```
#include <iostream>
#include <string>
#include <SDL2/SDL.h>
#include <QApplication>
#include <QMessageBox>
#include <QDebug>
#include "editor.h"
#include "yaml.h"
#include "map_game.h"
#include "yaml.h"
#include "inventory.h"
#include "inventory_editor.h"

#define EXIT_PADDING 5
#define EXIT_ICON_SIDE 20
#define SAVE_PADDING 10
#define SAVE_ICON_SIDE 60

Editor::Editor(YAML::Node map, std::string mn, std::string bgn, std::string bgp)
 :
bg_name(bgn),
bg_path(bgp),
map_name(mn),
mapNode(YAML::Clone(map)),
staticNode(mapNode["static"]),
mapGame(mapNode),
editorWindow(staticNode, 0, 0, true, true),
camera(editorWindow.getScreenWidth(),
        editorWindow.getScreenHeight(),
        editorWindow.getBgWidth(),
        editorWindow.getBgHeight()),
renderer(editorWindow.getRenderer()),
editorInventory(renderer,
             mapNode["static"]["teams_amount"].as<int>(),
             mapNode["static"]["worms_health"].as<int>()) {
        this->teamsAmount = mapNode["static"]["teams_amount"].as<int>();
        this->wormsHealth = mapNode["static"]["worms_health"].as<int>();
  this->editorInventory.toggleOpen();
        this->mapGame.setRenderer(this->renderer);
        this->mapGame.initializeStates();
  this->mapGame.createMapToSave();
        this->exitTexture.loadFromFile(gPath.PATH_EXIT_ICON, this->renderer);
        this->exitTexture.setX(this->editorWindow.getScreenWidth() - EXIT_PADDIN
G - EXIT_ICON_SIDE);
        this->exitTexture.setY(EXIT_PADDING);
        this->saveTexture.loadFromFile(gPath.PATH_SAVE_ICON, this->renderer);
        this->saveTexture.setX(this->editorWindow.getScreenWidth() - SAVE_PADDIN
G - SAVE_ICON_SIDE);
        this->saveTexture.setY(EXIT_PADDING + EXIT_ICON_SIDE + SAVE_PADDING);
        this->unsaved_changes = false;
        this->notice.setScreenWidth(this->editorWindow.getScreenWidth());
        this->notice.setScreenHeight(this->editorWindow.getScreenHeight());
}

int Editor::start(void) {
  bool quit = false;
        SDL_Event e;
        while (!quit) {
                int camX = camera.getX(), camY = camera.getY();

                while (SDL_PollEvent(&e) != 0) {
                        if (e.type == SDL_QUIT) {
```

```
                                quit = true;
                                editorWindow.hide();
                                validMap = mapGame.hasWorms();
                                if (!validMap) {
                    QMessageBox msgBox;
                    msgBox.setWindowTitle("Mapa invÃ¡lido.");
                    msgBox.setText("El mapa debe tener al menos un worm de cada team."  "Â¿Dese
a continuar editando el mapa?");
                    msgBox.setStandardButtons(QMessageBox::Yes);
                    msgBox.addButton(QMessageBox::No);
                    msgBox.setDefaultButton(QMessageBox::Yes);
                    if(msgBox.exec() == QMessageBox::Yes) {
                        editorWindow.show();
                                        quit = false;
                    }
                                }
                        }

                        if (e.type == SDL_KEYDOWN) {

                                        if (e.key.keysym.sym == SDLK_z && (e.key
.keysym.mod & KMOD_CTRL)) {
                                                mapGame.setPreviousState(editorI
nventory);
                                        }

                                        if (e.key.keysym.sym == SDLK_y && (e.key
.keysym.mod & KMOD_CTRL)) {
                                                mapGame.setNextState(editorInven
tory);
                                        }

                        }

                        if (e.type == SDL_MOUSEBUTTONDOWN) {
                                int mouseX, mouseY;
                        SDL_GetMouseState(&mouseX, &mouseY);
                                if (e.button.button == SDL_BUTTON_LEFT) {
                                        if (
                                                mouseX > this->saveTexture.getX(
) &&
                                                mouseX < this->saveTexture.getX(
) + SAVE_ICON_SIDE &&
                                                mouseY > this->saveTexture.getY(
) &&
                                                mouseY < this->saveTexture.getY(
) + SAVE_ICON_SIDE
                                        ) {
                                                if (!this->unsaved_changes) {
                                                        std::cout << "No hay cambi
os sin guardar." << std::endl;
                                                        this->notice.showFlashNo
tice(this->renderer, "No hay cambios sin guardar.");
                                                        continue;

                                                }
                                                validMap = mapGame.hasWorms();
                                                if (!validMap) {
                                                        std::cout << "El mapa debe t
ener al menos un worm de cada team." << std::endl;
                                                        this->notice.showFlashEr
ror(this->renderer, "El mapa debe tener al menos un worm de cada team.");
```

```
                                                                      continue;
                                                            }
                                                            mapGame.saveAs(this->map_name, t
