerLifeLabel(): id(0), player_name(""), life(0){
9      this->label.set_use_markup(true);
10 }
11
12 PlayerLifeLabel::~PlayerLifeLabel(){}
13
14 void PlayerLifeLabel::setPlayerName(int id, const std::string& player_name){
15     this->id = id;
16     this->player_name = player_name;
17     this->updateLabel();
18 }
19
20 void PlayerLifeLabel::addLife(int life){
21     this->life += life;
22     this->updateLabel();
23 }
24
25 void PlayerLifeLabel::reduceLife(int life){
26     this->life -= life;
27     this->updateLabel();
28 }
29
30 Gtk::Label& PlayerLifeLabel::getLabel(){
31     return this->label;
32 }
33
34 void PlayerLifeLabel::updateLabel(){
35     std::string message = begining + colors[this->id] + middle;
36     message += std::to_string(this->id) + "-" + this->player_name;
37     message += ":" + std::to_string(this->life) + ending;
38     this->label.set_markup(message);
39 }

```

May 30, 18 20:03

## PlayerLifeLabel.h

Page 1/1

```

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

```

Jun 06, 18 20:08

## PlayersList.cpp

Page 1/1

```

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

```

May 30, 18 20:03

## PlayersList.h

Page 1/1

```

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

```

Jun 06, 18 20:08

## ScreenView.cpp

Page 1/1

```

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

```

May 31, 18 12:08

## ScreenView.h

Page 1/1

```

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

```

May 31, 18 12:08

## 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 tu 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::setWinner(const std::string& winner, bool i_win){
36     this->message.set_markup(begining + "Termino el juego" + ending);
37     std::string winner_message;
38     if (winner.empty()){
39         winner_message = "Empate";
40     } else if (i_win) {
41         winner_message = "GANASTE!!!!";
42     } else {
43         winner_message = "Perdiste :( El ganador fue: " + winner;
44     }
45     this->time.set_markup(begining + winner_message + ending);
46 }
47
48 Gtk::Container& TurnLabel::getWindow() {
49     return this->label;
50 }

```

May 31, 18 12:08

## TurnLabel.h

Page 1/1

```

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

```

Jun 05, 18 14:07

## WeaponButton.cpp

Page 1/1

```

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

```

Jun 05, 18 15:28

## WeaponButton.h

Page 1/1

```

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

```

Jun 03, 18 12:56

## WeaponView.cpp

Page 1/1

```

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

```

Jun 03, 18 12:56

## WeaponView.h

Page 1/1

```

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

```

Jun 02, 18 13:59

## WindView.cpp

Page 1/1

```

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

```

May 30, 18 22:01

## WindView.h

Page 1/1

```

1  #ifndef __WINDVIEW_H__
2  #define __WINDVIEW_H__
3
4  #include <gtkmm/hvbox.h>
5  #include <gtkmm/label.h>
6  #include <gtkmm/image.h>
7
8  class WindView{
9  private:
10     Gtk::VBox container;
11     Gtk::Label velocity;
12     Gtk::Image direction;
13
14  public:
15     WindView();
16     ~WindView();
17
18     //Actualiza la vista del viento
19     void update(float wind);
20
21     Gtk::VBox& getWindow();
22 };
23
24 #endif
25

```

Jun 06, 18 20:39

## WorldView.cpp

Page 1/2

```

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

```



Jun 06, 18 20:39

## WorldView.cpp

Page 2/2

```

62 void WorldView::setFocus(Gtk::Widget& element){
63     this->window.get_hadjustment()->set_value(element.get_allocation().get_x() -
64     this->window.get_hadjustment()->get_page_size() / 2);
65     this->window.get_vadjustment()->set_value(element.get_allocation().get_y() -
66     this->window.get_vadjustment()->get_page_size() / 2);
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.get_pointer(), image.get_max_size());
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

Turn.cpp

Page 1/1

```

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

```

May 31, 18 12:08

Turn.h

Page 1/1

```

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

```

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

