```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();
}

```

```cpp

#ifndef WORMS_FILECONTROLLER_H
#define WORMS_FILECONTROLLER_H

#include <gtkmm/filechooserdialog.h>
#include "FileBoxView.h"
#include "UsablesController.h"
#include "MapController.h"

// Clase que se encarga de establecer una conexion entre la seccion de archivos
// y  el resto del programa
class FileBoxController {
private:
    UsablesController &usables_controller;
    std::shared_ptr<MapController> map_controller;
    Gtk::FileChooserDialog* save_dialog;
    Gtk::FileChooserDialog* open_dialog;
    Gtk::Label* map_name;

public:
    FileBoxController(UsablesController &wep_controller,
                      std::shared_ptr<MapController> map_controller,
                      const Glib::RefPtr<Gtk::Builder> &builder);

    // Se encarga de mostrar un cuadro de dialogo para seleccionar un archivo
    // cuando se eligio guardar en la vista
    void onSaveClicked() const;

    // Se encarga de mostrar un cuadro de dialogo para seleccionar un archivo
    // cuando se eligio cargar en la vista
    void onLoadClicked() const;

    // Crea un nuevo mapa y actualiza la informacion del nombre del mapa actual
    void onNewClicked() const;
};


#endif //WORMS_FILECONTROLLER_H
```

```cpp

#include <gtkmm/messagedialog.h>
#include <ViewPositionTransformer.h>
#include <vector>
#include <string>
#include "MapController.h"
#include "InvalidMapError.h"
#include "Path.h"

#define ADD_MODE_ID 0
#define MOVE_CMD_ID 1
#define SELECT_MODE_ID 2

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

MapController::MapController(Map model, Builder &builder):
        model(std::move(model)), item_id_to_add(1),
        actual_mode(ADD_MODE_ID){
    builder->get_widget_derived("map", view);
    builder->get_widget_derived("toolbox", toolBox);
    view->bindController(this);
    toolBox->bindController(this);

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

void MapController::addModeSignal(const unsigned int &id) {
    this->actual_mode = ADD_MODE_ID;
    this->item_id_to_add = id;
}

void MapController::eraseSignal() {
    model.erase(index_object_selected);
    view->erase(index_object_selected);
    toolBox->hideSelected();
    toolBox->disableMovingItems();
}

void MapController::newMapSignal() {
    model.clean();
    view->clean();
    toolBox->closeSelectionMode();
}

void MapController::moveSignal() {
    this->actual_mode = MOVE_CMD_ID;
}

void MapController::changeModeSignal() {
    this->actual_mode = (actual_mode==ADD_MODE_ID? SELECT_MODE_ID:ADD_MODE_ID);
    if (actual_mode==ADD_MODE_ID) toolBox->closeSelectionMode();
}

void MapController::turn(const int &rotation) {
    if (model.isGirder(index_object_selected)) {
        unsigned int id;
        int new_angle = this->model.turn(index_object_selected, id, rotation);
        this->view->turn(id, new_angle, index_object_selected);
    }
}

void MapController::turnCCWSignal()  {
    turn(10);
}
```

```cpp

void MapController::turnCWSignal()  {
    turn(-10);
}

void MapController::mapClickedSignal(GdkEventButton *event_button) {
    if (actual_mode == MOVE_CMD_ID) {
        this->model.move(index_object_selected, event_button->x,
                        event_button->y);
        this->view->move(index_object_selected, event_button->x,
                        event_button->y);
        actual_mode = SELECT_MODE_ID;
    } else if (actual_mode == SELECT_MODE_ID) {
        this->index_object_selected = view->select(event_button->x,
                                                    event_button->y);
        if (index_object_selected > -1) {
            toolBox->enableMovingItems();
            toolBox->showSelected(model.getItemID(index_object_selected));
        } else {
            toolBox->disableMovingItems();
            toolBox->hideSelected();
        }
        //cambio de estado del toolbox llama a add mode
        actual_mode = SELECT_MODE_ID;
    } else {
        this->model.add(item_id_to_add, event_button->x, event_button->y);
        this->view->add(item_id_to_add, event_button->x, event_button->y);
    }
}

void MapController::getObjects(std::vector<std::vector<double>> &worms,
                            std::vector<std::vector<double>> &girders) const{
    model.getObjects(worms, girders);
    if (worms.empty()){
        throw InvalidMapError("El mapa actual no contiene worms");
