```
1
2   #include <Path.h>
3   #include <string>
4   #include <vector>
5   #include <gtkmm/image.h>
6   #include "FileBoxController.h"
7   #include "FileWriter.h"
8   #include "FileReader.h"
9   #include "InvalidMapError.h"
10
11  static const char *const NEW_FILE_NAME = "Sin titulo.yaml";
12
13  FileBoxController::FileBoxController(UsablesController &wep_controller,
14          std::shared_ptr<MapController> map_controller,
15          const Glib::RefPtr<Gtk::Builder> &builder )
16          : usables_controller(wep_controller),
17            map_controller(std::move(map_controller))
18  {
19      builder->get_widget("save_dialog",save_dialog);
20      save_dialog->add_button("Cancelar", Gtk::RESPONSE_CANCEL);
21      save_dialog->add_button("Guardar", Gtk::RESPONSE_OK);
22
23      builder->get_widget("map_name",map_name);
24
25      builder->get_widget("open_dialog",open_dialog);
26      open_dialog->add_button("Cancelar", Gtk::RESPONSE_CANCEL);
27      open_dialog->add_button("Abrir", Gtk::RESPONSE_OK);
28  }
29
30  void FileBoxController::onSaveClicked() const {
31      try {
32          std::vector<std::vector<double>> worms;
33          std::vector<std::vector<double>> girders;
34          map_controller->getObjects(worms, girders);
35          auto background = map_controller->getBackground();
36
37          std::vector<int> weapons_ammo;
38          unsigned int life;
39          usables_controller.getWeaponsAndLife(weapons_ammo, life);
40
41          save_dialog->set_current_folder(MAPS_PATH);
42          save_dialog->set_current_name(map_name->get_text());
43          int result = save_dialog->run();
44          if (result==Gtk::RESPONSE_OK){
45              std::string path = save_dialog->get_filename();
46              std::string filename = save_dialog->get_current_name();
47              map_name->set_label(filename);
48
49              FileWriter file(path);
50              file.save(weapons_ammo, worms, girders, life, background);
51          }
52          save_dialog->hide();
53      } catch(const InvalidMapError &error){
54          error.what();
55      }
56  }
57
58  void FileBoxController::onLoadClicked() const {
59      open_dialog->set_current_folder(MAPS_PATH);
60      int result = open_dialog->run();
61      if (result==Gtk::RESPONSE_OK) {
62          std::string filename = open_dialog->get_filename();
63          map_name->set_label(open_dialog->get_current_name());
64
65          std::vector<std::vector<double>> worms;
66          std::vector<std::vector<double>> girders;
```



```
67          std::vector<int> weps_ammo;
68          unsigned int life;
69          Glib::RefPtr<Gdk::Pixbuf> background;
70
71          FileReader file(filename);
72          file.read(worms, girders,
73                    weps_ammo, life, background);
74
75          map_controller->loadBackground(background);
76          usables_controller.loadWeapons(weps_ammo, life);
77          map_controller->loadObjects(worms, girders);
78      }
79      open_dialog->hide();
80  }
81
82
83  void FileBoxController::onNewClicked() const {
84      map_name->set_label(NEW_FILE_NAME);
85      usables_controller.onResetSignal();
86      map_controller->newMapSignal();
87  }
88
```

jun 07, 18 23:33 **FileBoxController.h** Page 1/1

```
1
2  #ifndef WORMS_FILECONTROLLER_H
3  #define WORMS_FILECONTROLLER_H
4
5  #include <gtkmm/filechooserdialog.h>
6  #include "FileBoxView.h"
7  #include "UsablesController.h"
8  #include "MapController.h"
9
10 // Clase que se encarga de establecer una conexion entre la seccion de archivos
11 // y  el resto del programa
12 class FileBoxController {
13 private:
14     UsablesController &usables_controller;
15     std::shared_ptr<MapController> map_controller;
16     Gtk::FileChooserDialog* save_dialog;
17     Gtk::FileChooserDialog* open_dialog;
18     Gtk::Label* map_name;
19
20 public:
21     FileBoxController(UsablesController &wep_controller,
22                       std::shared_ptr<MapController> map_controller,
23                       const Glib::RefPtr<Gtk::Builder> &builder);
24
25     // Se encarga de mostrar un cuadro de dialogo para seleccionar un archivo
26     // cuando se eligio guardar en la vista
27     void onSaveClicked() const;
28
29     // Se encarga de mostrar un cuadro de dialogo para seleccionar un archivo
30     // cuando se eligio cargar en la vista
31     void onLoadClicked() const;
32
33     // Crea un nuevo mapa y actualiza la informacion del nombre del mapa actual
34     void onNewClicked() const;
35 };
36
37
38 #endif //WORMS_FILECONTROLLER_H
```