```

Jun 05, 18 14:07

## GirderView.cpp

Page 1/1

```

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

```

Jun 05, 18 14:07

**GirderView.h**

Page 1/1

```

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

```

May 28, 18 18:21

**Viewable.cpp**

Page 1/1

```

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

```

May 31, 18 12:08

## Viewable.h

Page 1/1

```

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

```

Jun 06, 18 20:08

## WormLifeView.cpp

Page 1/1

```

1  #include "WormLifeView.h"
2
3  const std::string begining("<span color='");
4  const std::string middle("><b>");
5  const std::string ending("</b></span>");
6
7  WormLifeView::WormLifeView(int life, const std::string& color): color(color){
8      this->label.set_use_markup(true);
9      this->updateLife(life);
10 }
11
12 WormLifeView::~WormLifeView(){}
13
14 WormLifeView::WormLifeView(WormLifeView&& other):
15     label(std::move(other.label)), color(std::move(other.color)){}
16
17 void WormLifeView::updateLife(int life){
18     this->label.override_background_color(Gdk::RGBA(this->color));
19     this->label.set_markup(begining + "white" + middle + std::to_string(life) + e
20 nding);
21 }
22
23 Gtk::Widget& WormLifeView::getWidget(){
24     return this->label;
25 }

```

May 27, 18 21:56

## WormLifeView.h

Page 1/1

```

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

```

Jun 06, 18 20:08

## WormView.cpp

Page 1/2

```

1  #include "WormView.h"
2  #include <string>
3  #include <glibmm/main.h>
4  #include "ObjectSizes.h"
5  #include "WeaponNames.h"
6  #include "GamePlayers.h"
7
8  WormView::WormView(WorldView& worldView, int life, char dir, Position pos, int p
9  layer_id):
10     Viewable(worldView), player_id(player_id), life(life), is_moving(false),
11     last_position(Position(-1, -1)), label(life, colors[player_id]),
12     walkingAnimation(&this->image), weaponAnimation(DEFAULT_WEAPON, &this->image
13 ) {
14     this->worm.attach(this->label.getWidget(), 0, 0, 1, 1);
15     this->worm.attach(this->image, 0, 1, 1, 1);
16     this->walkingAnimation.setStaticImage();
17     this->addToWorld(pos, worm_size, worm_size + 0.5);
18 }
19
20 WormView::~WormView() {}
21
22 WormView::WormView(WormView&& other): Viewable(std::move(other)), player_id(oth
23 er.player_id),
24     life(other.life), is_moving(other.is_moving),
25     last_position(other.last_position), label(std::move(other.label)),
26     image(std::move(other.image)),
27     worm(std::move(other.worm)), walkingAnimation(std::move(other.walkingAnimati
28 on)),
29     weaponAnimation(std::move(other.weaponAnimation)) {
30     this->weaponAnimation.updateWormImage(&this->image);
31     this->walkingAnimation.updateWormImage(&this->image);
32 }
33
34 void WormView::updateData(int new_life, char new_dir, const Position& new_pos, b
35 ool colliding, bool is_current_worm, bool has_shot) {
36     if (new_life != this->life){
37         this->label.updateLife(new_life);
38     }
39     this->life = new_life;
40     this->is_moving = !(this->last_position == new_pos);
41     this->last_position = new_pos;
42     this->setNewImage(new_dir, colliding, is_current_worm, has_shot);
43     this->move(new_pos, worm_size, worm_size + 0.5);
44 }
45
46 void WormView::updateScope(int angle) {
47     this->weaponAnimation.changeAngle(angle, this->getDir());
48 }
49
50 void WormView::changeWeapon(const std::string& weapon) {
51     this->weaponAnimation.changeWeapon(weapon, this->getDir());
52 }
53
54 void WormView::setNewImage(char dir, bool colliding, bool is_current_worm, bool
55 has_shot){
56     if (is_current_worm){
57         if (!this->is_moving && !has_shot && colliding){
58             this->weaponAnimation.setWeaponImage(dir);
59         } else if (colliding){
60             this->walkingAnimation.setMovementImage(dir);
61         }
62         return;
63     }
64     this->walkingAnimation.setStaticImage();
65 }
66

