```cpp

#include <Path.h>
#include <string>
#include <vector>
#include "FileBoxController.h"
#include "FileWriter.h"
#include "FileReader.h"
#include "InvalidMapError.h"

static const char *const NEW_FILE_NAME = "Sin titulo.yaml";

FileBoxController::FileBoxController(UsablesController &wep_controller,
        std::shared_ptr<MapController> map_controller,
        const Glib::RefPtr<Gtk::Builder> &builder )
        : usables_controller(wep_controller),
          map_controller(std::move(map_controller))
{
    builder->get_widget("save_dialog",save_dialog);
    save_dialog->add_button("Cancelar", Gtk::RESPONSE_CANCEL);
    save_dialog->add_button("Guardar", Gtk::RESPONSE_OK);

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

    builder->get_widget("open_dialog",open_dialog);
    open_dialog->add_button("Cancelar", Gtk::RESPONSE_CANCEL);
    open_dialog->add_button("Abrir", Gtk::RESPONSE_OK);
}

void FileBoxController::onSaveClicked() const {
    try {
        std::vector<std::vector<double>> worms;
        std::vector<std::vector<double>> girders;
        map_controller->getObjects(worms, girders);
        auto background = map_controller->getBackground();

        std::vector<int> weapons_ammo;
        unsigned int life;
        usables_controller.getWeaponsAndLife(weapons_ammo, life);

        save_dialog->set_current_folder(MAPS_PATH);
        save_dialog->set_current_name(map_name->get_text());
        int result = save_dialog->run();
        if (result==Gtk::RESPONSE_OK){
            std::string path = save_dialog->get_filename();
            std::string filename = save_dialog->get_current_name();
            map_name->set_label(filename);

            size_t extension = filename.rfind(".");
            std::string bg_name = filename.substr(0, extension) + ".png";
            background->save(BACKGROUND_PATH + bg_name, "png");

            FileWriter file(path);
            file.save(weapons_ammo, worms, girders, life, bg_name);
        }
        save_dialog->hide();
    } catch(const InvalidMapError &error){
        error.what();
    }
}

void FileBoxController::onLoadClicked() const {
    open_dialog->set_current_folder(MAPS_PATH);
    int result = open_dialog->run();
    if (result==Gtk::RESPONSE_OK) {
        std::string filename = open_dialog->get_filename();
        map_name->set_label(open_dialog->get_current_name());
```



```cpp

        std::vector<std::vector<double>> worms;
        std::vector<std::vector<double>> girders;
        std::vector<int> weps_ammo;
        unsigned int life;
        std::string background;

        FileReader file(filename);
        file.read(worms, girders,
                  weps_ammo, life, background);

        map_controller->loadBackground(background);
        usables_controller.loadWeapons(weps_ammo, life);
        map_controller->loadObjects(worms, girders);
    }
    open_dialog->hide();
}


void FileBoxController::onNewClicked() const {
    map_name->set_label(NEW_FILE_NAME);
    usables_controller.onResetSignal();
    map_controller->newMapSignal();
}

```

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 10, 18 19:29 **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 }
40
41 void MapController::newMapSignal() {
42     model.clean();
43     view->clean();
44     toolBox->closeSelectionMode();
45 }
46
47 void MapController::moveSignal() {
48     this->actual_mode = MOVE_CMD_ID;
49 }
50
51 void MapController::changeModeSignal() {
52     this->actual_mode = (actual_mode==ADD_MODE_ID? SELECT_MODE_ID:ADD_MODE_ID);
53     if (actual_mode==ADD_MODE_ID) toolBox->closeSelectionMode();
54 }
55
56 void MapController::turn(const int &rotation) {
57     if (model.isGirder(index_object_selected)) {
58         unsigned int id;
59         int new_angle = this->model.turn(index_object_selected, id, rotation);
60         this->view->turn(id, new_angle, index_object_selected);
61     }
62 }
63
64 void MapController::turnCCWSignal()  {
65     turn(10);
66 }
```

jun 10, 18 19:29 **MapController.cpp** Page 2/3

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

jun 10, 18 19:29 **MapController.cpp** Page 3/3

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