jun 19, 18 14:51 **MapController.cpp** Page 1/3

```
1
2  #include <gtkmm/messagedialog.h>
3  #include <ViewPositionTransformer.h>
4  #include <vector>
5  #include <string>
6  #include "MapController.h"
7  #include "InvalidMapError.h"
8  #include "Path.h"
9
10 #define ADD_MODE_ID 0
11 #define MOVE_CMD_ID 1
12 #define SELECT_MODE_ID 2
13
14 typedef const Glib::RefPtr<Gtk::Builder> Builder;
15
16 MapController::MapController(Map model, Builder &builder):
17         model(std::move(model)), item_id_to_add(1),
18         actual_mode(ADD_MODE_ID){
19     builder->get_widget_derived("map", view);
20     builder->get_widget_derived("toolbox", toolBox);
21     view->bindController(this);
22     toolBox->bindController(this);
23
24     builder->get_widget("background_dialog",background_dialog);
25     background_dialog->add_button("Cancelar", Gtk::RESPONSE_CANCEL);
26     background_dialog->add_button("Abrir", Gtk::RESPONSE_OK);
27 }
28
29 void MapController::addModeSignal(const unsigned int &id) {
30     this->actual_mode = ADD_MODE_ID;
31     this->item_id_to_add = id;
32 }
33
34 void MapController::eraseSignal() {
35     model.erase(index_object_selected);
36     view->erase(index_object_selected);
37     toolBox->hideSelected();
38     toolBox->disableMovingItems();
39     actual_mode = SELECT_MODE_ID;
40 }
41
42 void MapController::newMapSignal() {
43     model.clean();
44     view->clean();
45     toolBox->closeSelectionMode();
46 }
47
48 void MapController::moveSignal() {
49     this->actual_mode = MOVE_CMD_ID;
50 }
51
52 void MapController::changeModeSignal() {
53     this->actual_mode = (actual_mode==ADD_MODE_ID? SELECT_MODE_ID:ADD_MODE_ID);
54     if (actual_mode==ADD_MODE_ID) toolBox->closeSelectionMode();
55 }
56
57 void MapController::turn(const int &rotation) {
58     if (model.isGirder(index_object_selected)) {
59         unsigned int id;
60         int new_angle = this->model.turn(index_object_selected, id, rotation);
61         this->view->turn(id, new_angle, index_object_selected);
62     }
63 }
64
65 void MapController::turnCCWSignal()  {
66     turn(10);
```

```
67  }
68
69  void MapController::turnCWSignal()  {
70      turn(-10);
71  }
72
73  void MapController::mapClickedSignal(GdkEventButton *event_button) {
74      if (actual_mode == MOVE_CMD_ID) {
75          this->model.move(index_object_selected, event_button->x,
76                           event_button->y);
77          this->view->move(index_object_selected, event_button->x,
78                           event_button->y);
79          actual_mode = SELECT_MODE_ID;
80      } else if (actual_mode == SELECT_MODE_ID) {
81          this->index_object_selected = view->select(event_button->x,
82                                                     event_button->y);
83          if (index_object_selected > -1) {
84              toolBox->enableMovingItems();
85              toolBox->showSelected(model.getItemID(index_object_selected));
86          } else {
87              toolBox->disableMovingItems();
88              toolBox->hideSelected();
89          }
90          //cambio de estado del toolbox llama a add mode
91          actual_mode = SELECT_MODE_ID;
92      } else {
93          this->model.add(item_id_to_add, event_button->x, event_button->y);
94          this->view->add(item_id_to_add, event_button->x, event_button->y);
95      }
96  }
97
98  void MapController::getObjects(std::vector<std::vector<double>> &worms,
99                                 std::vector<std::vector<double>> &girders) const{
100     model.getObjects(worms, girders);
101     if (worms.empty()){
102         throw InvalidMapError("El mapa actual no contiene worms");
103     }
104     if (girders.empty()){
105         throw InvalidMapError("El mapa actual no contiene vigas");
106     }
107
108     ViewPositionTransformer transformer(*view);
109     for (std::vector<double> &worm : worms){
110         Position position(worm[0],worm[1]);
111         Position new_pos = transformer.transformToPosition(position);
112         worm[0] = new_pos.getX();
113         worm[1] = new_pos.getY();
114     }
115     for (std::vector<double> &girder : girders){
116         Position position(girder[1],girder[2]);
117         Position new_pos = transformer.transformToPosition(position);
118         girder[1] = new_pos.getX();
119         girder[2] = new_pos.getY();
120     }
121 }
122
123 void MapController::loadObjects(std::vector<std::vector<double>> &worms,
124                                 std::vector<std::vector<double>> &girders) {
125     newMapSignal();
126     ViewPositionTransformer transformer(*view);
127     for (std::vector<double> &girder:girders) {
128         Position position(girder[1],girder[2]);
129         Position new_pos = transformer.transformToScreen(position);
130         girder[1] = new_pos.getX();
131         girder[2] = new_pos.getY();
132         this->model.add(girder[0], girder[1], girder[2], girder[3]);
```



```
133         this->view->add(girder[0], girder[1], girder[2], girder[3]);
134     }
135     for (std::vector<double> &worm:worms) {
136         Position position(worm[0],worm[1]);
137         Position new_pos = transformer.transformToScreen(position);
138         worm[0] = new_pos.getX();
139         worm[1] = new_pos.getY();
140         this->model.add(1, worm[0], worm[1]);
141         this->view->add(1, worm[0], worm[1]);
142     }
143     this->view->setInitialPosition();
144 }
145
146 void MapController::changeBackgroundSignal() const {
147     this->background_dialog->set_current_folder(BACKGROUND_PATH);
148     int result = this->background_dialog->run();
149     if (result==Gtk::RESPONSE_OK) {
150         std::string path = this->background_dialog->get_filename();
151         this->view->changeBackground(path);
152     }
153     this->background_dialog->hide();
154 }
155
156 Glib::RefPtr<const Gdk::Pixbuf> MapController::getBackground() const{
157     return view->getBackground();
158 }
159
160 void MapController::loadBackground(Glib::RefPtr<Gdk::Pixbuf> &background)  {
161     view->loadBackground(background);
162 }
```

```
1
2   #ifndef WORMS_MAPCONTROLLER_H
3   #define WORMS_MAPCONTROLLER_H
4
5   #include <gtkmm/filechooserdialog.h>
6   #include <string>
7   #include <vector>
8   #include "MapView.h"
9   #include "Map.h"
10  #include "ToolBoxView.h"
11
12  class MapView;
13  class ToolBoxView;
14
15  // Clase que se encarga de comunicar la vista con el modelo, y a su vez, se
16  // comunica con el resto del programa
17  class MapController {
18      Map model;
19      MapView *view;
20      ToolBoxView *toolBox;
21      unsigned int item_id_to_add;
22      unsigned int actual_mode;
23      int index_object_selected;
24      Gtk::FileChooserDialog* background_dialog;
25
26      void turn(const int &rotation);
27
28  public:
29      /* Constructor */
30      MapController(Map model, const Glib::RefPtr<Gtk::Builder> &builder);
31
32      /* Cambia al modo de agregado, en donde el objeto
33       * a agregar es el de id */
34      void addModeSignal(const unsigned int &id);
35
36      /* Envia una seÃ±al de borrado */
37      void eraseSignal();
38
39      /* Envia una seÃ±al de nuevo mapa */
40      void newMapSignal();
41
42      /* Envia una seÃ±al de movimiento */
43      void moveSignal();
44
45      /* Envia una seÃ±al de rotacion anti horario */
46      void turnCCWSignal();
47
48      /* Envia una seÃ±al de click sobre el mapa */
49      void mapClickedSignal(GdkEventButton *event_button);
50
51      /* Obtiene los objetos del mapa */
52      void getObjects(std::vector<std::vector<double>> &worms,
53                      std::vector<std::vector<double>> &girders) const;
54
55      /* Carga los worms y las vigas en el mapa */
56      void loadObjects(std::vector<std::vector<double>> &worms,
57                       std::vector<std::vector<double>> &girders);
58
59      /* Envia una seÃ±al de rotacion horaria */
60      void turnCWSignal();
61
62      /* Envia una seÃ±al de cambio de imagen de fondo */
63      void changeBackgroundSignal() const;
64
65      /* Envia una seÃ±al de cambio de modo */
66      void changeModeSignal();
```



```
67
68      /* Devuelve la imagen de fondo */
69      Glib::RefPtr<const Gdk::Pixbuf> getBackground() const;
70
71      /* Carga la imagen de fondo */
72      void loadBackground(Glib::RefPtr<Gdk::Pixbuf> &background);
73  };
74
75
76  #endif //WORMS_MAPCONTROLLER_H
```

jun 10, 18 19:29 **UsablesController.cpp** Page 1/1

```cpp

#include "UsablesController.h"
#include "InvalidMapError.h"
#include <vector>

UsablesController::UsablesController(const Glib::RefPtr<Gtk::Builder> &builder){
    builder->get_widget("btn_reset", reset_button);
    reset_button->signal_clicked().connect(sigc::mem_fun(
            *this, &UsablesController::onResetSignal));

    builder->get_widget_derived("life", life_spinner);

    for (size_t i = 1; i <= 10; ++i) {
        std::shared_ptr<WeaponView> weapon_view(new WeaponView(builder, i));

        std::shared_ptr<Weapon> weapon
                (new Weapon(weapon_view->getInitialAmmo()));

        weapons.push_back(weapon);

        std::shared_ptr<WeaponController> weapon_controller(
                new WeaponController(weapon_view, weapon));
        wep_controllers.push_back(std::move(weapon_controller));
        weapons_view.push_back(weapon_view);
    }
}

void UsablesController::onResetSignal() {
    life_spinner->reset();
    for (auto &actual_controller : wep_controllers){
        actual_controller->resetAmmo();
    }
}

void UsablesController::getWeaponsAndLife(std::vector<int> &weps_ammo,
                                          unsigned int &life) const {
    life = life_spinner->get_value();
    for (auto &actual_controller : wep_controllers) {
        weps_ammo.push_back(actual_controller->getAmmo());
    }
    if (!isValidWeaponSet(weps_ammo)){
        throw InvalidMapError("NingÃºn arma tiene municiÃ³n");
    }
}

void UsablesController::loadWeapons(std::vector<int> &weps_ammo,
                                    const unsigned int &life) const {
    int i = 0;
    for (const std::shared_ptr<WeaponController> &actual_controller
            :wep_controllers) {
        actual_controller->updateAmmo(weps_ammo[i]);
        i++;
    }
    life_spinner->update(life);
}

bool
UsablesController::isValidWeaponSet(std::vector<int> &ammo_vector) const {
    for (int actual_ammo : ammo_vector) {
        if (actual_ammo != 0)
            return true;
    }
    return false;
}
```