```

Jun 06, 18 20:08

## WormView.cpp

Page 2/2

```

61  Gtk::Widget& WormView::getWidget() {
62      return this->worm;
63  }
64
65  Gtk::Image& WormView::getImage() {
66      return this->image;
67  }
68
69  int WormView::getLife() const {
70      return this->life;
71  }
72
73  char WormView::getDir() const {
74      return this->walkingAnimation.getDir();
75  }
76
77  int WormView::getPlayerId() const {
78      return this->player_id;
79  }
80
81  bool WormView::isMoving() const {
82      return this->is_moving;
83  }
84
85  void WormView::setVictory() {
86      this->image.set(VICTORY_ANIMATION);
87  }
88
89  void WormView::weaponShoot(const std::string& weapon) {
90      this->weaponAnimation.weaponShootAnimation(weapon, this->getDir());
91  }
92
93  void WormView::resetFocus() {
94      this->is_moving = false;
95      this->setFocus(false);
96      this->walkingAnimation.setStaticImage();
97  }

```

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 has_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& iter, const int width);
42
43     public:
44         /* Constructor */
45         WormView(WorldView& worldView, int life, char dir, Position pos, int player_id);
46
47         /* Destructor */
48         ~WormView();
49
50         /* Constructor por movimiento */
51         WormView(WormView&& other);
52
53         /* Actualiza la posicion y vida del worm */
54         void updateData(int new_life, char new_dir, const Position& new_pos, bool colliding, bool is_current_worm, bool has_shot);
55
56         /* Actualiza la imagen del arma con el angulo actual */
57         void updateScope(int angle);
58
59         /* Actualiza el arma del worm y cambia la imagen */
60         void changeWeapon(const std::string &weapon);
61
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  #endif
96

```

May 26, 18 12:13

**DistanceWeapon.cpp**

Page 1/1

```

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

```

May 27, 18 21:56

**DistanceWeapon.h**

Page 1/1

```

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

```

May 26, 18 12:13

**MeleeWeapon.cpp**

Page 1/1

```

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

```

May 27, 18 21:56

**MeleeWeapon.h**

Page 1/1

```

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

```

May 26, 18 12:13

**AirAttack.cpp**

Page 1/1

```

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

```

May 27, 18 21:56

**AirAttack.h**

Page 1/1

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

May 26, 18 12:13

**Banana.cpp**

Page 1/1

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

May 27, 18 21:56

**Banana.h**

Page 1/1

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

May 26, 18 12:13

**Bat.cpp**

Page 1/1

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

May 31, 18 12:08

Bat.h

Page 1/1

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

May 26, 18 12:13

Bazooka.cpp

Page 1/1

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

May 27, 18 21:56

**Bazooka.h**

Page 1/1

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

May 26, 18 12:13

**Dynamite.cpp**

Page 1/1

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

May 27, 18 21:56

**Dynamite.h**

Page 1/1

```

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

```

May 26, 18 12:13

**GreenGrenade.cpp**

Page 1/1

```

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

```



May 31, 18 12:08

**GreenGrenade.h**

Page 1/1

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

May 26, 18 12:13

**HolyGrenade.cpp**

Page 1/1

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

May 31, 18 12:08

**HolyGrenade.h**

Page 1/1

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

May 26, 18 12:13

**Mortar.cpp**

Page 1/1

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

May 27, 18 21:56

**Mortar.h**

Page 1/1

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

May 26, 18 12:13

**RedGrenade.cpp**

Page 1/1

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

May 31, 18 12:08

**RedGrenade.h**

Page 1/1

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

May 26, 18 12:13

**Teleportation.cpp**

Page 1/1

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

May 31, 18 12:08

**Teleportation.h**

Page 1/1

```

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

```

May 26, 18 12:13

**SelfDirectedWeapon.cpp**

Page 1/1

```

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

```

May 31, 18 12:08

## SelfDirectedWeapon.h

Page 1/1

```

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

```

May 26, 18 12:13

## Weapon.cpp

Page 1/1

```

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

```

May 27, 18 21:56

## Weapon.h

Page 1/1

```

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

```

May 27, 18 21:56

## WeaponList.cpp

Page 1/1

