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



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

Jun 03, 18 12:56 **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  class FileBoxController {
11  private:
12      UsablesController &usables_controller;
13      std::shared_ptr<MapController> map_controller;
14      Gtk::FileChooserDialog* save_dialog;
15      Gtk::FileChooserDialog* open_dialog;
16      Gtk::Label* map_name;
17
18  public:
19      FileBoxController(UsablesController &wep_controller,
20                        std::shared_ptr<MapController> map_controller,
21                        const Glib::RefPtr<Gtk::Builder> &builder);
22
23      void onSaveClicked() const;
24
25      void onLoadClicked() const;
26
27      void onNewClicked() const;
28  };
29
30
31  #endif //WORMS_FILECONTROLLER_H
```

Jun 05, 18 15:21 **MapController.cpp** Page 1/3

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

```
67      if (actual_mode == MOVE_CMD_ID) {
68          this->model.move(index_object_selected, event_button->x,
69                           event_button->y);
70          this->view->move(index_object_selected, event_button->x,
71                           event_button->y);
72          actual_mode = SELECT_MODE_ID;
73      } else if (actual_mode == SELECT_MODE_ID) {
74          this->index_object_selected = view->select(event_button->x,
75                                                     event_button->y);
76          if (index_object_selected > -1) {
77              toolBox->enableMovingItems();
78              toolBox->showSelected(model.getItemID(index_object_selected));
79          } else {
80              toolBox->disableMovingItems();
81              toolBox->hideSelected();
82          }
83          actual_mode = SELECT_MODE_ID; //cambio de estado del toolbox llama a add
     mode
84      } else {
85          this->model.add(item_id_to_add, event_button->x, event_button->y);
86          this->view->add(item_id_to_add, event_button->x, event_button->y);
87      }
88  }
89
90  void MapController::getObjects(std::vector<std::vector<double>> &worms,
91                                 std::vector<std::vector<double>> &girders) const
    {
92      model.getObjects(worms, girders);
93      if (worms.empty()){
94          throw InvalidMapError("El mapa actual no contiene worms");
95      }
96      if (girders.empty()){
97          throw InvalidMapError("El mapa actual no contiene vigas");
98      }
99
100     ViewPositionTransformer transformer(*view);
101     for (std::vector<double> &worm : worms){
102         Position position(worm[0],worm[1]);
103         Position new_pos = transformer.transformToPosition(position);
104         worm[0] = new_pos.getX();
105         worm[1] = new_pos.getY();
106     }
107     for (std::vector<double> &girder : girders){
108         Position position(girder[1],girder[2]);
109         Position new_pos = transformer.transformToPosition(position);
110         girder[1] = new_pos.getX();
111         girder[2] = new_pos.getY();
112     }
113 }
114
115 void MapController::loadObjects(std::vector<std::vector<double>> &worms,
116                                 std::vector<std::vector<double>> &girders) {
117     newMapSignal();
118     ViewPositionTransformer transformer(*view);
119     for (std::vector<double> &worm:worms) {
120         Position position(worm[0],worm[1]);
121         Position new_pos = transformer.transformToScreen(position);
122         worm[0] = new_pos.getX();
123         worm[1] = new_pos.getY();
124         this->model.add(1, worm[0], worm[1]);
125         this->view->add(1, worm[0], worm[1]);
126     }
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 }
136
137 void MapController::changeBackgroundSignal() const {
138     this->view->changeBackground();
139 }
140
141 const std::string MapController::getBackgroundName() const{
142     return view->getBackgroundName();
143 }
144
145 void MapController::loadBackground(const std::string &background)  {
146     view->loadBackground(background);
147 }
```

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