```
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(const std::string &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 10, 18 19:29 **FileReader.cpp** Page 1/1

```
1
2  #include "FileReader.h"
3  #include <map>
4  #include <string>
5  #include <vector>
6
7  FileReader::FileReader(const std::string &filename):
8      file(filename, std::fstream::in),
9      filename(filename) {}
10
11 void FileReader::read(std::vector<std::vector<double>> &worms,
12                       std::vector<std::vector<double>> &girders,
13                       std::vector<int> &weps_ammo,
14                       unsigned int &worms_life, std::string& background) {
15     YAML::Node config = YAML::LoadFile(filename);
16
17     background = config[BACKGROUND_IMAGE].as<std::string>();
18
19     worms_life = config[WORMS_LIFE].as<unsigned int>();
20
21     std::map<std::string, int> ammo =
22                     config[WEAPON_AMMO].as<std::map<std::string, int>>();
23
24     weps_ammo.push_back(ammo[BAZOOKA_NAME]);
25     weps_ammo.push_back(ammo[MORTAR_NAME]);
26     weps_ammo.push_back(ammo[GREEN_GRENADE_NAME]);
27     weps_ammo.push_back(ammo[RED_GRENADE_NAME]);
28     weps_ammo.push_back(ammo[BANANA_NAME]);
29     weps_ammo.push_back(ammo[AIR_ATTACK_NAME]);
30     weps_ammo.push_back(ammo[BAT_NAME]);
31     weps_ammo.push_back(ammo[TELEPORT_NAME]);
32     weps_ammo.push_back(ammo[DYNAMITE_NAME]);
33     weps_ammo.push_back(ammo[HOLY_GRENADE_NAME]);
34
35     worms = config[WORMS_DATA].as<std::vector<std::vector<double>>>();
36
37     girders = config[GIRDERS_DATA].as<std::vector<std::vector<double>>>();
38 }
```

jun 10, 18 19:29 **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
13 // Clase que se encarga de manejar la carga de un mapa
14 class FileReader{
15 private:
16     std::fstream file;
17     std::string filename;
18
19 public:
20     explicit FileReader(const std::string &filename);
21
22     // Carga todos los componentes de un mapa desde un archivo YAML
23     void read(std::vector<std::vector<double>> &worms,
24               std::vector<std::vector<double>> &girders,
25               std::vector<int> &weps_ammo,
26               unsigned int &worm_life, std::string& background);
27 };
28
29
30 #endif //WORMS_FILEREADER_H
```

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

```
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, const std::string& background) {
15     YAML::Emitter out;
16
17     out << YAML::BeginMap;
18
19     out << YAML::Key << BACKGROUND_IMAGE;
20     out << YAML::Value << background;
21
22     out << YAML::Key << WORMS_LIFE;
23     out << YAML::Value << worm_life;
24
25     out << YAML::Key << WEAPON_AMMO;
26
27     out << YAML::Value << YAML::BeginMap;
28
29     out << YAML::Key << BAZOOKA_NAME;
30     out << YAML::Value << weapons[0];
31     out << YAML::Key << MORTAR_NAME;
32     out << YAML::Value << weapons[1];
33     out << YAML::Key << GREEN_GRENADE_NAME;
34     out << YAML::Value << weapons[2];
35     out << YAML::Key << RED_GRENADE_NAME;
36     out << YAML::Value << weapons[3];
37     out << YAML::Key << BANANA_NAME;
38     out << YAML::Value << weapons[4];
39     out << YAML::Key << HOLY_GRENADE_NAME;
40     out << YAML::Value << weapons[9];
41     out << YAML::Key << DYNAMITE_NAME;
42     out << YAML::Value << weapons[8];
43     out << YAML::Key << BAT_NAME;
44     out << YAML::Value << weapons[6];
45     out << YAML::Key << AIR_ATTACK_NAME;
46     out << YAML::Value << weapons[5];
47     out << YAML::Key << TELEPORT_NAME;
48     out << YAML::Value << weapons[7];
49
50     out << YAML::EndMap;
51
52     out << YAML::Key << WORMS_DATA;
53     out << worms;
54
55     out << YAML::Key << GIRDERS_DATA;
56     out << girders;
57
58     out << YAML::EndMap;
59
60     file << out.c_str();
61 }
```

jun 10, 18 19:29 **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
13 // Clase que se encarga de manejar el guardado de un mapa
14 class FileWriter{
15 private:
16     std::fstream file;
17
18 public:
19     explicit FileWriter(const std::string &filename);
20
21     // Guarda todos los componentes de un mapa en un archivo YAML
22     void
23     save(std::vector<int> weapons,
24          const std::vector<std::vector<double>> &worms,
25          const std::vector<std::vector<double>> &girders,
26          const unsigned int &worm_life, const std::string& background);
27 };
28
29 #endif //WORMS_FILEWRITER_H
```