```

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

```

May 31, 18 12:08

## WeaponList.h

Page 1/1

```

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

```

May 27, 18 21:56

## WeaponsFactory.cpp

Page 1/1

```

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

```



May 28, 18 18:21WeaponsFactory.hPage 1/1

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

Jun 06, 18 22:31Table of ContentPage 1/2

1	Table of Contents				
2	1 ExplosionView.cpp...	sheets	1 to	1 ( 1) pages	1- 1 44 lines
3	2 ExplosionView.h.....	sheets	1 to	1 ( 1) pages	2- 2 39 lines
4	3 ExplosionViewList.cpp	sheets	2 to	2 ( 1) pages	3- 3 24 lines
5	4 ExplosionViewList.h.	sheets	2 to	2 ( 1) pages	4- 4 31 lines
6	5 WalkingAnimation.cpp	sheets	3 to	3 ( 1) pages	5- 5 44 lines
7	6 WalkingAnimation.h..	sheets	3 to	3 ( 1) pages	6- 6 45 lines
8	7 WeaponAnimation.cpp.	sheets	4 to	4 ( 1) pages	7- 8 80 lines
9	8 WeaponAnimation.h...	sheets	5 to	5 ( 1) pages	9- 9 56 lines
10	9 main.cpp.....	sheets	5 to	5 ( 1) pages	10- 10 21 lines
11	10 ButtonBuilder.cpp...	sheets	6 to	6 ( 1) pages	11- 11 12 lines
12	11 ButtonBuilder.h.....	sheets	6 to	6 ( 1) pages	12- 12 14 lines
13	12 CreateGameMenu.cpp..	sheets	7 to	7 ( 1) pages	13- 13 60 lines
14	13 CreateGameMenu.h....	sheets	7 to	7 ( 1) pages	14- 14 29 lines
15	14 GameMenu.cpp.....	sheets	8 to	8 ( 1) pages	15- 16 76 lines
16	15 GameMenuField.cpp...	sheets	9 to	9 ( 1) pages	17- 17 30 lines
17	16 GameMenuField.h.....	sheets	9 to	9 ( 1) pages	18- 18 35 lines
18	17 GameMenu.h.....	sheets	10 to	10 ( 1) pages	19- 19 35 lines
19	18 JoinGameMenu.cpp....	sheets	10 to	10 ( 1) pages	20- 20 45 lines
20	19 JoinGameMenu.h.....	sheets	11 to	11 ( 1) pages	21- 21 25 lines
21	20 MenuView.cpp.....	sheets	11 to	11 ( 1) pages	22- 22 39 lines
22	21 MenuView.h.....	sheets	12 to	12 ( 1) pages	23- 23 49 lines
23	22 SelectableListMenu.cpp	sheets	12 to	12 ( 1) pages	24- 24 47 lines
24	23 SelectableListMenu.h	sheets	13 to	13 ( 1) pages	25- 25 48 lines
25	24 ServerFatalError.cpp	sheets	13 to	13 ( 1) pages	26- 26 11 lines
26	25 ServerFatalError.h..	sheets	14 to	14 ( 1) pages	27- 27 14 lines
27	26 ServerMenu.cpp.....	sheets	14 to	15 ( 2) pages	28- 29 65 lines
28	27 ServerMenu.h.....	sheets	15 to	15 ( 1) pages	30- 30 50 lines
29	28 WaitingLabel.cpp....	sheets	16 to	16 ( 1) pages	31- 31 17 lines
30	29 WaitingLabel.h.....	sheets	16 to	16 ( 1) pages	32- 32 24 lines
31	30 MusicPath.h.....	sheets	17 to	17 ( 1) pages	33- 33 26 lines
32	31 MusicPlayer.cpp.....	sheets	17 to	18 ( 2) pages	34- 36 139 lines
33	32 MusicPlayerException.cpp	sheets	19 to	19 ( 1) pages	37- 37 13 lines