    }
    if (girders.empty()){
        throw InvalidMapError("El mapa actual no contiene vigas");
    }

    ViewPositionTransformer transformer(*view);
    for (std::vector<double> &worm : worms){
        Position position(worm[0],worm[1]);
        Position new_pos = transformer.transformToPosition(position);
        worm[0] = new_pos.getX();
        worm[1] = new_pos.getY();
    }
    for (std::vector<double> &girder : girders){
        Position position(girder[1],girder[2]);
        Position new_pos = transformer.transformToPosition(position);
        girder[1] = new_pos.getX();
        girder[2] = new_pos.getY();
    }
}

void MapController::loadObjects(std::vector<std::vector<double>> &worms,
                                std::vector<std::vector<double>> &girders) {
    newMapSignal();
    ViewPositionTransformer transformer(*view);
    for (std::vector<double> &worm:worms) {
        Position position(worm[0],worm[1]);
        Position new_pos = transformer.transformToScreen(position);
        worm[0] = new_pos.getX();
        worm[1] = new_pos.getY();
        this->model.add(1, worm[0], worm[1]);
        this->view->add(1, worm[0], worm[1]);
```

```cpp
    }
    for (std::vector<double> &girder:girders) {
        Position position(girder[1],girder[2]);
        Position new_pos = transformer.transformToScreen(position);
        girder[1] = new_pos.getX();
        girder[2] = new_pos.getY();
        this->model.add(girder[0], girder[1], girder[2], girder[3]);
        this->view->add(girder[0], girder[1], girder[2], girder[3]);
    }
}

void MapController::changeBackgroundSignal() const {
    this->background_dialog->set_current_folder(BACKGROUND_PATH);
    int result = this->background_dialog->run();
    if (result==Gtk::RESPONSE_OK) {
        std::string path = this->background_dialog->get_filename();
        this->view->changeBackground(path);
    }
    this->background_dialog->hide();
}

Glib::RefPtr<const Gdk::Pixbuf> MapController::getBackground() const{
    return view->getBackground();
}

void MapController::loadBackground(const std::string &background)  {
    view->loadBackground(background);
}
```

```cpp

#ifndef WORMS_MAPCONTROLLER_H
#define WORMS_MAPCONTROLLER_H

#include <gtkmm/filechooserdialog.h>
#include <string>
#include <vector>
#include "MapView.h"
#include "Map.h"
#include "ToolBoxView.h"

class MapView;
class ToolBoxView;

// Clase que se encarga de comunicar la vista con el modelo, y a su vez, se
// comunica con el resto del programa
class MapController {
    Map model;
    MapView *view;
    ToolBoxView *toolBox;
    unsigned int item_id_to_add;
    unsigned int actual_mode;
    int index_object_selected;
    Gtk::FileChooserDialog* background_dialog;

    void turn(const int &rotation);

public:
    /* Constructor */
    MapController(Map model, const Glib::RefPtr<Gtk::Builder> &builder);

    /* Cambia al modo de agregado, en donde el objeto
     * a agregar es el de id */
    void addModeSignal(const unsigned int &id);

    /* Envia una seÃ±al de borrado */
    void eraseSignal();

    /* Envia una seÃ±al de nuevo mapa */
    void newMapSignal();

    /* Envia una seÃ±al de movimiento */
    void moveSignal();

    /* Envia una seÃ±al de rotacion anti horario */
    void turnCCWSignal();

    /* Envia una seÃ±al de click sobre el mapa */
    void mapClickedSignal(GdkEventButton *event_button);

    /* Obtiene los objetos del mapa */
    void getObjects(std::vector<std::vector<double>> &worms,
                    std::vector<std::vector<double>> &girders) const;

    /* Carga los worms y las vigas en el mapa */
    void loadObjects(std::vector<std::vector<double>> &worms,
                     std::vector<std::vector<double>> &girders);

    /* Envia una seÃ±al de rotacion horaria */
    void turnCWSignal();

    /* Envia una seÃ±al de cambio de imagen de fondo */
    void changeBackgroundSignal() const;

    /* Envia una seÃ±al de cambio de modo */
    void changeModeSignal();
```

```cpp

    /* Devuelve la imagen de fondo */
    Glib::RefPtr<const Gdk::Pixbuf> getBackground() const;

    /* Carga la imagen de fondo */
    void loadBackground(const std::string &background);
};


#endif //WORMS_MAPCONTROLLER_H
```

```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;
}
```

```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