jun 10, 18 19:29 **UsablesController.h** Page 1/1

```cpp

#ifndef WORMS_WEAPONSLISTCONTROLLER_H
#define WORMS_WEAPONSLISTCONTROLLER_H


#include <gtkmm/button.h>
#include <gtkmm/spinbutton.h>
#include <vector>
#include "Weapon.h"
#include "WeaponView.h"
#include "LifeView.h"

// Clase que se encaga de manejar la comunicacion de la vida y el arma con las
// demas partes del programa
class UsablesController {
private:
    LifeView *life_spinner;
    Gtk::Button *reset_button;
    std::vector<std::shared_ptr<Weapon>> weapons;
    std::vector<std::shared_ptr<WeaponView>> weapons_view;
    std::vector<std::shared_ptr<WeaponController> > wep_controllers;

    // Indica si el set actual de armas es valido (alguno con municion positiva)
    bool isValidWeaponSet(std::vector<int> &ammo_vector) const;

public:
    explicit UsablesController(
            const Glib::RefPtr<Gtk::Builder> &builder);

    // Indica a los controladores de armas y vida que deben reiniciarse
    void onResetSignal();

    // Obtiene a los valores actuales de las armas y la vida
    void getWeaponsAndLife(std::vector<int> &ammo, unsigned int &life) const;

    // Establece los valores de las armas y la vida
    void
    loadWeapons(std::vector<int> &weps_ammo, const unsigned int &life) const;
};

#endif //WORMS_WEAPONSLISTCONTROLLER_H
```

jun 04, 18 21:54 **WeaponController.cpp** Page 1/1

```cpp

#include "WeaponController.h"

WeaponController::WeaponController(std::shared_ptr<WeaponView> View,
                                   std::shared_ptr<Weapon> model)
        : weapon_view(std::move(View)),
          weapon_model(std::move(model)) {
    weapon_view->bindController(this);
}

void WeaponController::resetAmmo() {
    weapon_view->resetAmmo();
    weapon_model->resetAmmo();
}

void WeaponController::updateAmmo(const int &ammo) {
    weapon_model->setAmmo(ammo);
    weapon_view->setAmmo(ammo);
}

int WeaponController::getAmmo() {
    return weapon_model->getAmmo();
}
```

jun 07, 18 23:33 **WeaponController.h** Page 1/1

```cpp

#ifndef WORMS_WEAPONCONTROLLER_H
#define WORMS_WEAPONCONTROLLER_H

#include "WeaponView.h"
#include "Weapon.h"
class WeaponView;

// Clase que se encarga de manejar la informacion del arma entre el modelo
// y la vista
class WeaponController {
private:
    std::shared_ptr<WeaponView> weapon_view;
    std::shared_ptr<Weapon> weapon_model;
public:
    WeaponController(std::shared_ptr<WeaponView>,
                     std::shared_ptr<Weapon>
                     model);

    // Indica a la vista y al modelo que deben resetear la municion
    void resetAmmo();

    // Indica a la vista y al modelo que deben establecer un nuevo valor de
    // municion especificado
    void updateAmmo(const int &ammo);

    // Obtiene el valor de la municion desde el modelo
    int getAmmo();
};


#endif //WORMS_WEAPONCONTROLLER_H
```

jun 10, 18 19:29 **main.cpp** Page 1/1

```cpp
#include <gtkmm/application.h>
#include <gtkmm/builder.h>
#include <giomm.h>
#include <iostream>
#include <gtkmm/scrolledwindow.h>
#include <gtkmm/window.h>
#include "Editor.h"
#include "Path.h"

int main() {
    Glib::RefPtr<Gtk::Application> app = Gtk::Application::create();
    Glib::RefPtr<Gtk::Builder> refBuilder = Gtk::Builder::create();
    try {
        refBuilder->add_from_file(GLADE_PATH + "editor.glade");
    }
    catch(const Glib::FileError &ex) {
        std::cerr << "FileError: " << ex.what() << std::endl;
        return 1;
    }
    catch(const Glib::MarkupError &ex) {
        std::cerr << "MarkupError: " << ex.what() << std::endl;
        return 1;
    }
    catch(const Gtk::BuilderError &ex) {
        std::cerr << "BuilderError: " << ex.what() << std::endl;
        return 1;
    }

    Editor *mainWindow = nullptr;
    refBuilder->get_widget_derived("main_window", mainWindow);
    if (mainWindow) {
        mainWindow->set_title(EDITOR_WINDOW_NAME);
        mainWindow->set_icon_from_file(ICON_PATH);
        app->run(*mainWindow);
        delete mainWindow;
    }
    return 0;
}
```

jun 10, 18 19:29 **Editor.cpp** Page 1/1

```cpp

#include "Editor.h"

typedef const Glib::RefPtr<Gtk::Builder> Builder;

Editor::Editor(BaseObjectType *cobject, Builder &builder):
        Gtk::Window(cobject),
        usables_controller(builder) {
    maximize();
    builder->get_widget("map_window", map_window);

    std::shared_ptr<MapController> map_controller
            (new MapController(map_model, builder));

    builder->get_widget_derived("filebox", filebox);
    std::shared_ptr<FileBoxController> filebox_controller(
            new FileBoxController(usables_controller, map_controller,builder));
    filebox->bindController(filebox_controller);

    show_all_children();
}
```

jun 10, 18 19:29 **Editor.h** Page 1/1

```
1
2  #ifndef WORMS_EDITOR_H
3  #define WORMS_EDITOR_H
4
5  #include <gtkmm/builder.h>
6  #include <gtkmm/window.h>
7  #include <gtkmm/scrolledwindow.h>
8  #include <gtkmm/spinbutton.h>
9  #include "MapView.h"
10 #include "ToolBoxView.h"
11 #include "UsablesController.h"
12 #include "FileBoxController.h"
13 #include "FileBoxView.h"
14
15 class Editor : public Gtk::Window {
16     Gtk::ScrolledWindow *map_window;
17     Map map_model;
18     UsablesController usables_controller;
19     FileBoxView *filebox;
20
21 public:
22     Editor(BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);
23 };
24
25 #endif //WORMS_EDITOR_H
```