```
1
2  #ifndef WORMS_MAPCONTROLLER_H
3  #define WORMS_MAPCONTROLLER_H
4
5  #include "MapView.h"
6  #include "Map.h"
7  #include "ToolBoxView.h"
8
9  class MapView;
10 class ToolBoxView;
11
12 class MapController {
13     Map model;
14     MapView *view;
15     ToolBoxView *toolBox;
16     unsigned int item_id_to_add;
17     unsigned int actual_mode;
18     int index_object_selected;
19
20     void turn(const int &rotation);
21 public:
22     MapController(Map model, const Glib::RefPtr<Gtk::Builder> &builder);
23
24     void addModeSignal(const unsigned int &id);
25
26     void eraseSignal();
27
28     void newMapSignal();
29
30     void moveSignal();
31
32     void turnCCWSignal();
33
34     void mapClickedSignal(GdkEventButton *event_button);
35
36     void getObjects(std::vector<std::vector<double>> &worms,
37                     std::vector<std::vector<double>> &girders) const;
38
39     void loadObjects(std::vector<std::vector<double>> &worms,
40                      std::vector<std::vector<double>> &girders);
41
42     void turnCWSignal();
43
44     void changeBackgroundSignal() const;
45
46     void changeModeSignal();
47
48     const std::string getBackgroundName() const;
49
50     void loadBackground(const std::string &background);
51 };
52
53
54 #endif //WORMS_MAPCONTROLLER_H
```

Jun 05, 18 14:07 **UsablesController.cpp** Page 1/2

```
1
2  #include "UsablesController.h"
3  #include "InvalidMapError.h"
4
5  UsablesController::UsablesController(
6          const Glib::RefPtr<Gtk::Builder> &builder) {
7      builder->get_widget("btn_reset", reset_button);
8      reset_button->signal_clicked().connect(
9              sigc::mem_fun(*this,
10                           &UsablesController::onResetSignal));
11
12     builder->get_widget_derived("life", life_spinner);
13
14     for (size_t i = 1; i <= 10; ++i) {
15         std::shared_ptr<WeaponView> weapon_view(new WeaponView(builder, i));
16
17         std::shared_ptr<Weapon> weapon
18                 (new Weapon(weapon_view->getInitialAmmo()));
19
20         weapons.push_back(weapon);
21
22         std::shared_ptr<WeaponController> weapon_controller(
23                 new WeaponController(weapon_view,
24                                      weapon));
25         wep_controllers.push_back(std::move(weapon_controller));
26         weapons_view.push_back(weapon_view);
27     }
28 }
29
30 void UsablesController::onResetSignal() {
31     life_spinner->reset();
32     for (const std::shared_ptr<WeaponController> &actual_controller:wep_controll
   ers) {
33         actual_controller->resetAmmo();
34     }
35 }
36
37 void UsablesController::getWeaponsAndLife(std::vector<int> &weps_ammo,
38                                           unsigned int &life) const {
39     life = life_spinner->get_value();
40     for (const std::shared_ptr<WeaponController> &actual_controller:wep_controll
   ers) {
41         weps_ammo.push_back(actual_controller->getAmmo());
42     }
43     if (!isValidWeaponSet(weps_ammo)){
44         throw InvalidMapError("NingÃºn arma tiene municiÃ³n");
45     }
46 }
47
48 void UsablesController::loadWeapons(std::vector<int> &weps_ammo,
49                                     const unsigned int &life) const {
50     int i = 0;
51     for (const std::shared_ptr<WeaponController> &actual_controller
52             :wep_controllers) {
53         actual_controller->updateAmmo(weps_ammo[i]);
54         i++;
55     }
56     life_spinner->update(life);
57 }
58
59 bool
60 UsablesController::isValidWeaponSet(std::vector<int> &ammo_vector) const {
61     for (int actual_ammo : ammo_vector) {
62         if(actual_ammo !=0)
63             return true;
64     }
```

Jun 05, 18 14:07 **UsablesController.cpp** Page 2/2

```
65      return false;
66  }
```

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