his->bg_name, this->bg_path);

                                                            this->unsaved_changes = false;
                                                            std::cout << "Mapa guardado." << st
d::endl;

                                                            this->notice.showFlashNotice(thi
s->renderer, "Mapa guardado en /usr/etc/worms/maps/" + this->map_name);
                                                  } else if (
                                                            mouseX > this->exitTexture.getX(
) &&
                                                            mouseX < this->exitTexture.getX(
) + EXIT_ICON_SIDE &&
                                                            mouseY > this->exitTexture.getY(
) &&
                                                            mouseY < this->exitTexture.getY(
) + EXIT_ICON_SIDE) {
                                                                      quit = true;
                                                                      editorWindow.hide();
                                                                      validMap = mapGame.hasWo
rms();

                                                                      if (!validMap) {
                                                                                QMessageBox msgB
ox;

                                                                                msgBox.setWindow
Title("Mapa invÃ¡lido.");

                                                                                msgBox.setText("
El mapa debe tener al menos un worm de cada team."  "Â¿Desea continuar editando el mapa?");
                                                                                msgBox.setStanda
rdButtons(QMessageBox::Yes);

                                                                                msgBox.addButton
(QMessageBox::No);

                                                                                msgBox.setDefaul
tButton(QMessageBox::Yes);

                                                                                if(msgBox.exec()
 == QMessageBox::Yes) {

editorWindow.show();

quit = false;

continue;
                                                                                }
                                                                      }
                                                                      if (this->unsaved_change
s) {
                                                                                QMessageBox msgB
ox;

                                                                                msgBox.setWindow
Title("Guardar antes de salir.");

                                                                                msgBox.setText("
Hay cambios sin guardar. Desea guardar el mapa antes de salir?");
                                                                                msgBox.setStanda
rdButtons(QMessageBox::Yes);

                                                                                msgBox.addButton
(QMessageBox::No);

                                                                                msgBox.setDefaul
tButton(QMessageBox::Yes);

                                                                                if(msgBox.exec()
 == QMessageBox::Yes) {
```

```
mapGame.saveAs(this->map_name, this->bg_name, this->bg_path);

this->unsaved_changes = false;

continue;
                                                                                }

                                                                      }
                                                  } else {
                                                            editorInventory.handleEvent(rend
erer, e, mapGame, camX, camY);
                                                            this->unsaved_changes = true;
                                                  }
                                        }
                              } else {
                                        editorInventory.handleEvent(renderer, e, mapGame
, camX, camY);
                              }
                    }

                    SDL_SetRenderDrawColor(renderer, 0x00, 0x00, 0x00, 0x00);
                    SDL_RenderClear(renderer);

                    camera.updateCameraPosition();

                    editorWindow.render(camera);

                    mapGame.render(renderer, camX, camY);

                    editorInventory.renderSelectedInMouse(renderer);

                    editorWindow.renderWater(camera);

                    editorInventory.render(renderer);

                    notice.render(renderer);

                    this->saveTexture.render(this->renderer, this->saveTexture.getX(
), this->saveTexture.getY(), SAVE_ICON_SIDE, SAVE_ICON_SIDE);
                    this->exitTexture.render(this->renderer, this->exitTexture.getX(
), this->exitTexture.getY(), EXIT_ICON_SIDE, EXIT_ICON_SIDE);

                    SDL_RenderPresent(renderer);
                    SDL_Delay(50); // Para no usar al mango el CPU
          }

          if (validMap && this->unsaved_changes) {
                    QMessageBox msgBox;
                    msgBox.setWindowTitle("Fin de ediciÃ³n");
                    msgBox.setText("Â¿Desea guardar el mapa?");
                    msgBox.setStandardButtons(QMessageBox::Yes);