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

```
1
2  #include <gtkmm/enums.h>
3  #include <gtkmm/messagedialog.h>
4  #include "InvalidMapError.h"
5
6  InvalidMapError::InvalidMapError(const char *message) noexcept:
7      message(message){}
8
9  const char *InvalidMapError::what() const noexcept{
10     Gtk::Window dialog_window;
11     Gtk::MessageDialog dialog("Error al guardar archivo",
12                               false, Gtk::MESSAGE_WARNING);
13     dialog.set_transient_for(dialog_window);
14     dialog.set_secondary_text(message);
15     dialog.run();
16     return message;
17 }
18
19 InvalidMapError::~InvalidMapError() {
20 }
```

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

```
1
2  #ifndef WORMS_INVALIDMAP_H
3  #define WORMS_INVALIDMAP_H
4
5
6  #include <exception>
7
8  // Clase que se encarga de lanzar una excepcion
9  // cuando el mapa a guardar es invalido
10 class InvalidMapError : public std::exception{
11 private:
12     const char* message;
13
14 public:
15     explicit InvalidMapError(const char *message) noexcept;
16
17     virtual const char *what() const noexcept;
18
19     ~InvalidMapError();
20 };
21
22
23 #endif //WORMS_INVALIDMAP_H
```

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

```
1
2  #include <vector>
3  #include "Map.h"
4
5  void Map::erase(const int &index) {
6      if (!contained_objects.empty())
7          this->contained_objects.erase(contained_objects.begin() + index);
8  }
9
10 void Map::clean() {
11     this->contained_objects.clear();
12 }
13
14 void
15 Map::add(const unsigned int &id, const double &x,
16                          const double &y, const int &angle) {
17     MapObject new_object(x, y, angle);
18     contained_objects.emplace_back(std::make_pair(id, new_object));
19 }
20
21 void Map::move(const int &index, const double &x,const double &y) {
22     MapObject &object = contained_objects[index].second;
23     object.updatePosition(x, y);
24 }
25
26 const int Map::turn(const unsigned int &index,
27                                 unsigned int &id, const int &rotation) {
28     MapObject &object = contained_objects[index].second;
29     id = contained_objects[index].first;
30     return object.turn(rotation);
31 }
32
33 const bool Map::isGirder(int &index) const {
34     return (contained_objects[index].first > 1);
35 }
36
37 void Map::getObjects(std::vector<std::vector<double>> &worms,
38                      std::vector<std::vector<double>> &girders) const {
39     for (auto &object : contained_objects) {
40         float x, y;
41         object.second.getPosition(x, y);
42         if (object.first == 1) {
43             std::vector<double> position;
44             position.push_back(x);
45             position.push_back(y);
46             worms.push_back(position);
47         } else {
48             std::vector<double> data;
49             data.push_back(object.first);
50             data.push_back(x);
51             data.push_back(y);
52             data.push_back(object.second.getAngle());
53             girders.push_back(data);
54         }
55     }
56 }
57
58 const int Map::getItemID(const int &index) const{
59     return contained_objects[index].first;
60 }
```

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

```
1
2  #ifndef WORMS_MAPMODEL_H
3  #define WORMS_MAPMODEL_H
4
5
6  #include <utility>
7  #include <vector>
8  #include "MapObject.h"
9
10 // Clase que se encarga de modelar el mapa
11 class Map {
12     std::vector<std::pair<int, MapObject>> contained_objects;
13
14 public:
15     // Borra el objeto que se encuentra en la posicion index del vector
16     void erase(const int &index);
17
18     // Borra todos los objetos contenidos en el mapa
19     void clean();
20
21     // Agregar un objeto en la posicion (x,y)
22     void add(const unsigned int &id, const double &x, const double &y,
23              const int &angle = 0);
24
25     // Obtiene todos los objetos contenidos en el mapa separados por tipo
26     void getObjects(std::vector<std::vector<double>> &worms,
27                     std::vector<std::vector<double>> &girders) const;
28
29     // Mueve el objeto en la posicion index del vector hacia la posicion
30     // (x,y) del mapa
31     void move(const int &index, const double &x, const double &y);
32
33     // Devuelve verdadero si el objeto en la posicion index es una viga
34     const bool isGirder(int &index) const;
35
36     // Obtiene el tipo del objeto en la posicion index del vector
37     const int getItemID(const int &index) const;
38
39     // Gira el objeto en la posicion index del vector en un angulo indicado
40     const int
41     turn(const unsigned int &index, unsigned int &id, const int &rotation);
42 };
43
44
45 #endif //WORMS_MAPMODEL_H
```