```
1
2   #ifndef WORMS_WEAPONSLISTCONTROLLER_H
3   #define WORMS_WEAPONSLISTCONTROLLER_H
4
5
6   #include <gtkmm/button.h>
7   #include <gtkmm/spinbutton.h>
8   #include "Weapon.h"
9   #include "WeaponView.h"
10  #include "LifeView.h"
11
12  class UsablesController {
13  private:
14      LifeView *life_spinner;
15      Gtk::Button *reset_button;
16      std::vector<std::shared_ptr<Weapon>> weapons;
17      std::vector<std::shared_ptr<WeaponView>> weapons_view;
18      std::vector<std::shared_ptr<WeaponController> > wep_controllers;
19
20      bool isValidWeaponSet(std::vector<int> &ammo_vector) const;
21  public:
22      explicit UsablesController(
23              const Glib::RefPtr<Gtk::Builder> &builder);
24
25      void onResetSignal();
26
27      void getWeaponsAndLife(std::vector<int> &weps_ammo, unsigned int &life) cons
    t;
28
29      void
30      loadWeapons(std::vector<int> &weps_ammo, const unsigned int &life) const;
31
32
33  };
34
35
36  #endif //WORMS_WEAPONSLISTCONTROLLER_H
```

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

```cpp

#ifndef WORMS_WEAPONCONTROLLER_H
#define WORMS_WEAPONCONTROLLER_H

#include "WeaponView.h"
#include "Weapon.h"

class WeaponView;

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);

    void resetAmmo();

    void updateAmmo(const int &ammo);

    int getAmmo();
};


#endif //WORMS_WEAPONCONTROLLER_H
```

Jun 01, 18 13:12 **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 03, 18 12:56 **Editor.cpp** Page 1/1

```cpp

#include "Editor.h"

Editor::Editor(BaseObjectType *cobject,
               const Glib::RefPtr<Gtk::Builder> &builder)
        : Gtk::Window(cobject),
          weps_list_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(weps_list_controller, map_controller,builder))
;
    filebox->bindController(filebox_controller);

    show_all_children();
}
```

Jun 03, 18 12:56 **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
16 class Editor : public Gtk::Window {
17     Gtk::ScrolledWindow *map_window;
18     Map map_model;
19     UsablesController weps_list_controller;
20     FileBoxView *filebox;
21
22 public:
23     Editor(BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);
24
25 };
26
27
28 #endif //WORMS_EDITOR_H
```

Jun 02, 18 18:24 **FileReader.cpp** Page 1/1

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

Jun 02, 18 13:44 **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
11 class FileReader{
12 private:
13     std::fstream file;
14     std::string filename;
15 public:
16
17     explicit FileReader(const std::string &filename);
18
19
20     void read(std::vector<std::vector<double>> &worms,
21               std::vector<std::vector<double>> &girders,
22               std::vector<int> &weps_ammo,
23               unsigned int &worm_life, std::string& background);
24 };
25
26
27 #endif //WORMS_FILEREADER_H
```

Jun 02, 18 18:24 **FileWriter.cpp** Page 1/1

```
1
2
3  #include "FileWriter.h"
4
5  FileWriter::FileWriter(const std::string &filename)
6          : file(filename, std::fstream::out | std::ios_base::trunc) {}
7
8
9  void FileWriter::save(std::vector<int> weapons,
10                        const std::vector<std::vector<double>> &worms,
11                        const std::vector<std::vector<double>> &girders,
12                        const unsigned int &worm_life, const std::string& backgrou
   nd) {
13     YAML::Emitter out;
14
15     out << YAML::BeginMap;
16
17     out << YAML::Key << BACKGROUND_IMAGE;
18     out << YAML::Value << background;
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::EndMap;
57
58     file << out.c_str();
59 }
```

Jun 02, 18 13:44 **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
11 class FileWriter{
12 private:
13     std::fstream file;
14 public:
15     explicit FileWriter(const std::string &filename);
16
17     void
18     save(std::vector<int> weapons,
19          const std::vector<std::vector<double>> &worms,
20          const std::vector<std::vector<double>> &girders,
21          const unsigned int &worm_life, const std::string& background);
22 };
23
24
25 #endif //WORMS_FILEWRITER_H
```