jun 21, 18 12:36 **FileReader.cpp** Page 1/1

```
1
2  #include "FileReader.h"
3  #include <map>
4  #include <string>
5  #include <vector>
6  #include <giomm/memoryinputstream.h>
7  #include "Buffer.h"
8
9  FileReader::FileReader(const std::string &filename):
10     file(filename, std::fstream::in),
11     filename(filename) {}
12
13 void FileReader::read(std::vector<std::vector<double>> &worms,
14                       std::vector<std::vector<double>> &girders,
15                       std::vector<int> &weps_ammo,
16                       unsigned int &worms_life,
17                       Glib::RefPtr<Gdk::Pixbuf> &background) {
18     YAML::Node config = YAML::LoadFile(filename);
19
20     worms_life = config[WORMS_LIFE].as<unsigned int>();
21
22     std::map<std::string, int> ammo =
23                     config[WEAPON_AMMO].as<std::map<std::string, int>>();
24
25     weps_ammo.push_back(ammo[BAZOOKA_NAME]);
26     weps_ammo.push_back(ammo[MORTAR_NAME]);
27     weps_ammo.push_back(ammo[GREEN_GRENADE_NAME]);
28     weps_ammo.push_back(ammo[RED_GRENADE_NAME]);
29     weps_ammo.push_back(ammo[BANANA_NAME]);
30     weps_ammo.push_back(ammo[AIR_ATTACK_NAME]);
31     weps_ammo.push_back(ammo[BAT_NAME]);
32     weps_ammo.push_back(ammo[TELEPORT_NAME]);
33     weps_ammo.push_back(ammo[DYNAMITE_NAME]);
34     weps_ammo.push_back(ammo[HOLY_GRENADE_NAME]);
35
36     worms = config[WORMS_DATA].as<std::vector<std::vector<double>>>();
37
38     girders = config[GIRDERS_DATA].as<std::vector<std::vector<double>>>();
39
40     std::vector<int> backgrounds =
41             config[BACKGROUND_IMAGE].as<std::vector<int>>();
42     Buffer buffer(backgrounds.size());
43     for (const int &actual : backgrounds) {
44         buffer.setNext(actual);
45     }
46
47     auto stream = Gio::MemoryInputStream::create();
48     stream->add_data(buffer.getPointer(), buffer.getMaxSize());
49     background = Gdk::Pixbuf::create_from_stream(stream);
50 }
```

jun 19, 18 14:51 **FileReader.h** Page 1/1

```
1
2  #ifndef WORMS_FILEREADER_H
3  #define WORMS_FILEREADER_H
4
5  #include <fstream>
6  #include "MapObject.h"
7  #include <yaml.h>
8  #include <WeaponNames.h>
9  #include <ConfigFields.h>
10 #include <string>
11 #include <vector>
12 #include <gtkmm/image.h>
13
14 // Clase que se encarga de manejar la carga de un mapa
15 class FileReader{
16 private:
17     std::fstream file;
18     std::string filename;
19
20 public:
21     explicit FileReader(const std::string &filename);
22
23     // Carga todos los componentes de un mapa desde un archivo YAML
24     void read(std::vector<std::vector<double>> &worms,
25               std::vector<std::vector<double>> &girders,
26               std::vector<int> &weps_ammo,
27               unsigned int &worm_life, Glib::RefPtr<Gdk::Pixbuf> &background);
28 };
29
30
31 #endif //WORMS_FILEREADER_H
```

jun 19, 18 14:51 **FileWriter.cpp** Page 1/2

```
1
2
3  #include "FileWriter.h"
4  #include <string>
5  #include <vector>
6
7  FileWriter::FileWriter(const std::string &filename):
8      file(filename, std::fstream::out | std::ios_base::trunc) {}
9
10
11 void FileWriter::save(std::vector<int> weapons,
12                 const std::vector<std::vector<double>> &worms,
13                 const std::vector<std::vector<double>> &girders,
14                 const unsigned int &worm_life,
15                 Glib::RefPtr<const Gdk::Pixbuf>& background) {
16     YAML::Emitter out;
17
18     out << YAML::BeginMap;
19
20     out << YAML::Key << WORMS_LIFE;
21     out << YAML::Value << worm_life;
22
23     out << YAML::Key << WEAPON_AMMO;
24
25     out << YAML::Value << YAML::BeginMap;
26
27     out << YAML::Key << BAZOOKA_NAME;
28     out << YAML::Value << weapons[0];
29     out << YAML::Key << MORTAR_NAME;
30     out << YAML::Value << weapons[1];
31     out << YAML::Key << GREEN_GRENADE_NAME;
32     out << YAML::Value << weapons[2];
33     out << YAML::Key << RED_GRENADE_NAME;
34     out << YAML::Value << weapons[3];
35     out << YAML::Key << BANANA_NAME;
36     out << YAML::Value << weapons[4];
37     out << YAML::Key << HOLY_GRENADE_NAME;
38     out << YAML::Value << weapons[9];
39     out << YAML::Key << DYNAMITE_NAME;
40     out << YAML::Value << weapons[8];
41     out << YAML::Key << BAT_NAME;
42     out << YAML::Value << weapons[6];
43     out << YAML::Key << AIR_ATTACK_NAME;
44     out << YAML::Value << weapons[5];
45     out << YAML::Key << TELEPORT_NAME;
46     out << YAML::Value << weapons[7];
47
48     out << YAML::EndMap;
49
50     out << YAML::Key << WORMS_DATA;
51     out << worms;
52
53     out << YAML::Key << GIRDERS_DATA;
54     out << girders;
55
56     out << YAML::Key << BACKGROUND_IMAGE;
57     out << YAML::Value << YAML::Flow << YAML::BeginSeq;
58
59     char* image;
60     gsize size;
61     background->save_to_buffer(image, size);
62
63     for (size_t i = 0; i < size; i++){
64         out << (int)image[i];
65     }
66
```

jun 19, 18 14:51 **FileWriter.cpp** Page 2/2

```
67      out << YAML::EndSeq;
68      out << YAML::EndMap;
69
70      file << out.c_str();
71  }
```

jun 19, 18 14:51 **FileWriter.h** Page 1/1

```
1
2   #ifndef WORMS_FILEWRITER_H
3   #define WORMS_FILEWRITER_H
4
5   #include <fstream>
6   #include "MapObject.h"
7   #include <yaml.h>
8   #include <WeaponNames.h>
9   #include <ConfigFields.h>
10  #include <vector>
11  #include <string>
12  #include <gtkmm/image.h>
13
14  // Clase que se encarga de manejar el guardado de un mapa
15  class FileWriter{
16  private:
17      std::fstream file;
18
19  public:
20      explicit FileWriter(const std::string &filename);
21
22      // Guarda todos los componentes de un mapa en un archivo YAML
23      void
24      save(std::vector<int> weapons,
25           const std::vector<std::vector<double>> &worms,
26           const std::vector<std::vector<double>> &girders,
27           const unsigned int &worm_life, Glib::RefPtr<const Gdk::Pixbuf>& backgro
    und);
28  };
29
30  #endif //WORMS_FILEWRITER_H
```

jun 10, 18 19:29 **InvalidMapError.cpp** Page 1/1

```cpp

#include <gtkmm/enums.h>
#include <gtkmm/messagedialog.h>
#include "InvalidMapError.h"

InvalidMapError::InvalidMapError(const char *message) noexcept:
    message(message){}

const char *InvalidMapError::what() const noexcept{
    Gtk::Window dialog_window;
    Gtk::MessageDialog dialog("Error al guardar archivo",
                              false, Gtk::MESSAGE_WARNING);
    dialog.set_transient_for(dialog_window);
    dialog.set_secondary_text(message);
    dialog.run();
    return message;
}

InvalidMapError::~InvalidMapError() {
}
```