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

```
1
2  #include <cstdlib>
3  #include "MapObject.h"
4
5  MapObject::MapObject(const float &x, const float &y, const int &angle) :
6          position(x,y), angle(angle) {}
7
8  void MapObject::updatePosition(const float &x, const float &y) {
9      position= Position(x,y);
10 }
11
12 int MapObject::turn(const int &rotation){
13     if (angle == 0)
14         angle = 180;
15     return angle = abs((angle+rotation)%180);
16 }
17
18 void MapObject::getPosition(float &x, float &y) const {
19     y=position.getY();
20     x=position.getX();
21 }
22
23 const int MapObject::getAngle() const {
24     return angle;
25 }
26
27
```

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

```
1
2  #ifndef WORMS_OBJECTMODEL_H
3  #define WORMS_OBJECTMODEL_H
4
5  #include <Position.h>
6
7  // Clase que modela un objeto contenido en el mapa
8  class MapObject {
9      Position position;
10     int angle;
11 public:
12     MapObject(const float &x, const float &y, const int &angle = 0);
13
14     // Actualiza la posicion en la que se encuentra el objeto
15     void updatePosition(const float &x, const float &y);
16
17     // Obtiene la posicion en la que se encuentra el objeto
18     void getPosition(float &x, float &y) const;
19
20     // Actualiza el angulo en la que se encuentra el objeto
21     const int getAngle() const;
22
23     // Gira el objeto la cantidad especificada
24     int turn(const int &rotation);
25 };
26
27
28 #endif //WORMS_OBJECTMODEL_H
```

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

```
1
2  #include "Weapon.h"
3
4  Weapon::Weapon(const int &default_ammo)
5          : default_ammo(default_ammo),
6            actual_ammo(default_ammo) {}
7
8  void Weapon::resetAmmo() {
9      actual_ammo = default_ammo;
10 }
11
12 void Weapon::setAmmo(const int &new_ammo) {
13     this->actual_ammo = new_ammo;
14 }
15
16 int Weapon::getAmmo() const {
17     return actual_ammo;
18 }
```

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

```
1
2  #ifndef WORMS_WEAPONMODEL_H
3  #define WORMS_WEAPONMODEL_H
4
5  // Clase que modela un arma
6  class Weapon {
7  private:
8      const int default_ammo;
9      int actual_ammo;
10 public:
11     explicit Weapon(const int &default_ammo);
12
13     // Establece el valor de la municion por defecto en el modelo
14     void resetAmmo();
15
16     // Establece el valor de la municion indicado en el modelo
17     void setAmmo(const int &new_ammo);
18
19     // Obtiene el valor actual de la municion
20     int getAmmo() const;
21 };
22
23
24 #endif //WORMS_WEAPONMODEL_H
```

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

```
1
2  #include "FileBoxView.h"
3
4  FileBoxView::FileBoxView(BaseObjectType *cobject,
5                           const Glib::RefPtr<Gtk::Builder> &builder)
6          : Gtk::Grid(cobject) {
7      builder->get_widget("btn_save", save);
8      builder->get_widget("btn_load", load);
9      builder->get_widget("btn_clean", new_map);
10 }
11
12 void FileBoxView::bindController(std::shared_ptr<FileBoxController> controller){
13     this->file_box_controller = std::move(controller);
14
15     save->signal_clicked().connect(
16             sigc::mem_fun(*file_box_controller,
17                           &FileBoxController::onSaveClicked));
18
19     load->signal_clicked().connect(
20             sigc::mem_fun(*file_box_controller,
21                           &FileBoxController::onLoadClicked));
22
23     new_map->signal_clicked().connect(
24             sigc::mem_fun(*file_box_controller,
25                           &FileBoxController::onNewClicked));
26 }
```

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

```
1
2  #ifndef WORMS_FILEBOXVIEW_H
3  #define WORMS_FILEBOXVIEW_H
4
5  #include <gtkmm/builder.h>
6  #include <gtkmm/hvbox.h>
7  #include <gtkmm/button.h>
8  #include <gtkmm/grid.h>
9  #include "FileBoxController.h"
10
11 class FileBoxController;
12
13 // Clase que se encarga de manipular la zona de archivos
14 class FileBoxView : public Gtk::Grid {
15 private:
16     Gtk::Button *save;
17     Gtk::Button *load;
18     Gtk::Button *new_map;
19     std::shared_ptr<FileBoxController> file_box_controller;
20 public:
21     FileBoxView(BaseObjectType *cobject,
22                 const Glib::RefPtr<Gtk::Builder> &builder);
23
24     // Enlaza el controlador a la vista
25     void bindController(std::shared_ptr<FileBoxController> controller);
26 };
27
28
29 #endif //WORMS_FILEBOXVIEW_H
```