Jun 02, 18 18:24 **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 : message(message
   ){
7  }
8
9  const char *InvalidMapError::what() const noexcept{
10     Gtk::Window dialog_window;
11     Gtk::MessageDialog dialog("Error al guardar archivo", false,Gtk::MESSAGE_WARNING);
12     dialog.set_transient_for(dialog_window);
13     dialog.set_secondary_text(message);
14     dialog.run();
15     return message;
16 }
17
18 InvalidMapError::~InvalidMapError() {
19 }
```

Jun 02, 18 13:44 **InvalidMapError.h** Page 1/1

```
1
2  #ifndef WORMS_INVALIDMAP_H
3  #define WORMS_INVALIDMAP_H
4
5
6  #include <exception>
7
8  class InvalidMapError : public std::exception{
9  private:
10     const char* message;
11 public:
12     InvalidMapError(const char *message) noexcept;
13
14     virtual const char *what() const noexcept;
15
16     ~InvalidMapError() override;
17 };
18
19
20 #endif //WORMS_INVALIDMAP_H
```

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

```
1
2  #include <yaml.h>
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, const double &y, const int &an
   gle) {
16     MapObject new_object(x, y, angle);
17     contained_objects.emplace_back(std::make_pair(id, new_object));
18 }
19
20 void Map::move(const int &index, const double &x,const double &y) {
21     MapObject &object = contained_objects[index].second;
22     object.updatePosition(x, y);
23 }
24
25 const int Map::turn(const unsigned int &index, unsigned int &id, const int &rota
   tion) {
26     MapObject &object = contained_objects[index].second;
27     id = contained_objects[index].first;
28     return object.turn(rotation);
29 }
30
31 const bool Map::isGirder(int &index) const {
32     return (contained_objects[index].first > 1);
33 }
34
35 void Map::getObjects(std::vector<std::vector<double>> &worms,
36                      std::vector<std::vector<double>> &girders) const {
37     for (auto &object : contained_objects) {
38         float x, y;
39         object.second.getPosition(x, y);
40         if (object.first == 1) {
41             std::vector<double> position;
42             position.push_back(x);
43             position.push_back(y);
44             worms.push_back(position);
45         } else {
46             std::vector<double> data;
47             data.push_back(object.first);
48             data.push_back(x);
49             data.push_back(y);
50             data.push_back(object.second.getAngle());
51             girders.push_back(data);
52         }
53     }
54 }
55
56 const int Map::getItemID(const int &index) const{
57     return contained_objects[index].first ;
58 }
59
60
61
```

Jun 05, 18 14:07 **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 class Map {
11     std::vector<std::pair<int, MapObject>> contained_objects;
12
13 public:
14     void erase(const int &index);
15
16     void clean();
17
18     void add(const unsigned int &id, const double &x, const double &y,
19              const int &angle = 0);
20
21
22     void getObjects(std::vector<std::vector<double>> &worms,
23                     std::vector<std::vector<double>> &girders) const;
24
25     void move(const int &index, const double &x, const double &y);
26
27     const bool isGirder(int &index) const;
28
29     const int getItemID(const int &index) const;
30
31     const int
32     turn(const unsigned int &index, unsigned int &id, const int &rotation);
33 };
34
35
36 #endif //WORMS_MAPMODEL_H
```

Jun 02, 18 18:24 **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 02, 18 18:24 **MapObject.h** Page 1/1

```
1
2  #ifndef WORMS_OBJECTMODEL_H
3  #define WORMS_OBJECTMODEL_H
4
5  #include <Position.h>
6
7  class MapObject {
8      Position position;
9      int angle;
10 public:
11     MapObject(const float &x, const float &y, const int &angle = 0);
12
13     void updatePosition(const float &x, const float &y);
14
15     void getPosition(float &x, float &y) const;
16
17     const int getAngle() const;
18
19     int turn(const int &rotation);
20 };
21
22
23 #endif //WORMS_OBJECTMODEL_H
```

Jun 02, 18 18:24 **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 02, 18 18:24 **Weapon.h** Page 1/1

```
1
2  #ifndef WORMS_WEAPONMODEL_H
3  #define WORMS_WEAPONMODEL_H
4
5  class Weapon {
6  private:
7      const int default_ammo;
8      int actual_ammo;
9  public:
10     explicit Weapon(const int &default_ammo);
11
12     void resetAmmo();
13
14     void setAmmo(const int &new_ammo);
15
16     int getAmmo() const;
17 };
18
19
20 #endif //WORMS_WEAPONMODEL_H
```