jun 10, 18 19:29 **InvalidMapError.h** Page 1/1

```cpp

#ifndef WORMS_INVALIDMAP_H
#define WORMS_INVALIDMAP_H


#include <exception>

// Clase que se encarga de lanzar una excepcion
// cuando el mapa a guardar es invalido
class InvalidMapError : public std::exception{
private:
    const char* message;

public:
    explicit InvalidMapError(const char *message) noexcept;

    virtual const char *what() const noexcept;

    ~InvalidMapError();
};


#endif //WORMS_INVALIDMAP_H
```

jun 10, 18 19:29 **Map.cpp** Page 1/1

```cpp

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

void Map::erase(const int &index) {
    if (!contained_objects.empty())
        this->contained_objects.erase(contained_objects.begin() + index);
}

void Map::clean() {
    this->contained_objects.clear();
}

void
Map::add(const unsigned int &id, const double &x,
                        const double &y, const int &angle) {
    MapObject new_object(x, y, angle);
    contained_objects.emplace_back(std::make_pair(id, new_object));
}

void Map::move(const int &index, const double &x,const double &y) {
    MapObject &object = contained_objects[index].second;
    object.updatePosition(x, y);
}

const int Map::turn(const unsigned int &index,
                            unsigned int &id, const int &rotation) {
    MapObject &object = contained_objects[index].second;
    id = contained_objects[index].first;
    return object.turn(rotation);
}

const bool Map::isGirder(int &index) const {
    return (contained_objects[index].first > 1);
}

void Map::getObjects(std::vector<std::vector<double>> &worms,
                        std::vector<std::vector<double>> &girders) const {
    for (auto &object : contained_objects) {
        float x, y;
        object.second.getPosition(x, y);
        if (object.first == 1) {
            std::vector<double> position;
            position.push_back(x);
            position.push_back(y);
            worms.push_back(position);
        } else {
            std::vector<double> data;
            data.push_back(object.first);
            data.push_back(x);
            data.push_back(y);
            data.push_back(object.second.getAngle());
            girders.push_back(data);
        }
    }
}

const int Map::getItemID(const int &index) const{
    return contained_objects[index].first;
}
```

jun 07, 18 23:33 **Map.h** Page 1/1

```cpp

#ifndef WORMS_MAPMODEL_H
#define WORMS_MAPMODEL_H


#include <utility>
#include <vector>
#include "MapObject.h"

// Clase que se encarga de modelar el mapa
class Map {
    std::vector<std::pair<int, MapObject>> contained_objects;

public:
    // Borra el objeto que se encuentra en la posicion index del vector
    void erase(const int &index);

    // Borra todos los objetos contenidos en el mapa
    void clean();

    // Agregar un objeto en la posicion (x,y)
    void add(const unsigned int &id, const double &x, const double &y,
             const int &angle = 0);

    // Obtiene todos los objetos contenidos en el mapa separados por tipo
    void getObjects(std::vector<std::vector<double>> &worms,
                    std::vector<std::vector<double>> &girders) const;

    // Mueve el objeto en la posicion index del vector hacia la posicion
    // (x,y) del mapa
    void move(const int &index, const double &x, const double &y);

    // Devuelve verdadero si el objeto en la posicion index es una viga
    const bool isGirder(int &index) const;

    // Obtiene el tipo del objeto en la posicion index del vector
    const int getItemID(const int &index) const;

    // Gira el objeto en la posicion index del vector en un angulo indicado
    const int
    turn(const unsigned int &index, unsigned int &id, const int &rotation);
};


#endif //WORMS_MAPMODEL_H
```

jun 02, 18 20:07 **MapObject.cpp** Page 1/1

```cpp

#include <cstdlib>
#include "MapObject.h"

MapObject::MapObject(const float &x, const float &y, const int &angle) :
        position(x,y), angle(angle) {}

void MapObject::updatePosition(const float &x, const float &y) {
    position= Position(x,y);
}

int MapObject::turn(const int &rotation){
    if (angle == 0)
        angle = 180;
    return angle = abs((angle+rotation)%180);
}

void MapObject::getPosition(float &x, float &y) const {
    y=position.getY();
    x=position.getX();
}

const int MapObject::getAngle() const {
    return angle;
}


```

jun 07, 18 23:33 **MapObject.h** Page 1/1

```cpp

#ifndef WORMS_OBJECTMODEL_H
#define WORMS_OBJECTMODEL_H

#include <Position.h>

// Clase que modela un objeto contenido en el mapa
class MapObject {
    Position position;
    int angle;
public:
    MapObject(const float &x, const float &y, const int &angle = 0);

    // Actualiza la posicion en la que se encuentra el objeto
    void updatePosition(const float &x, const float &y);

    // Obtiene la posicion en la que se encuentra el objeto
    void getPosition(float &x, float &y) const;

    // Actualiza el angulo en la que se encuentra el objeto
    const int getAngle() const;

    // Gira el objeto la cantidad especificada
    int turn(const int &rotation);
};


#endif //WORMS_OBJECTMODEL_H
```

jun 02, 18 20:07 **Weapon.cpp** Page 1/1

```cpp

#include "Weapon.h"

Weapon::Weapon(const int &default_ammo)
        : default_ammo(default_ammo),
          actual_ammo(default_ammo) {}

void Weapon::resetAmmo() {
    actual_ammo = default_ammo;
}

void Weapon::setAmmo(const int &new_ammo) {
    this->actual_ammo = new_ammo;
}

int Weapon::getAmmo() const {
    return actual_ammo;
}
```

jun 07, 18 23:33 **Weapon.h** Page 1/1

```cpp

#ifndef WORMS_WEAPONMODEL_H
#define WORMS_WEAPONMODEL_H

// Clase que modela un arma
class Weapon {
private:
    const int default_ammo;
    int actual_ammo;
public:
    explicit Weapon(const int &default_ammo);

    // Establece el valor de la municion por defecto en el modelo
    void resetAmmo();

    // Establece el valor de la municion indicado en el modelo
    void setAmmo(const int &new_ammo);

    // Obtiene el valor actual de la municion
    int getAmmo() const;
};


#endif //WORMS_WEAPONMODEL_H
```

jun 10, 18 19:29 **FileBoxView.cpp** Page 1/1

```cpp

#include "FileBoxView.h"

FileBoxView::FileBoxView(BaseObjectType *cobject,
                         const Glib::RefPtr<Gtk::Builder> &builder)
        : Gtk::Grid(cobject) {
    builder->get_widget("btn_save", save);
    builder->get_widget("btn_load", load);
    builder->get_widget("btn_clean", new_map);
}

void FileBoxView::bindController(std::shared_ptr<FileBoxController> controller){
    this->file_box_controller = std::move(controller);

    save->signal_clicked().connect(
            sigc::mem_fun(*file_box_controller,
                          &FileBoxController::onSaveClicked));

    load->signal_clicked().connect(
            sigc::mem_fun(*file_box_controller,
                          &FileBoxController::onLoadClicked));

    new_map->signal_clicked().connect(
            sigc::mem_fun(*file_box_controller,
                          &FileBoxController::onNewClicked));
}
```