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

```
1
2  #include "LifeView.h"
3
4  LifeView::LifeView(BaseObjectType *cobject,
5                     const Glib::RefPtr<Gtk::Builder> &builder)
6          : Gtk::SpinButton(cobject),
7            default_hp(this->get_value()) {
8  }
9
10 void LifeView::reset() {
11     this->set_value(default_hp);
12 }
13
14 void LifeView::update(const unsigned int &new_life) {
15     this->set_value(new_life);
16 }
```

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

```
1
2  #ifndef WORMS_LIFEVIEW_H
3  #define WORMS_LIFEVIEW_H
4
5
6  #include <gtkmm/spinbutton.h>
7  #include <gtkmm/builder.h>
8
9  // Clase que se encarga de manipular la vista de la vida
10 class LifeView : public Gtk::SpinButton {
11 private:
12     const unsigned int default_hp;
13 public:
14     LifeView(BaseObjectType *cobject,
15              const Glib::RefPtr<Gtk::Builder> &builder);
16
17     // Establece el valor por defecto de la vida
18     void reset();
19
20     // Establece un nuevo valor a mostrar en la vista de la vida
21     void update(const unsigned int &new_life);
22 };
23
24
25 #endif //WORMS_LIFEVIEW_H
```

jun 10, 18 19:29 **MapView.cpp** Page 1/3

```
1
2  #include <Path.h>
3  #include <gtkmm/adjustment.h>
4  #include <gtkmm/scrolledwindow.h>
5  #include <glibmm/main.h>
6  #include <vector>
7  #include <string>
8  #include "MapView.h"
9  #include "GirderSize.h"
10
11 const std::string DEFAULT_BACKGROUND("default_background.png");
12
13 MapView::MapView(BaseObjectType *cobject,
14                  const Glib::RefPtr<Gtk::Builder> &builder)
15         : Gtk::Layout(cobject),
16           scroll_handler(*(Gtk::ScrolledWindow*)this->get_parent()){
17     add_events(Gdk::BUTTON_PRESS_MASK);
18     signal_button_press_event().connect(
19             sigc::mem_fun(*this, &MapView::onButtonClicked));
20
21     setInitialPosition();
22     changeBackground(BACKGROUND_PATH + DEFAULT_BACKGROUND);
23     initializeWormsImages();
24     initializeGirderImages();
25 }
26
27 bool MapView::onButtonClicked(GdkEventButton *button_event) {
28     controller->mapClickedSignal(button_event);
29     return true;
30 }
31
32 void MapView::setInitialPosition() {
33     guint w, h;
34     get_size(w, h);
35     ((Gtk::ScrolledWindow*) get_parent())->get_hadjustment()->set_value(w / 2);
36     ((Gtk::ScrolledWindow*) get_parent())->get_vadjustment()->set_value(h);
37 }
38
39 void MapView::initializeGirderImages(){
40     std::vector<std::string> girder_3_imgs;
41     std::vector<std::string> girder_6_imgs;
42
43     for (int i = 0; i < 180; i = i + 10) {
44         girder_3_imgs.emplace_back(
45                 GIRDER_PATH + "3_" + std::to_string(i) + ".png");
46         girder_6_imgs.push_back(
47                 GIRDER_PATH + "6_" + std::to_string(i) + ".png");
48     }
49     objects_pallete.push_back(girder_3_imgs);
50     objects_pallete.push_back(girder_6_imgs);
51 }
52
53 void MapView::initializeWormsImages()  {
54     std::vector<std::string> worms_imgs;
55     worms_imgs.emplace_back(IMAGES_PATH + "/right_worm.png");
56     objects_pallete.push_back(worms_imgs);
57 }
58
59 void MapView::add(const unsigned int &id, const double &x, const double &y,
60                   const int &angle) {
61     Gtk::Image new_image(objects_pallete[id - id / 2 - 1][0]);
62     const Glib::RefPtr<Gdk::Pixbuf> &img = new_image.get_pixbuf();
63     int width = img->get_width();
64     int height = img->get_height();
65     double x_bound = x - width / 2;
66     double y_bound = y - height / 2;
```

```
67
68      put(new_image, x_bound, y_bound);
69      new_image.show();
70      contained_objects.push_back(std::move(new_image));
71      if (angle > 0){
72          sigc::slot<bool> my_slot = sigc::bind(sigc::mem_fun(
73              *this, &MapView::turn), id, angle, contained_objects.size() - 1);
74          Glib::signal_idle().connect(my_slot);
75      }
76  }
77
78  void MapView::move(const int &index, const double &x, const double &y) {
79      if (!contained_objects.empty()) {
80          Gtk::Image &actual_object = contained_objects[index];
81          Gtk::Layout::move(actual_object, x - actual_object.get_width() / 2,