```

#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

```

#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
```

```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;
}
```

```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();
}
```

```cpp

#ifndef WORMS_EDITOR_H
#define WORMS_EDITOR_H

#include <gtkmm/builder.h>
#include <gtkmm/window.h>
#include <gtkmm/scrolledwindow.h>
#include <gtkmm/spinbutton.h>
#include "MapView.h"
#include "ToolBoxView.h"
#include "UsablesController.h"
#include "FileBoxController.h"
#include "FileBoxView.h"

class Editor : public Gtk::Window {
    Gtk::ScrolledWindow *map_window;
    Map map_model;
    UsablesController usables_controller;
    FileBoxView *filebox;

public:
    Editor(BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);
};

#endif //WORMS_EDITOR_H
```

```cpp

#include "FileReader.h"
#include <map>
#include <string>
#include <vector>

FileReader::FileReader(const std::string &filename):
    file(filename, std::fstream::in),
    filename(filename) {}

void FileReader::read(std::vector<std::vector<double>> &worms,
                      std::vector<std::vector<double>> &girders,
                      std::vector<int> &weps_ammo,
                      unsigned int &worms_life, std::string& background) {
    YAML::Node config = YAML::LoadFile(filename);

    background = config[BACKGROUND_IMAGE].as<std::string>();

    worms_life = config[WORMS_LIFE].as<unsigned int>();

    std::map<std::string, int> ammo =
                    config[WEAPON_AMMO].as<std::map<std::string, int>>();

    weps_ammo.push_back(ammo[BAZOOKA_NAME]);
    weps_ammo.push_back(ammo[MORTAR_NAME]);
    weps_ammo.push_back(ammo[GREEN_GRENADE_NAME]);
    weps_ammo.push_back(ammo[RED_GRENADE_NAME]);
    weps_ammo.push_back(ammo[BANANA_NAME]);
    weps_ammo.push_back(ammo[AIR_ATTACK_NAME]);
    weps_ammo.push_back(ammo[BAT_NAME]);
    weps_ammo.push_back(ammo[TELEPORT_NAME]);
    weps_ammo.push_back(ammo[DYNAMITE_NAME]);
    weps_ammo.push_back(ammo[HOLY_GRENADE_NAME]);

    worms = config[WORMS_DATA].as<std::vector<std::vector<double>>>();

    girders = config[GIRDERS_DATA].as<std::vector<std::vector<double>>>();
}
```

```

#ifndef WORMS_FILEREADER_H
#define WORMS_FILEREADER_H

#include <fstream>
#include "MapObject.h"
#include <yaml.h>
#include <WeaponNames.h>
#include <ConfigFields.h>
#include <string>
#include <vector>

// Clase que se encarga de manejar la carga de un mapa
class FileReader{
private:
    std::fstream file;
    std::string filename;

public:
    explicit FileReader(const std::string &filename);

    // Carga todos los componentes de un mapa desde un archivo YAML
    void read(std::vector<std::vector<double>> &worms,
              std::vector<std::vector<double>> &girders,
              std::vector<int> &weps_ammo,
              unsigned int &worm_life, std::string& background);
};


#endif //WORMS_FILEREADER_H
```

```


#include "FileWriter.h"
#include <string>
#include <vector>

FileWriter::FileWriter(const std::string &filename):
    file(filename, std::fstream::out | std::ios_base::trunc) {}


void FileWriter::save(std::vector<int> weapons,
                const std::vector<std::vector<double>> &worms,
                const std::vector<std::vector<double>> &girders,
                const unsigned int &worm_life, const std::string& background) {
    YAML::Emitter out;

    out << YAML::BeginMap;

    out << YAML::Key << BACKGROUND_IMAGE;
    out << YAML::Value << background;

    out << YAML::Key << WORMS_LIFE;
    out << YAML::Value << worm_life;

    out << YAML::Key << WEAPON_AMMO;

    out << YAML::Value << YAML::BeginMap;

    out << YAML::Key << BAZOOKA_NAME;
    out << YAML::Value << weapons[0];
    out << YAML::Key << MORTAR_NAME;
    out << YAML::Value << weapons[1];
    out << YAML::Key << GREEN_GRENADE_NAME;
    out << YAML::Value << weapons[2];
    out << YAML::Key << RED_GRENADE_NAME;
    out << YAML::Value << weapons[3];
    out << YAML::Key << BANANA_NAME;
    out << YAML::Value << weapons[4];
    out << YAML::Key << HOLY_GRENADE_NAME;
    out << YAML::Value << weapons[9];
    out << YAML::Key << DYNAMITE_NAME;
    out << YAML::Value << weapons[8];
    out << YAML::Key << BAT_NAME;
    out << YAML::Value << weapons[6];
    out << YAML::Key << AIR_ATTACK_NAME;
    out << YAML::Value << weapons[5];
    out << YAML::Key << TELEPORT_NAME;
    out << YAML::Value << weapons[7];

    out << YAML::EndMap;

    out << YAML::Key << WORMS_DATA;
    out << worms;

    out << YAML::Key << GIRDERS_DATA;
    out << girders;

    out << YAML::EndMap;

    file << out.c_str();
}
```

```cpp

#ifndef WORMS_FILEWRITER_H
#define WORMS_FILEWRITER_H

#include <fstream>
#include "MapObject.h"
#include <yaml.h>
#include <WeaponNames.h>
#include <ConfigFields.h>
#include <vector>
#include <string>

// Clase que se encarga de manejar el guardado de un mapa
class FileWriter{
private:
    std::fstream file;

public:
    explicit FileWriter(const std::string &filename);

    // Guarda todos los componentes de un mapa en un archivo YAML
    void
    save(std::vector<int> weapons,
         const std::vector<std::vector<double>> &worms,
         const std::vector<std::vector<double>> &girders,
         const unsigned int &worm_life, const std::string& background);
};

#endif //WORMS_FILEWRITER_H
```

```cpp

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

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

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

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