jun 07, 18 23:33 **FileBoxView.h** Page 1/1

```cpp

#ifndef WORMS_FILEBOXVIEW_H
#define WORMS_FILEBOXVIEW_H

#include <gtkmm/builder.h>
#include <gtkmm/hvbox.h>
#include <gtkmm/button.h>
#include <gtkmm/grid.h>
#include "FileBoxController.h"

class FileBoxController;

// Clase que se encarga de manipular la zona de archivos
class FileBoxView : public Gtk::Grid {
private:
    Gtk::Button *save;
    Gtk::Button *load;
    Gtk::Button *new_map;
    std::shared_ptr<FileBoxController> file_box_controller;
public:
    FileBoxView(BaseObjectType *cobject,
                const Glib::RefPtr<Gtk::Builder> &builder);

    // Enlaza el controlador a la vista
    void bindController(std::shared_ptr<FileBoxController> controller);
};


#endif //WORMS_FILEBOXVIEW_H
```

jun 02, 18 13:26 **LifeView.cpp** Page 1/1

```cpp

#include "LifeView.h"

LifeView::LifeView(BaseObjectType *cobject,
                   const Glib::RefPtr<Gtk::Builder> &builder)
        : Gtk::SpinButton(cobject),
          default_hp(this->get_value()) {
}

void LifeView::reset() {
    this->set_value(default_hp);
}

void LifeView::update(const unsigned int &new_life) {
    this->set_value(new_life);
}
```

jun 07, 18 23:33 **LifeView.h** Page 1/1

```cpp

#ifndef WORMS_LIFEVIEW_H
#define WORMS_LIFEVIEW_H


#include <gtkmm/spinbutton.h>
#include <gtkmm/builder.h>

// Clase que se encarga de manipular la vista de la vida
class LifeView : public Gtk::SpinButton {
private:
    const unsigned int default_hp;
public:
    LifeView(BaseObjectType *cobject,
             const Glib::RefPtr<Gtk::Builder> &builder);

    // Establece el valor por defecto de la vida
    void reset();

    // Establece un nuevo valor a mostrar en la vista de la vida
    void update(const unsigned int &new_life);
};


#endif //WORMS_LIFEVIEW_H
```

```
#include <Path.h>
#include <gtkmm/adjustment.h>
#include <gtkmm/scrolledwindow.h>
#include <glibmm/main.h>
#include <vector>
#include <string>
#include "MapView.h"
#include "GirderSize.h"

const std::string DEFAULT_BACKGROUND("default_background.png");

MapView::MapView(BaseObjectType *cobject,
                 const Glib::RefPtr<Gtk::Builder> &builder)
        : Gtk::Layout(cobject),
          scroll_handler(*(Gtk::ScrolledWindow*)this->get_parent()){
    add_events(Gdk::BUTTON_PRESS_MASK);
    signal_button_press_event().connect(
            sigc::mem_fun(*this, &MapView::onButtonClicked));

    setInitialPosition();
    changeBackground(BACKGROUND_PATH + DEFAULT_BACKGROUND);
    initializeWormsImages();
    initializeGirderImages();
}

bool MapView::onButtonClicked(GdkEventButton *button_event) {
    controller->mapClickedSignal(button_event);
    return true;
}

void MapView::setInitialPosition() {
    guint w, h;
    get_size(w, h);
    ((Gtk::ScrolledWindow*) get_parent())->get_hadjustment()->set_value(w / 2);
    ((Gtk::ScrolledWindow*) get_parent())->get_vadjustment()->set_value(h);
}

void MapView::initializeGirderImages(){
    std::vector<std::string> girder_3_imgs;
    std::vector<std::string> girder_6_imgs;

    for (int i = 0; i < 180; i = i + 10) {
        girder_3_imgs.emplace_back(
                GIRDER_PATH + "3_" + std::to_string(i) + ".png");
        girder_6_imgs.push_back(
                GIRDER_PATH + "6_" + std::to_string(i) + ".png");
    }
    objects_pallete.push_back(girder_3_imgs);
    objects_pallete.push_back(girder_6_imgs);
}

void MapView::initializeWormsImages()  {
    std::vector<std::string> worms_imgs;
    worms_imgs.emplace_back(IMAGES_PATH + "/right_worm.png");
    objects_pallete.push_back(worms_imgs);
}

void MapView::add(const unsigned int &id, const double &x, const double &y,
                  const int &angle) {
    Gtk::Image new_image(objects_pallete[id - id / 2 - 1][0]);
    const Glib::RefPtr<Gdk::Pixbuf> &img = new_image.get_pixbuf();
    int width = img->get_width();
    int height = img->get_height();
    double x_bound = x - width / 2;
    double y_bound = y - height / 2;
```



```

    put(new_image, x_bound, y_bound);
    new_image.show();
    contained_objects.push_back(std::move(new_image));
    if (angle > 0){
        sigc::slot<bool> my_slot = sigc::bind(sigc::mem_fun(
            *this, &MapView::turn), id, angle, contained_objects.size() - 1);
        Glib::signal_idle().connect(my_slot);
    }
}

void MapView::move(const int &index, const double &x, const double &y) {
    if (!contained_objects.empty()) {
        Gtk::Image &actual_object = contained_objects[index];
        Gtk::Layout::move(actual_object, x - actual_object.get_width() / 2,
                          y - actual_object.get_height() / 2);
        actual_object.show();
    }
}

bool MapView::turn(const unsigned int &id, const int &angle, const int &index) {
    if (!contained_objects.empty()) {
        Gtk::Image &image = contained_objects[index];
        float x = child_property_x(image) + image.get_width() / 2;
        float y = child_property_y(image) + image.get_height() / 2;
        image.set(objects_pallete[id - id / 2 - 1][angle / 10]);

        int height = GirderSize::getGirderHeightPixels(id, angle);
        int width = GirderSize::getGirderWidthPixels(id, angle);
        Gtk::Layout::move(image, x - width / 2, y - height / 2);
    }
    return false;
}

void MapView::erase(const int &index) {
    if (!contained_objects.empty()) {
        contained_objects[index].hide();
        contained_objects.erase(contained_objects.begin() + index);
    }
}

void MapView::clean() {
    contained_objects.clear();
    this->setInitialPosition();
}

void MapView::bindController(MapController *map_controller) {
    this->controller = map_controller;
}

void MapView::changeBackground(const std::string &path) {
    Gtk::Image bg(path);
    loadBackground(bg.get_pixbuf());
}

void MapView::redrawMap() {
    for (Gtk::Image &object : contained_objects){
        const Gtk::Allocation &alloc = object.get_allocation();
        remove(object);
        put(object,alloc.get_x(),alloc.get_y());
    }
    this->water.show(*this);
}

int MapView::select(const double &x, const double &y) {
    Gdk::Rectangle new_object(x, y, 1, 1);
```