Jun 03, 18 12:56 **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 03, 18 12:56 **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 class FileBoxView : public Gtk::Grid {
14 private:
15     Gtk::Button *save;
16     Gtk::Button *load;
17     Gtk::Button *new_map;
18     std::shared_ptr<FileBoxController> file_box_controller;
19 public:
20     FileBoxView(BaseObjectType *cobject,
21                 const Glib::RefPtr<Gtk::Builder> &builder);
22
23     void bindController(std::shared_ptr<FileBoxController> controller);
24 };
25
26
27 #endif //WORMS_FILEBOXVIEW_H
```

Jun 02, 18 13:44 **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 02, 18 13:44 **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  class LifeView : public Gtk::SpinButton {
10 private:
11     const unsigned int default_hp;
12 public:
13     LifeView(BaseObjectType *cobject,
14              const Glib::RefPtr<Gtk::Builder> &builder);
15
16     void reset();
17
18     void update(const unsigned int &new_life);
19 };
20
21
22 #endif //WORMS_LIFEVIEW_H
```

Jun 05, 18 14:07 **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 "MapView.h"
7  #include "GirderSize.h"
8
9  #define BACKGROUND_QUANTITY 8
10
11 MapView::MapView(BaseObjectType *cobject,
12                  const Glib::RefPtr<Gtk::Builder> &builder)
13         : Gtk::Layout(cobject),
14           scroll_handler(*(Gtk::ScrolledWindow*)this->get_parent()), actual_back
   ground_index(0) {
15
16     add_events(Gdk::BUTTON_PRESS_MASK);
17     signal_button_press_event().connect(
18             sigc::mem_fun(*this, &MapView::onButtonClicked));
19
20     setInitialPosition();
21     initializeBackground();
22     initializeWormsImages();
23     initializeGirderImages();
24 }
25
26 bool MapView::onButtonClicked(GdkEventButton *button_event) {
27     controller->mapClickedSignal(button_event);
28     return true;
29 }
30
31 void MapView::setInitialPosition() {
32     guint width, height;
33     get_size(width, height);
34     ((Gtk::ScrolledWindow*) get_parent())->get_hadjustment()->set_value(width /
   2);
35     ((Gtk::ScrolledWindow*) get_parent())->get_vadjustment()->set_value(height);
36 }
37
38 void MapView::initializeBackground() {
39     for (size_t i = 0; i < BACKGROUND_QUANTITY; i++){
40         bg_paths.emplace_back(BACKGROUND_PATH + "background" + std::to_string(i) +
    ".jpg");
41     }
42     setBackground(bg_paths[actual_background_index]);
43 }
44
45 void MapView::initializeGirderImages(){
46     std::vector<std::string> girder_3_imgs;
47     std::vector<std::string> girder_6_imgs;
48
49     for (int i = 0; i < 180; i = i + 10) {
50         girder_3_imgs.emplace_back(
51                 GIRDER_PATH + "3_" + std::to_string(i) +
52                 ".png");
53         girder_6_imgs.push_back(
54                 GIRDER_PATH + "6_" + std::to_string(i) +
55                 ".png");
56     }
57     objects_pallete.push_back(girder_3_imgs);
58     objects_pallete.push_back(girder_6_imgs);
59 }
60
61 void MapView::initializeWormsImages()  {
62     std::vector<std::string> worms_imgs;
63     worms_imgs.emplace_back(IMAGES_PATH + "/right_worm.png");
```