```cpp

#ifndef WORMS_INVALIDMAP_H
#define WORMS_INVALIDMAP_H


#include <exception>

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

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

    virtual const char *what() const noexcept;

    ~InvalidMapError();
};


#endif //WORMS_INVALIDMAP_H
```

```cpp

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

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

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

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

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

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

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

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

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

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

```cpp

#ifndef WORMS_MAPMODEL_H
#define WORMS_MAPMODEL_H


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

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

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

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

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

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

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

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

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

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


#endif //WORMS_MAPMODEL_H
```

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

```cpp

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

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

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

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

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

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

```

```cpp

#ifndef WORMS_OBJECTMODEL_H
#define WORMS_OBJECTMODEL_H

#include <Position.h>

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

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

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

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

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


#endif //WORMS_OBJECTMODEL_H
```

```cpp

#include "Weapon.h"

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

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

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

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

```

#ifndef WORMS_WEAPONMODEL_H
#define WORMS_WEAPONMODEL_H

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

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

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

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


#endif //WORMS_WEAPONMODEL_H
```

```

#include "FileBoxView.h"

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

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

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

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

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

```

#ifndef WORMS_FILEBOXVIEW_H
#define WORMS_FILEBOXVIEW_H

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

class FileBoxController;

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

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


#endif //WORMS_FILEBOXVIEW_H
```

```

#include "LifeView.h"

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

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

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

```

#ifndef WORMS_LIFEVIEW_H
#define WORMS_LIFEVIEW_H


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

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

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

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


#endif //WORMS_LIFEVIEW_H
```

```

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

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

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

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

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

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

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

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

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

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

```cpp

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

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

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

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

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

void MapView::clean() {
    contained_objects.clear();
}

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

void MapView::changeBackground(const std::string &path) {
    background.clear();
    Gtk::Image bg(path);
    int img_width = bg.get_pixbuf()->get_width();
    int img_height = bg.get_pixbuf()->get_height();
    guint window_width, window_height;
    this->get_size(window_width, window_height);
    for (size_t x = 0; x < window_width; x += img_width) {
        for (size_t y = 0; y < window_height; y += img_height) {
            Gtk::Image image(path);
            image.show();
            put(image, x, y);
            background.push_back(std::move(image));
        }
    }
    redrawMap();
}
```

```cpp

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

int MapView::select(const double &x, const double &y) {
    Gdk::Rectangle new_object(x, y, 1, 1);
    for (ssize_t i = contained_objects.size() - 1; i >= 0; i--) {
        bool collision = contained_objects[i].intersect(new_object);
        if (collision) {
            return i;
        }
    }
    return -1;
}

Glib::RefPtr<const Gdk::Pixbuf> MapView::getBackground() const{
    return this->background[0].get_pixbuf();
}

void MapView::loadBackground(const std::string &name) {
    changeBackground(BACKGROUND_PATH + name);
}