jun 19, 18 14:51 **MapView.cpp** Page 3/3

```
133     for (ssize_t i = contained_objects.size() - 1; i >= 0; i--) {
134         bool collision = contained_objects[i].intersect(new_object);
135         if (collision) {
136             return i;
137         }
138     }
139     return -1;
140 }
141
142 Glib::RefPtr<const Gdk::Pixbuf> MapView::getBackground() const{
143     return this->background[0].get_pixbuf();
144 }
145
146 void MapView::loadBackground(Glib::RefPtr<Gdk::Pixbuf> pixbuf) {
147     background.clear();
148     int img_width = pixbuf->get_width();
149     int img_height = pixbuf->get_height();
150     guint window_width, window_height;
151     this->get_size(window_width, window_height);
152     for (size_t x = 0; x < window_width; x += img_width) {
153         for (size_t y = 0; y < window_height; y += img_height) {
154             Gtk::Image image(pixbuf);
155             image.show();
156             put(image, x, y);
157             background.push_back(std::move(image));
158         }
159     }
160     redrawMap();
161 }
162
```

jun 19, 18 14:51 **MapView.h** Page 1/2

```
1
2  #ifndef WORMS_MAP_H
3  #define WORMS_MAP_H
4
5  #include <gtkmm/builder.h>
6  #include <gtkmm/layout.h>
7  #include <gtkmm/image.h>
8  #include <string>
9  #include <vector>
10 #include "MapController.h"
11 #include "Water.h"
12 #include "ScrollHandler.h"
13
14 class MapController;
15
16 // Clase que se encarga de manipular la vista del mapa
17 class MapView : public Gtk::Layout {
18 private:
19     std::vector<Gtk::Image> contained_objects;
20     std::vector<std::vector<std::string>> objects_pallete;
21     MapController *controller;
22     std::vector<Gtk::Image> background;
23     Water water;
24     ScrollHandler scroll_handler;
25
26
27     // Inicializa el vector de imagenes de los worms
28     void initializeWormsImages();
29
30     // Inicializa el vector de imagenes de las vigas
31     void initializeGirderImages();
32
33     // Dibuja nuevamente el contenido del mapa
34     void redrawMap();
35
36 public:
37     MapView(BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);
38
39     // Se ejecuta al clickear el mapa
40     bool onButtonClicked(GdkEventButton *button_event);
41
42     // Borra el objeto en la posiciÃ³n indicada
43     void erase(const int &index);
44
45     // Elimina todo el contenido del mapa
46     void clean();
47
48     // Enlaza el controlador a la vista
49     void bindController(MapController *map_controller);
50
51     // Establece la posicion inicial del mapa a mostrar
52     void setInitialPosition();
53
54     // Agregar un nuevo objeto al mapa, en la posicion (x,y)
55     void add(const unsigned int &id, const double &x, const double &y,
56              const int &angle = 0);
57
58     // Gira el objeto seleccionado
59     bool turn(const unsigned int &id, const int &angle, const int &index);
60
61     // Cambia el fondo actual
62     void changeBackground(const std::string &path);
63
64     // Selecciona el objeto en la posiciÃ³n (x,y)
65     int select(const double &x, const double &y);
66
```

jun 19, 18 14:51 **MapView.h** Page 2/2

```
67      // Mueve el objeto seleccionado a la posicÃ³n (x.y)
68      void move(const int& index, const double &x, const double &y);
69
70      // Obtiene el nombre del fondo actual
71      Glib::RefPtr<const Gdk::Pixbuf> getBackground() const;
72
73      // Establece el fondo especificado
74      void loadBackground(Glib::RefPtr<Gdk::Pixbuf> pixbuf);
75  };
76
77  #endif //WORMS_MAP_H
```

jun 12, 18 20:48 **ToolBoxView.cpp** Page 1/3

```
1
2   #include <gtkmm/builder.h>
3   #include <Path.h>
4   #include "ToolBoxView.h"
5
6   ToolBoxView::ToolBoxView(BaseObjectType *cobject,
7                            const Glib::RefPtr<Gtk::Builder> &builder)
8           : Gtk::Grid(cobject) {
9       processing=false;
10
11      builder->get_widget("tbtn_worm", worm);
12      worm->set_active(true);
13      builder->get_widget("tbtn_grd", girder_3m);
14      builder->get_widget("tbtn_grd6", girder_6m);
15
16      builder->get_widget("btn_move", move);
17      builder->get_widget("btn_undo", erase);
18      builder->get_widget("btn_turn_ccw", turnccw);
19      builder->get_widget("btn_turn_cw", turncw);
20      builder->get_widget("btn_bg", change_bg);
21      builder->get_widget("btn_mode", mode);
22      builder->get_widget("img_selected",selected);
23
24      worm->signal_clicked().connect(sigc::bind<int>
25              (sigc::mem_fun(*this, &ToolBoxView::onNewObjectClicked),
26               WORM_BUTTON_ID));
27      girder_3m->signal_clicked().connect(sigc::bind<int>
28              (sigc::mem_fun(*this, &ToolBoxView::onNewObjectClicked),
29               GIRDER_3_BUTTON_ID));
30
31      girder_6m->signal_clicked().connect(sigc::bind<int>
32              (sigc::mem_fun(*this, &ToolBoxView::onNewObjectClicked),
33               GIRDER_6_BUTTON_ID));
34  }
35
36  void ToolBoxView::bindController(MapController *controller) {
37      this->map_controller = controller;
38
39      erase->signal_clicked().connect(
40              sigc::mem_fun(*map_controller, &MapController::eraseSignal));
41
42      move->signal_clicked().connect(
43              sigc::mem_fun(*map_controller, &MapController::moveSignal));
44
45      turnccw->signal_clicked().connect(
46              sigc::mem_fun(*map_controller, &MapController::turnCCWSignal));
47
48      turncw->signal_clicked().connect(
49              sigc::mem_fun(*map_controller, &MapController::turnCWSignal));
50
51      change_bg->signal_clicked().connect(
52              sigc::mem_fun(*map_controller,
53                            &MapController::changeBackgroundSignal));
54
55      mode->signal_toggled().connect(
56              sigc::mem_fun(*this, &ToolBoxView::changeMode));
57  }
58
59  void ToolBoxView::onNewObjectClicked(unsigned id) {
60      if (!processing) {
61          processing=true;
62          if (id == WORM_BUTTON_ID) {
63              if (worm->get_active()) {
64                  girder_3m->set_active(false);
65                  girder_6m->set_active(false);
66              }
```

```
67          } else if (id == GIRDER_3_BUTTON_ID) {
68              if (girder_3m->get_active()) {
69                  worm->set_active(false);
70                  girder_6m->set_active(false);
71              }
72          } else {
73              girder_3m->set_active(false);
74              worm->set_active(false);
75          }
76          disableMovingItems();
77          mode->set_active(false);
78          map_controller->addModeSignal(id);
79          leaveConsistent();
80          processing=false;
81      }
82  }
83
84  void ToolBoxView::enableMovingItems() {
85      turncw->set_sensitive(true);
86      turnccw->set_sensitive(true);
87      move->set_sensitive(true);
88      erase->set_sensitive(true);
89  }
90
91  void ToolBoxView::disableMovingItems() {
92      turncw->set_sensitive(false);
93      turnccw->set_sensitive(false);
94      move->set_sensitive(false);
95      erase->set_sensitive(false);
96  }
97
98  void ToolBoxView::changeMode() {
99      worm->set_sensitive(!mode->get_active());
100     girder_3m->set_sensitive(!mode->get_active());
101     girder_6m->set_sensitive(!mode->get_active());
102     if (!mode->get_active()){
103         disableMovingItems();
104     }
105     map_controller->changeModeSignal();
106 }
107
108 void ToolBoxView::leaveConsistent() {
109     if (!worm->get_active() && !girder_6m->get_active() &&
110                                             !girder_3m->get_active()){
111         processing=true;
112         worm->set_active(true);
113         map_controller->addModeSignal(WORM_BUTTON_ID);
114     }
115 }
116
117 void ToolBoxView::showSelected(int id) {
118     switch (id){
119         case WORM_BUTTON_ID:
120             selected->set(IMAGES_PATH+"/right_worm.png");
121             selected->show();
122             hideRotatingButtons();
123             break;
124         case GIRDER_3_BUTTON_ID:
125             selected->set(IMAGES_PATH+"Girder/girder_3_selected.png");
126             selected->show();
127             break;
128         case GIRDER_6_BUTTON_ID:
129             selected->set(IMAGES_PATH+"Girder/girder_6_selected.png");
130             selected->show();
131             break;
132         default:
```



```
133             hideSelected();
134             break;
135     }
136 }
137
138 void ToolBoxView::hideSelected() {
139     selected->hide();
140 }
141
142 void ToolBoxView::closeSelectionMode() {
143     disableMovingItems();
144     hideSelected();
145     mode->set_active(false);
146 }
147
148 void ToolBoxView::hideRotatingButtons() {
149     turncw->set_sensitive(false);
150     turnccw->set_sensitive(false);
151 }
152
```