34	33 MusicPlayerException.h	sheets	19 to	19 ( 1) pages	38- 38 23 lines
35	34 MusicPlayer.h.....	sheets	20 to	20 ( 1) pages	39- 40 70 lines
36	35 Player.cpp.....	sheets	21 to	21 ( 1) pages	41- 42 110 lines
37	36 Player.h.....	sheets	22 to	22 ( 1) pages	43- 44 84 lines
38	37 Scope.cpp.....	sheets	23 to	23 ( 1) pages	45- 45 33 lines
39	38 Scope.h.....	sheets	23 to	23 ( 1) pages	46- 46 33 lines
40	39 ViewsList.cpp.....	sheets	24 to	25 ( 2) pages	47- 49 176 lines
41	40 ViewsList.h.....	sheets	25 to	26 ( 2) pages	50- 51 91 lines
42	41 WeaponPowerAccum.cpp	sheets	26 to	26 ( 1) pages	52- 52 36 lines
43	42 WeaponPowerAccum.h..	sheets	27 to	27 ( 1) pages	53- 53 36 lines
44	43 ClientProtocol.cpp..	sheets	27 to	28 ( 2) pages	54- 55 125 lines
45	44 ClientProtocol.h....	sheets	28 to	29 ( 2) pages	56- 57 74 lines
46	45 DataReceiver.cpp....	sheets	29 to	30 ( 2) pages	58- 59 97 lines
47	46 DataReceiver.h.....	sheets	30 to	30 ( 1) pages	60- 60 34 lines
48	47 Handlers.cpp.....	sheets	31 to	32 ( 2) pages	61- 63 165 lines
49	48 Handlers.h.....	sheets	32 to	33 ( 2) pages	64- 65 77 lines
50	49 PlayerLifeLabel.cpp.	sheets	33 to	33 ( 1) pages	66- 66 40 lines
51	50 PlayerLifeLabel.h...	sheets	34 to	34 ( 1) pages	67- 67 40 lines
52	51 PlayersList.cpp.....	sheets	34 to	34 ( 1) pages	68- 68 41 lines
53	52 PlayersList.h.....	sheets	35 to	35 ( 1) pages	69- 69 46 lines
54	53 ScreenView.cpp.....	sheets	35 to	35 ( 1) pages	70- 70 64 lines
55	54 ScreenView.h.....	sheets	36 to	36 ( 1) pages	71- 71 62 lines
56	55 TurnLabel.cpp.....	sheets	36 to	36 ( 1) pages	72- 72 51 lines
57	56 TurnLabel.h.....	sheets	37 to	37 ( 1) pages	73- 73 45 lines
58	57 WeaponButton.cpp....	sheets	37 to	37 ( 1) pages	74- 74 40 lines
59	58 WeaponButton.h.....	sheets	38 to	38 ( 1) pages	75- 75 38 lines
60	59 WeaponView.cpp.....	sheets	38 to	38 ( 1) pages	76- 76 30 lines
61	60 WeaponView.h.....	sheets	39 to	39 ( 1) pages	77- 77 42 lines
62	61 WindView.cpp.....	sheets	39 to	39 ( 1) pages	78- 78 31 lines
63	62 WindView.h.....	sheets	40 to	40 ( 1) pages	79- 79 26 lines
64	63 WorldView.cpp.....	sheets	40 to	41 ( 2) pages	80- 81 96 lines
65	64 WorldView.h.....	sheets	41 to	42 ( 2) pages	82- 83 67 lines
66	65 Turn.cpp.....	sheets	42 to	42 ( 1) pages	84- 84 41 lines

Jun 06, 18 22:31

## Table of Content

Page 2/2

67	66	Turn.h.....	sheets	43 to	43 ( 1)	pages	85- 85	39 lines
68	67	BulletView.cpp.....	sheets	43 to	43 ( 1)	pages	86- 86	31 lines
69	68	BulletView.h.....	sheets	44 to	44 ( 1)	pages	87- 87	37 lines
70	69	GirderView.cpp.....	sheets	44 to	44 ( 1)	pages	88- 88	27 lines
71	70	GirderView.h.....	sheets	45 to	45 ( 1)	pages	89- 89	32 lines