```

```cpp

#ifndef WORMS_MAP_H
#define WORMS_MAP_H

#include <gtkmm/builder.h>
#include <gtkmm/layout.h>
#include <gtkmm/image.h>
#include <string>
#include <vector>
#include "MapController.h"
#include "Water.h"
#include "ScrollHandler.h"

class MapController;

// Clase que se encarga de manipular la vista del mapa
class MapView : public Gtk::Layout {
private:
    std::vector<Gtk::Image> contained_objects;
    std::vector<std::vector<std::string>> objects_pallete;
    MapController *controller;
    std::vector<Gtk::Image> background;
    Water water;
    ScrollHandler scroll_handler;


    // Inicializa el vector de imagenes de los worms
    void initializeWormsImages();

    // Inicializa el vector de imagenes de las vigas
    void initializeGirderImages();

    // Establece la posicion actual del mapa a mostrar
    void setInitialPosition();

    // Dibuja nuevamente el contenido del mapa
    void redrawMap();

public:
    MapView(BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);

    // Se ejecuta al clickear el mapa
    bool onButtonClicked(GdkEventButton *button_event);

    // Borra el objeto en la posiciÃ³n indicada
    void erase(const int &index);

    // Elimina todo el contenido del mapa
    void clean();

    // Enlaza el controlador a la vista
    void bindController(MapController *map_controller);

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

    // Gira el objeto seleccionado
    bool turn(const unsigned int &id, const int &angle, const int &index);

    // Cambia el fondo actual
    void changeBackground(const std::string &path);

    // Selecciona el objeto en la posiciÃ³n (x,y)
    int select(const double &x, const double &y);

```

```cpp
    // Mueve el objeto seleccionado a la posiciÃ³n (x.y)
    void move(const int& index, const double &x, const double &y);

    // Obtiene el nombre del fondo actual
    Glib::RefPtr<const Gdk::Pixbuf> getBackground() const;

    // Establece el fondo especificado por su nombre
    void loadBackground(const std::string &name);
};

#endif //WORMS_MAP_H
```

```cpp

#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);
            }
```

```cpp
        } 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();
            break;
    }
}

void ToolBoxView::hideSelected() {
    selected->hide();
}

void ToolBoxView::closeSelectionMode() {
    disableMovingItems();
    hideSelected();
    mode->set_active(false);
}

```

```cpp

#ifndef WORMS_TOOLBOX_H
#define WORMS_TOOLBOX_H

#include <gtkmm/grid.h>
#include <gtkmm/button.h>
#include <gtkmm/layout.h>
#include <gtkmm/togglebutton.h>
#include <gtkmm/switch.h>
#include <gtkmm/hvbox.h>
#include "MapView.h"
#include "MapController.h"

#define WORM_BUTTON_ID 1
#define GIRDER_3_BUTTON_ID 3
#define GIRDER_6_BUTTON_ID 6
class MapController;

// Clase que contiene la vista de la botonera
class ToolBoxView : public Gtk::Grid {
private:
    Gtk::Button *erase;
    MapController *map_controller;
    Gtk::ToggleButton *worm;
    Gtk::ToggleButton *girder_3m;
    Gtk::ToggleButton *girder_6m;
    Gtk::Button *move;

    Gtk::Button *turnccw;
    Gtk::Button *turncw;
    Gtk::Button *change_bg;
    Gtk::ToggleButton *mode;
    Gtk::Image* selected;
    bool processing;

    // Deja en un estado consistente la zona "Agregar"
    void leaveConsistent();

public:
    ToolBoxView(BaseObjectType *cobject,
                const Glib::RefPtr<Gtk::Builder> &builder);

    // Se ejecuta cuando se selecciona un elemento de la zona "Agregar"
    void onNewObjectClicked(unsigned int id);

    // Habilita para el usuario la interacciÃ³n con las acciones de la zona
    // "Seleccion"
    void enableMovingItems();

    // Deshabilita para el usuario la interacciÃ³n con las acciones de la zona
    // "Seleccion"
    void disableMovingItems();

    // Enlaza la vista con el controlador
    void bindController(MapController *controller);

    // Alterna la vista entre el modo "Agregar" y modo "Seleccion"
    void changeMode();

    // Muestra el objeto seleccionado en el recuadro en la zona "Seleccion"
    void showSelected(int id);

    // VacÃa el recuadro en la zona "Seleccion"
    void hideSelected();

    // Sale del modo "Seleccion"
    void closeSelectionMode();
};


#endif //WORMS_TOOLBOX_H
```

```cpp
#include "WeaponView.h"

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

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

    ammo_selector->set_sensitive(!default_checkbox_state);

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

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

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

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

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

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

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

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