82                            y - actual_object.get_height() / 2);
83          actual_object.show();
84      }
85  }
86
87  bool MapView::turn(const unsigned int &id, const int &angle, const int &index) {
88      if (!contained_objects.empty()) {
89          Gtk::Image &image = contained_objects[index];
90          float x = child_property_x(image) + image.get_width() / 2;
91          float y = child_property_y(image) + image.get_height() / 2;
92          image.set(objects_pallete[id - id / 2 - 1][angle / 10]);
93
94          int height = GirderSize::getGirderHeightPixels(id, angle);
95          int width = GirderSize::getGirderWidthPixels(id, angle);
96          Gtk::Layout::move(image, x - width / 2, y - height / 2);
97      }
98      return false;
99  }
100
101 void MapView::erase(const int &index) {
102     if (!contained_objects.empty()) {
103         contained_objects[index].hide();
104         contained_objects.erase(contained_objects.begin() + index);
105     }
106 }
107
108 void MapView::clean() {
109     contained_objects.clear();
110 }
111
112 void MapView::bindController(MapController *map_controller) {
113     this->controller = map_controller;
114 }
115
116 void MapView::changeBackground(const std::string &path) {
117     background.clear();
118     Gtk::Image bg(path);
119     int img_width = bg.get_pixbuf()->get_width();
120     int img_height = bg.get_pixbuf()->get_height();
121     guint window_width, window_height;
122     this->get_size(window_width, window_height);
123     for (size_t x = 0; x < window_width; x += img_width) {
124         for (size_t y = 0; y < window_height; y += img_height) {
125             Gtk::Image image(path);
126             image.show();
127             put(image, x, y);
128             background.push_back(std::move(image));
129         }
130     }
131     redrawMap();
132 }
```



```
133
134 void MapView::redrawMap() {
135     for (Gtk::Image &object : contained_objects){
136         const Gtk::Allocation &alloc = object.get_allocation();
137         remove(object);
138         put(object,alloc.get_x(),alloc.get_y());
139     }
140     this->water.show(*this);
141 }
142
143 int MapView::select(const double &x, const double &y) {
144     Gdk::Rectangle new_object(x, y, 1, 1);
145     for (ssize_t i = contained_objects.size() - 1; i >= 0; i--) {
146         bool collision = contained_objects[i].intersect(new_object);
147         if (collision) {
148             return i;
149         }
150     }
151     return -1;
152 }
153
154 Glib::RefPtr<const Gdk::Pixbuf> MapView::getBackground() const{
155     return this->background[0].get_pixbuf();
156 }
157
158 void MapView::loadBackground(const std::string &name) {
159     changeBackground(BACKGROUND_PATH + name);
160 }
161
```

```
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      // Establece la posicion actual del mapa a mostrar
34      void setInitialPosition();
35
36      // Dibuja nuevamente el contenido del mapa
37      void redrawMap();
38
39  public:
40      MapView(BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);
41
42      // Se ejecuta al clickear el mapa
43      bool onButtonClicked(GdkEventButton *button_event);
44
45      // Borra el objeto en la posiciÃ³n indicada
46      void erase(const int &index);
47
48      // Elimina todo el contenido del mapa
49      void clean();
50
51      // Enlaza el controlador a la vista
52      void bindController(MapController *map_controller);
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
```



```
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 por su nombre
74      void loadBackground(const std::string &name);
75  };
76
77  #endif //WORMS_MAP_H
```

jun 10, 18 19:29 **ToolBoxView.cpp** Page 1/3

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

ToolBoxView::ToolBoxView(BaseObjectType *cobject,
                         const Glib::RefPtr<Gtk::Builder> &builder)
        : Gtk::Grid(cobject) {
    processing=false;

    builder->get_widget("tbtn_worm", worm);
    worm->set_active(true);
    builder->get_widget("tbtn_grd", girder_3m);
    builder->get_widget("tbtn_grd6", girder_6m);