72	71	Viewable.cpp.....	sheets	45 to	45 ( 1)	pages	90- 90	28 lines
73	72	Viewable.h.....	sheets	46 to	46 ( 1)	pages	91- 91	46 lines
74	73	WormLifeView.cpp....	sheets	46 to	46 ( 1)	pages	92- 92	25 lines
75	74	WormLifeView.h.....	sheets	47 to	47 ( 1)	pages	93- 93	32 lines
76	75	WormView.cpp.....	sheets	47 to	48 ( 2)	pages	94- 95	98 lines
77	76	WormView.h.....	sheets	48 to	49 ( 2)	pages	96- 97	97 lines
78	77	DistanceWeapon.cpp..	sheets	49 to	49 ( 1)	pages	98- 98	17 lines
79	78	DistanceWeapon.h....	sheets	50 to	50 ( 1)	pages	99- 99	24 lines
80	79	MeleeWeapon.cpp.....	sheets	50 to	50 ( 1)	pages	100-100	12 lines
81	80	MeleeWeapon.h.....	sheets	51 to	51 ( 1)	pages	101-101	20 lines
82	81	AirAttack.cpp.....	sheets	51 to	51 ( 1)	pages	102-102	10 lines
83	82	AirAttack.h.....	sheets	52 to	52 ( 1)	pages	103-103	20 lines
84	83	Banana.cpp.....	sheets	52 to	52 ( 1)	pages	104-104	10 lines
85	84	Banana.h.....	sheets	53 to	53 ( 1)	pages	105-105	20 lines
86	85	Bat.cpp.....	sheets	53 to	53 ( 1)	pages	106-106	10 lines
87	86	Bat.h.....	sheets	54 to	54 ( 1)	pages	107-107	20 lines
88	87	Bazooka.cpp.....	sheets	54 to	54 ( 1)	pages	108-108	10 lines
89	88	Bazooka.h.....	sheets	55 to	55 ( 1)	pages	109-109	20 lines
90	89	Dynamite.cpp.....	sheets	55 to	55 ( 1)	pages	110-110	10 lines
91	90	Dynamite.h.....	sheets	56 to	56 ( 1)	pages	111-111	20 lines
92	91	GreenGrenade.cpp....	sheets	56 to	56 ( 1)	pages	112-112	11 lines
93	92	GreenGrenade.h.....	sheets	57 to	57 ( 1)	pages	113-113	20 lines
94	93	HolyGrenade.cpp.....	sheets	57 to	57 ( 1)	pages	114-114	11 lines
95	94	HolyGrenade.h.....	sheets	58 to	58 ( 1)	pages	115-115	20 lines
96	95	Mortar.cpp.....	sheets	58 to	58 ( 1)	pages	116-116	10 lines
97	96	Mortar.h.....	sheets	59 to	59 ( 1)	pages	117-117	20 lines
98	97	RedGrenade.cpp.....	sheets	59 to	59 ( 1)	pages	118-118	11 lines
99	98	RedGrenade.h.....	sheets	60 to	60 ( 1)	pages	119-119	20 lines
100	99	Teleportation.cpp...	sheets	60 to	60 ( 1)	pages	120-120	10 lines
101	100	Teleportation.h....	sheets	61 to	61 ( 1)	pages	121-121	20 lines
102	101	SelfDirectedWeapon.cpp	sheets	61 to	61 ( 1)	pages	122-122	13 lines
103	102	SelfDirectedWeapon.h	sheets	62 to	62 ( 1)	pages	123-123	23 lines
104	103	Weapon.cpp.....	sheets	62 to	62 ( 1)	pages	124-124	56 lines
105	104	Weapon.h.....	sheets	63 to	63 ( 1)	pages	125-125	53 lines
106	105	WeaponList.cpp.....	sheets	63 to	63 ( 1)	pages	126-126	29 lines
107	106	WeaponList.h.....	sheets	64 to	64 ( 1)	pages	127-127	40 lines
108	107	WeaponsFactory.cpp..	sheets	64 to	64 ( 1)	pages	128-128	41 lines
109	108	WeaponsFactory.h....	sheets	65 to	65 ( 1)	pages	129-129	25 lines