```
64      objects_pallete.push_back(worms_imgs);
65  }
66
67  void MapView::add(const unsigned int &id, const double &x, const double &y,
68                    const int &angle) {
69      Gtk::Image new_image(objects_pallete[id - id / 2 - 1][0]);
70      const Glib::RefPtr<Gdk::Pixbuf> &img = new_image.get_pixbuf();
71      int width = img->get_width();
72      int height = img->get_height();
73      double x_bound = x - width / 2;
74      double y_bound = y - height / 2;
75
76      put(new_image, x_bound, y_bound);
77      new_image.show();
78      contained_objects.push_back(std::move(new_image));
79      if (angle > 0){
80          sigc::slot<bool> my_slot = sigc::bind(sigc::mem_fun(*this, &MapView::tur
    n), id, angle, contained_objects.size()-1);
81          Glib::signal_idle().connect(my_slot);
82      }
83  }
84
85  void MapView::move(const int &index, const double &x, const double &y) {
86      if (!contained_objects.empty()) {
87          Gtk::Image &actual_object = contained_objects[index];
88          Gtk::Layout::move(actual_object, x - actual_object.get_width() / 2,
89                            y - actual_object.get_height() / 2);
90          actual_object.show();
91      }
92  }
93
94  bool MapView::turn(const unsigned int &id, const int &angle, const int &index) {
95      if (!contained_objects.empty()) {
96          Gtk::Image &image = contained_objects[index];
97          float x = child_property_x(image) + image.get_width() / 2;
98          float y = child_property_y(image) + image.get_height() / 2;
99          image.set(objects_pallete[id - id / 2 - 1][angle / 10]);
100
101         int height = GirderSize::getGirderHeightPixels(id, angle);
102         int width = GirderSize::getGirderWidthPixels(id, angle);
103         Gtk::Layout::move(image, x - width / 2, y - height / 2);
104     }
105     return false;
106 }
107
108 void MapView::erase(const int &index) {
109     if (!contained_objects.empty()) {
110         contained_objects[index].hide();
111         contained_objects.erase(contained_objects.begin() + index);
112     }
113 }
114
115 void MapView::clean() {
116     contained_objects.clear();
117 }
118
119 void MapView::bindController(MapController *map_controller) {
120     this->controller = map_controller;
121 }
122
123 void MapView::changeBackground() {
124     background.clear();
125     actual_background_index = (actual_background_index + 1) % bg_paths.size();
126     setBackground(bg_paths[actual_background_index]);
127 }
128
```



```
129 void MapView::setBackground(const std::string &name) {
130     Gtk::Image bg(name);
131     int img_width = bg.get_pixbuf()->get_width();
132     int img_height = bg.get_pixbuf()->get_height();
133     guint window_width, window_height;
134     this->get_size(window_width, window_height);
135     for (size_t x = 0; x < window_width; x += img_width) {
136         for (size_t y = 0; y < window_height; y += img_height) {
137             Gtk::Image image(name);
138             image.show();
139             put(image, x, y);
140             background.push_back(std::move(image));
141         }
142     }
143     redrawMap();
144 }
145
146 void MapView::redrawMap() {
147     for(Gtk::Image &object : contained_objects){
148         const Gtk::Allocation &alloc = object.get_allocation();
149         remove(object);
150         put(object,alloc.get_x(),alloc.get_y());
151     }
152     this->water.show(*this);
153 }
154
155 int MapView::select(const double &x, const double &y) {
156     Gdk::Rectangle new_object(x, y, 1, 1);
157     for (ssize_t i = contained_objects.size() - 1; i >= 0; i--) {
158         bool collision = contained_objects[i].intersect(new_object);
159         if (collision) {
160             return i;
161         }
162     }
163     return -1;
164 }
165
166 const std::string MapView::getBackgroundName() const {
167     return "background"+std::to_string(actual_background_index)+".jpg";
168 }
169
170 void MapView::loadBackground(const std::string &name) {
171     background.clear();
172     setBackground(BACKGROUND_PATH+name);
173     const char &number = *(name.end() - 5);
174     actual_background_index=atoi(&number);
175 }
176
```

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