jun 19, 18 14:51 **ToolBoxView.h** Page 1/2

```
1
2   #ifndef WORMS_TOOLBOX_H
3   #define WORMS_TOOLBOX_H
4
5   #include <gtkmm/grid.h>
6   #include <gtkmm/button.h>
7   #include <gtkmm/layout.h>
8   #include <gtkmm/togglebutton.h>
9   #include <gtkmm/switch.h>
10  #include <gtkmm/hvbox.h>
11  #include "MapView.h"
12  #include "MapController.h"
13
14  #define WORM_BUTTON_ID 1
15  #define GIRDER_3_BUTTON_ID 3
16  #define GIRDER_6_BUTTON_ID 6
17  class MapController;
18
19  // Clase que contiene la vista de la botonera
20  class ToolBoxView : public Gtk::Grid {
21  private:
22      Gtk::Button *erase;
23      MapController *map_controller;
24      Gtk::ToggleButton *worm;
25      Gtk::ToggleButton *girder_3m;
26      Gtk::ToggleButton *girder_6m;
27      Gtk::Button *move;
28
29      Gtk::Button *turnccw;
30      Gtk::Button *turncw;
31      Gtk::Button *change_bg;
32      Gtk::ToggleButton *mode;
33      Gtk::Image* selected;
34      bool processing;
35
36      // Deja en un estado consistente la zona "Agregar"
37      void leaveConsistent();
38
39      // Deshabilita los botones de rotacion
40      void hideRotatingButtons();
41
42  public:
43      ToolBoxView(BaseObjectType *cobject,
44                  const Glib::RefPtr<Gtk::Builder> &builder);
45
46      // Se ejecuta cuando se selecciona un elemento de la zona "Agregar"
47      void onNewObjectClicked(unsigned int id);
48
49      // Habilita para el usuario la interacciÃ³n con las acciones de la zona
50      // "Seleccion"
51      void enableMovingItems();
52
53      // Deshabilita para el usuario la interacciÃ³n con las acciones de la zona
54      // "Seleccion"
55      void disableMovingItems();
56
57      // Enlaza la vista con el controlador
58      void bindController(MapController *controller);
59
60      // Alterna la vista entre el modo "Agregar" y modo "Seleccion"
61      void changeMode();
62
63      // Muestra el objeto seleccionado en el recuadro en la zona "Seleccion"
64      void showSelected(int id);
65
66      // VacÃ-a el recuadro en la zona "Seleccion"
```

jun 19, 18 14:51 **ToolBoxView.h** Page 2/2

```
67      void hideSelected();
68
69      // Sale del modo "Seleccion"
70      void closeSelectionMode();
71
72
73  };
74
75
76  #endif //WORMS_TOOLBOX_H
```

jun 10, 18 19:29 **WeaponView.cpp** Page 1/1

```cpp
#include "WeaponView.h"

WeaponView::WeaponView(const Glib::RefPtr<Gtk::Builder> &builder,
                       const unsigned int &id) {
    builder->get_widget("sc_wep" + std::to_string(id), ammo_selector);
    builder->get_widget("cb_wep" + std::to_string(id), infinite);

    default_checkbox_state = infinite->get_active();
    default_ammo_selector_value = ammo_selector->get_value();

    ammo_selector->set_sensitive(!default_checkbox_state);

    ammo_selector->signal_value_changed().connect(
            sigc::mem_fun(*this, &WeaponView::onAmmoValueChanged));

    infinite->signal_clicked().connect(
            sigc::mem_fun(*this, &WeaponView::onCheckboxClicked));
}

void WeaponView::onAmmoValueChanged() {
    controller->updateAmmo(ammo_selector->get_value());
}

void WeaponView::onCheckboxClicked() {
    ammo_selector->set_sensitive(!infinite->get_active());
    if (infinite->get_active()) {
        controller->updateAmmo(-1);
    } else {
        controller->updateAmmo(ammo_selector->get_value());
    }
}

void WeaponView::resetAmmo() {
    ammo_selector->set_sensitive(!default_checkbox_state);
    ammo_selector->set_value(default_ammo_selector_value);
    infinite->set_active(default_checkbox_state);
}

void WeaponView::bindController(WeaponController *controller) {
    this->controller = controller;
}

const int WeaponView::getInitialAmmo() {
    return default_checkbox_state ? -1 : default_ammo_selector_value;
}

void WeaponView::setAmmo(const int &ammo) {
    if (ammo < 0) {
        infinite->set_active(true);
        ammo_selector->set_sensitive(false);
    } else {
        infinite->set_active(false);
        ammo_selector->set_sensitive(true);
        ammo_selector->set_value(ammo);
    }
}
```

jun 07, 18 23:33 **WeaponView.h** Page 1/1

```cpp

#ifndef WORMS_WEP_H
#define WORMS_WEP_H


#include <gtkmm/hvbox.h>
#include <gtkmm/scale.h>
#include <gtkmm/checkbutton.h>
#include <gtkmm/builder.h>
#include "WeaponController.h"
class WeaponController;

// Clase que contiene la vista de cada arma
class WeaponView {
private:
    Gtk::Scale *ammo_selector;
    Gtk::CheckButton *infinite;
    bool default_checkbox_state;
    int default_ammo_selector_value;
    WeaponController *controller;

public:
    WeaponView(const Glib::RefPtr<Gtk::Builder> &builder,
               const unsigned int &id);

    // Al cambiar el valor del scale se llama a este mÃ©todo.
    void onAmmoValueChanged();

    // Al cambiar el estado del checkbox  se llama a este mÃ©todo.
    void onCheckboxClicked();

    // Muestra la municiÃ³n predeterminada de esta arma
    void resetAmmo();

    // Enlaza la vista al controlador
    void bindController(WeaponController *controller);

    // Obtiene la municiÃ³n inicial
    const int getInitialAmmo();

    // Establece la municiÃ³n a mostrar
    void setAmmo(const int &ammo);
};


#endif //WORMS_WEP_H
```

jun 24, 18 18:19 **Table of Content** Page 1/1

Table of Contents