```

#ifndef WORMS_WEP_H
#define WORMS_WEP_H


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

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

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

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

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

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

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

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

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


#endif //WORMS_WEP_H
```

Table of Contents

```
 1 FileBoxController.cpp sheets   1 to   1 ( 1) pages   1-  2   92 lines
 2 FileBoxController.h. sheets   2 to   2 ( 1) pages   3-  3   39 lines
 3 MapController.cpp... sheets   2 to   3 ( 2) pages   4-  6  161 lines
 4 MapController.h..... sheets   4 to   4 ( 1) pages   7-  8   77 lines
 5 UsablesController.cpp sheets   5 to   5 ( 1) pages   9-  9   65 lines
 6 UsablesController.h. sheets   5 to   5 ( 1) pages  10- 10   42 lines
 7 WeaponController.cpp sheets   6 to   6 ( 1) pages  11- 11   24 lines
 8 WeaponController.h.. sheets   6 to   6 ( 1) pages  12- 12   33 lines
 9 main.cpp............ sheets   7 to   7 ( 1) pages  13- 13   39 lines
10 Editor.cpp.......... sheets   7 to   7 ( 1) pages  14- 14   22 lines
11 Editor.h............ sheets   8 to   8 ( 1) pages  15- 15   26 lines
12 FileReader.cpp...... sheets   8 to   8 ( 1) pages  16- 16   39 lines
13 FileReader.h........ sheets   9 to   9 ( 1) pages  17- 17   31 lines
14 FileWriter.cpp...... sheets   9 to   9 ( 1) pages  18- 18   62 lines
15 FileWriter.h........ sheets  10 to  10 ( 1) pages  19- 19   30 lines
16 InvalidMapError.cpp. sheets  10 to  10 ( 1) pages  20- 20   21 lines
17 InvalidMapError.h... sheets  11 to  11 ( 1) pages  21- 21   24 lines
18 Map.cpp............. sheets  11 to  11 ( 1) pages  22- 22   61 lines
19 Map.h............... sheets  12 to  12 ( 1) pages  23- 23   46 lines
20 MapObject.cpp....... sheets  12 to  12 ( 1) pages  24- 24   28 lines
21 MapObject.h......... sheets  13 to  13 ( 1) pages  25- 25   29 lines
22 Weapon.cpp.......... sheets  13 to  13 ( 1) pages  26- 26   19 lines
23 Weapon.h............ sheets  14 to  14 ( 1) pages  27- 27   25 lines
24 FileBoxView.cpp..... sheets  14 to  14 ( 1) pages  28- 28   27 lines
25 FileBoxView.h....... sheets  15 to  15 ( 1) pages  29- 29   30 lines
26 LifeView.cpp........ sheets  15 to  15 ( 1) pages  30- 30   17 lines
27 LifeView.h.......... sheets  16 to  16 ( 1) pages  31- 31   26 lines
28 MapView.cpp......... sheets  16 to  17 ( 2) pages  32- 34  162 lines
29 MapView.h........... sheets  18 to  18 ( 1) pages  35- 36   78 lines
30 ToolBoxView.cpp..... sheets  19 to  20 ( 2) pages  37- 39  147 lines
31 ToolBoxView.h....... sheets  20 to  21 ( 2) pages  40- 41   72 lines
32 WeaponView.cpp...... sheets  21 to  21 ( 1) pages  42- 42   57 lines
33 WeaponView.h........ sheets  22 to  22 ( 1) pages  43- 43   47 lines
```