```
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 "MapController.h"
9  #include "Water.h"
10 #include "ScrollHandler.h"
11
12 class MapController;
13
14 class MapView : public Gtk::Layout {
15 private:
16     std::vector<Gtk::Image> contained_objects;
17     std::vector<std::vector<std::string>> objects_pallete;
18     MapController *controller;
19     std::vector<std::string> bg_paths;
20     std::vector<Gtk::Image> background;
21     Water water;
22     ScrollHandler scroll_handler;
23
24     int actual_background_index;
25
26     void initializeWormsImages();
27
28     void initializeGirderImages();
29
30     void setBackground(const std::string &name);
31
32     void initializeBackground();
33
34     void setInitialPosition();
35
36     void redrawMap();
37
38 public:
39     MapView(BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);
40
41     bool onButtonClicked(GdkEventButton *button_event);
42
43     void erase(const int &index);
44
45     void clean();
46
47     void bindController(MapController *map_controller);
48
49     void add(const unsigned int &id, const double &x, const double &y,
50              const int &angle = 0);
51
52     bool turn(const unsigned int &id, const int &angle, const int &index);
53
54     void changeBackground();
55
56     int select(const double &x, const double &y);
57
58     void move(const int& index, const double &x, const double &y);
59
60     const std::string getBackgroundName() const;
61
62     void loadBackground(const std::string &name);
63
64 };
65
66 #endif //WORMS_MAP_H
```

Jun 03, 18 12:56 **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
109 void ToolBoxView::leaveConsistent() {
110     if (!worm->get_active() && !girder_6m->get_active() && !girder_3m->get_activ
    e()){
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             break;
123         case GIRDER_3_BUTTON_ID:
124             selected->set(IMAGES_PATH+"Girder/girder_3_selected.png");
125             selected->show();
126             break;
127         case GIRDER_6_BUTTON_ID:
128             selected->set(IMAGES_PATH+"Girder/girder_6_selected.png");
129             selected->show();
130             break;
131         default:
```



```
132             hideSelected();
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 03, 18 12:56 **ToolBoxView.h** Page 1/1

```
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
18 class MapController;
19
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     void leaveConsistent();
37
38 public:
39     ToolBoxView(BaseObjectType *cobject,
40                 const Glib::RefPtr<Gtk::Builder> &builder);
41
42     void onNewObjectClicked(unsigned int id);
43
44     void enableMovingItems();
45
46     void disableMovingItems();
47
48     void bindController(MapController *controller);
49
50     void changeMode();
51
52     void showSelected(int id);
53
54     void hideSelected();
55
56     void closeSelectionMode();
57 };
58
59
60 #endif //WORMS_TOOLBOX_H
```

Jun 03, 18 12:56 **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 void WeaponView::resetAmmo() {
33     ammo_selector->set_sensitive(!default_checkbox_state);
34     ammo_selector->set_value(default_ammo_selector_value);
35     infinite->set_active(default_checkbox_state);
36 }
37
38 void WeaponView::bindController(WeaponController *controller) {
39     this->controller = controller;
40 }
41
42 const int WeaponView::getInitialAmmo() {
43     return default_checkbox_state ? -1 : default_ammo_selector_value;
44 }
45
46 void WeaponView::setAmmo(const int &ammo) {
47     if (ammo < 0) {
48         infinite->set_active(true);
49         ammo_selector->set_sensitive(false);
50     } else {
51         infinite->set_active(false);
52         ammo_selector->set_sensitive(true);
53         ammo_selector->set_value(ammo);
54     }
55
56 }
```

Jun 03, 18 12:56 **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
12 class WeaponController;
13
14 class WeaponView {
15     Gtk::Scale *ammo_selector;
16     Gtk::CheckButton *infinite;
17     bool default_checkbox_state;
18     int default_ammo_selector_value;
19     WeaponController *controller;
20 public:
21     WeaponView(const Glib::RefPtr<Gtk::Builder> &builder,
22                const unsigned int &id);
23
24     void onAmmoValueChanged();
25
26     void onCheckboxClicked();
27
28     void resetAmmo();
29
30     void bindController(WeaponController *controller);
31
32     const int getInitialAmmo();
33
34     void setAmmo(const int &ammo);
35 };
36
37
38 #endif //WORMS_WEP_H
```

Jun 06, 18 22:31 **Table of Content** Page 1/1