    builder->get_widget("btn_move", move);
    builder->get_widget("btn_undo", erase);
    builder->get_widget("btn_turn_ccw", turnccw);
    builder->get_widget("btn_turn_cw", turncw);
    builder->get_widget("btn_bg", change_bg);
    builder->get_widget("btn_mode", mode);
    builder->get_widget("img_selected",selected);

    worm->signal_clicked().connect(sigc::bind<int>
            (sigc::mem_fun(*this, &ToolBoxView::onNewObjectClicked),
             WORM_BUTTON_ID));
    girder_3m->signal_clicked().connect(sigc::bind<int>
            (sigc::mem_fun(*this, &ToolBoxView::onNewObjectClicked),
             GIRDER_3_BUTTON_ID));

    girder_6m->signal_clicked().connect(sigc::bind<int>
            (sigc::mem_fun(*this, &ToolBoxView::onNewObjectClicked),
             GIRDER_6_BUTTON_ID));
}

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

    erase->signal_clicked().connect(
            sigc::mem_fun(*map_controller, &MapController::eraseSignal));

    move->signal_clicked().connect(
            sigc::mem_fun(*map_controller, &MapController::moveSignal));

    turnccw->signal_clicked().connect(
            sigc::mem_fun(*map_controller, &MapController::turnCCWSignal));

    turncw->signal_clicked().connect(
            sigc::mem_fun(*map_controller, &MapController::turnCWSignal));

    change_bg->signal_clicked().connect(
            sigc::mem_fun(*map_controller,
                          &MapController::changeBackgroundSignal));

    mode->signal_toggled().connect(
            sigc::mem_fun(*this, &ToolBoxView::changeMode));
}

void ToolBoxView::onNewObjectClicked(unsigned id) {
    if (!processing) {
        processing=true;
        if (id == WORM_BUTTON_ID) {
            if (worm->get_active()) {
                girder_3m->set_active(false);
                girder_6m->set_active(false);
            }
        } else if (id == GIRDER_3_BUTTON_ID) {
            if (girder_3m->get_active()) {
                worm->set_active(false);
                girder_6m->set_active(false);
            }
        } else {
            girder_3m->set_active(false);
            worm->set_active(false);
        }
        disableMovingItems();
        mode->set_active(false);
        map_controller->addModeSignal(id);
        leaveConsistent();
        processing=false;
    }
}

void ToolBoxView::enableMovingItems() {
    turncw->set_sensitive(true);
    turnccw->set_sensitive(true);
    move->set_sensitive(true);
    erase->set_sensitive(true);
}

void ToolBoxView::disableMovingItems() {
    turncw->set_sensitive(false);
    turnccw->set_sensitive(false);
    move->set_sensitive(false);
    erase->set_sensitive(false);
}

void ToolBoxView::changeMode() {
    worm->set_sensitive(!mode->get_active());
    girder_3m->set_sensitive(!mode->get_active());
    girder_6m->set_sensitive(!mode->get_active());
    if (!mode->get_active()){
        disableMovingItems();
    }
    map_controller->changeModeSignal();
}

void ToolBoxView::leaveConsistent() {
    if (!worm->get_active() && !girder_6m->get_active() &&
                                                !girder_3m->get_active()){
        processing=true;
        worm->set_active(true);
        map_controller->addModeSignal(WORM_BUTTON_ID);
    }
}

void ToolBoxView::showSelected(int id) {
    switch (id){
        case WORM_BUTTON_ID:
            selected->set(IMAGES_PATH+"/right_worm.png");
            selected->show();
            break;
        case GIRDER_3_BUTTON_ID:
            selected->set(IMAGES_PATH+"Girder/girder_3_selected.png");
            selected->show();
            break;
        case GIRDER_6_BUTTON_ID:
            selected->set(IMAGES_PATH+"Girder/girder_6_selected.png");
            selected->show();
            break;
        default:
            hideSelected();
```

jun 10, 18 19:29 **ToolBoxView.cpp** Page 3/3

```
133                 break;
134         }
135 }
136
137 void ToolBoxView::hideSelected() {
138     selected->hide();
139 }
140
141 void ToolBoxView::closeSelectionMode() {
142     disableMovingItems();
143     hideSelected();
144     mode->set_active(false);
145 }
146
```

jun 07, 18 23:33 **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 public:
40     ToolBoxView(BaseObjectType *cobject,
41                 const Glib::RefPtr<Gtk::Builder> &builder);
42
43     // Se ejecuta cuando se selecciona un elemento de la zona "Agregar"
44     void onNewObjectClicked(unsigned int id);
45
46     // Habilita para el usuario la interacciÃ³n con las acciones de la zona
47     // "Seleccion"
48     void enableMovingItems();
49
50     // Deshabilita para el usuario la interacciÃ³n con las acciones de la zona
51     // "Seleccion"
52     void disableMovingItems();
53
54     // Enlaza la vista con el controlador
55     void bindController(MapController *controller);
56
57     // Alterna la vista entre el modo "Agregar" y modo "Seleccion"
58     void changeMode();
59
60     // Muestra el objeto seleccionado en el recuadro en la zona "Seleccion"
61     void showSelected(int id);
62
63     // VacÃ-a el recuadro en la zona "Seleccion"
64     void hideSelected();
65
66     // Sale del modo "Seleccion"
```

jun 07, 18 23:33 **ToolBoxView.h** Page 2/2

```
67      void closeSelectionMode();
68  };
69
70
71  #endif //WORMS_TOOLBOX_H
```

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

```
1   #include "WeaponView.h"
2
3   WeaponView::WeaponView(const Glib::RefPtr<Gtk::Builder> &builder,
4                          const unsigned int &id) {
5       builder->get_widget("sc_wep" + std::to_string(id), ammo_selector);
6       builder->get_widget("cb_wep" + std::to_string(id), infinite);
7
8       default_checkbox_state = infinite->get_active();
9       default_ammo_selector_value = ammo_selector->get_value();
10
11      ammo_selector->set_sensitive(!default_checkbox_state);
12
13      ammo_selector->signal_value_changed().connect(
14              sigc::mem_fun(*this, &WeaponView::onAmmoValueChanged));
15
16      infinite->signal_clicked().connect(
17              sigc::mem_fun(*this, &WeaponView::onCheckboxClicked));
18  }
19
20  void WeaponView::onAmmoValueChanged() {
21      controller->updateAmmo(ammo_selector->get_value());
22  }
23
24  void WeaponView::onCheckboxClicked() {
25      ammo_selector->set_sensitive(!infinite->get_active());
26      if (infinite->get_active()) {
27          controller->updateAmmo(-1);
28      } else {
29          controller->updateAmmo(ammo_selector->get_value());
30      }
31  }
32
33  void WeaponView::resetAmmo() {
34      ammo_selector->set_sensitive(!default_checkbox_state);
35      ammo_selector->set_value(default_ammo_selector_value);
36      infinite->set_active(default_checkbox_state);
37  }
38
39  void WeaponView::bindController(WeaponController *controller) {
40      this->controller = controller;
41  }
42
43  const int WeaponView::getInitialAmmo() {
44      return default_checkbox_state ? -1 : default_ammo_selector_value;
45  }
46
47  void WeaponView::setAmmo(const int &ammo) {
48      if (ammo < 0) {
49          infinite->set_active(true);
50          ammo_selector->set_sensitive(false);
51      } else {
52          infinite->set_active(false);
53          ammo_selector->set_sensitive(true);
54          ammo_selector->set_value(ammo);
55      }
56  }
```

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

```
1
2  #ifndef WORMS_WEP_H
3  #define WORMS_WEP_H
4
5
6  #include <gtkmm/hvbox.h>
7  #include <gtkmm/scale.h>
8  #include <gtkmm/checkbutton.h>
9  #include <gtkmm/builder.h>
10 #include "WeaponController.h"
11 class WeaponController;
12
13 // Clase que contiene la vista de cada arma
14 class WeaponView {
15 private:
16     Gtk::Scale *ammo_selector;
17     Gtk::CheckButton *infinite;
18     bool default_checkbox_state;
19     int default_ammo_selector_value;
20     WeaponController *controller;
21
22 public:
23     WeaponView(const Glib::RefPtr<Gtk::Builder> &builder,
24                const unsigned int &id);
25
26     // Al cambiar el valor del scale se llama a este mÃ©todo.
27     void onAmmoValueChanged();
28
29     // Al cambiar el estado del checkbox  se llama a este mÃ©todo.
30     void onCheckboxClicked();
31
32     // Muestra la municiÃ³n predeterminada de esta arma
33     void resetAmmo();
34
35     // Enlaza la vista al controlador
36     void bindController(WeaponController *controller);
37
38     // Obtiene la municiÃ³n inicial
39     const int getInitialAmmo();
40
41     // Establece la municiÃ³n a mostrar
42     void setAmmo(const int &ammo);
43 };
44
45
46 #endif //WORMS_WEP_H
```

jun 11, 18 22:24 **Table of Content** Page 1/1