                    msgBox.addButton(QMessageBox::No);
                    msgBox.setDefaultButton(QMessageBox::Yes);
                    if(msgBox.exec() == QMessageBox::Yes) {
                              mapGame.saveAs(this->map_name, this->bg_name, this->bg_p
ath);
                              this->unsaved_changes = false;
                              return 0;
                    }
          }

          return -1;
```

```
}
```

```
#ifndef __EDITOR_H__
#define __EDITOR_H__

#include <SDL2/SDL.h>
#include <string>
#include "map_game.h"
#include "window_game.h"
#include "yaml.h"
#include "inventory.h"
#include "inventory_editor.h"
#include "flash_notice.h"
#include "texture.h"

class Editor {
    private:
        std::string bg_name;
        std::string bg_path;
        std::string map_name;
        YAML::Node mapNode;
        YAML::Node staticNode;
        View::MapGame mapGame;
        View::WindowGame editorWindow;
        View::Camera camera;
        SDL_Renderer * renderer;
        View::EditorInventory editorInventory;
        View::Texture saveTexture;
        View::Texture exitTexture;
        FlashNotice notice;
        int teamsAmount;
        int wormsHealth;
        bool validMap;
        bool unsaved_changes;

    public:
        Editor(YAML::Node, std::string, std::string, std::string);
        int start(void);
};

#endif
```

```
#include <iostream>
#include <fstream>
#include <QFileDialog>
#include <QFileInfo>
#include <QMessageBox>
#include <sstream>
#include <QLineEdit>
#include "editor_launcher.h"
#include "types.h"
#include "ui_editor_launcher.h"
#include "yaml.h"
#include "editor.h"

#define DEFAULT_AMMO_QTY 10
#define DEFAULT_WORMS_HEALTH 200
#define DEFAULT_TEAMS_AMOUNT 2
#define DEFAULT_WATER_LEVEL 300

#define DEFAULT_SAVED_MAPS_PATH "/usr/etc/worms/maps/"
#define MAPS_EXT ".yml"

EditorLauncher::EditorLauncher(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::EditorLauncher)
{
    ui->setupUi(this);
    this->background_choosed = false;
    connectEvents();
}

EditorLauncher::~EditorLauncher()
{
    removeTempFiles();
    delete ui;
}

void EditorLauncher::removeTempFiles(void) {
    struct stat buffer1;
    struct stat buffer2;
    std::string path_map_yml = "/usr/etc/worms/temp/map.yml";
    std::string path_map_bg = "/usr/etc/worms/temp/background.png";
    if (stat (path_map_yml.c_str(), &buffer1) == 0) {
        std::string cmd_rm_map = "rm " + path_map_yml;
        std::system(cmd_rm_map.c_str());
    }
    if (stat (path_map_bg.c_str(), &buffer2) == 0) {
        std::string cmd_rm_bg = "rm " + path_map_bg;
        std::system(cmd_rm_bg.c_str());
    }
}

void EditorLauncher::connectEvents(void) {
    // Conecto el evento del boton exit

    QPushButton* choose_background = findChild<QPushButton*>("background_path");
    QObject::connect(choose_background, &QPushButton::clicked,
                     this, &EditorLauncher::chooseBackground);

    QPushButton* go_create = findChild<QPushButton*>("go_create");
    QObject::connect(go_create, &QPushButton::clicked,
                     this, &EditorLauncher::goCreate);
```

```
    QAction* load_and_edit = findChild<QAction*>("actionLoad_and_Edit");
    QObject::connect(load_and_edit, &QAction::triggered, this, &EditorLauncher::
loadAndEdit);

    QAction* create_new_map = findChild<QAction*>("actionNew_map");
    QObject::connect(create_new_map, &QAction::triggered, this, &EditorLauncher:
:createNewMap);
}

void EditorLauncher::chooseBackground(void) {
    QString bg_path;
    bg_path = QFileDialog::getOpenFileName(this, tr("Choose Background"), "/home", t
r("Image Files (*.png)"));
    this->background_path = bg_path.toUtf8().constData();
    QLabel* label_background_path = findChild<QLabel*>("label_background_path");
    label_background_path->setText(bg_path);
    if (bg_path.length() > 0) {
        this->background_choosed = true;
        QFileInfo bg_info(bg_path);
        this->background_name = bg_info.fileName().toUtf8().constData();
    }
}

void EditorLauncher::goCreate(void) {

    QString error_msg;
    bool error = false;
    std::string map_name;
    YAML::Node mapNode;

    if (!this->background_choosed) {
        error_msg += "Choose a background! \n";
        error = true;
    }

    if (findChild<QComboBox*>("background_options")->currentText() == "Background Option
s") {
        error_msg += "Background option is missing.\n";
        error = true;
    }

    if (findChild<QLineEdit*>("map_name")->text().length() == 0) {
        error_msg += "Write a map name!.\n";
    }

    if (error == true) {
        findChild<QLabel*>("label_errors")->setText(error_msg);
        return;
    }

    map_name = findChild<QLineEdit*>("map_name")->text().toUtf8().constData();
    mapNode["static"]["background"]["file"] = this->background_path;
    mapNode["static"]["background"]["display"] = findChild<QComboBox*>("background_options"
)->currentText().toUtf8().constData();
    mapNode["static"]["water_level"] = findChild<QSpinBox*>("water_level")->value();
    mapNode["static"]["teams_amount"] = findChild<QSpinBox*>("teams_amount")->value();
    mapNode["static"]["worms_health"] = findChild<QSpinBox*>("worms_health")->value();

    mapNode["static"]["init_inventory"][std::to_string(w_bazooka)]["item_name"] = "Bazook
a";
    mapNode["static"]["init_inventory"][std::to_string(w_bazooka)]["supplies"] = findChil
d<QSpinBox*>("bazooka_ammo")->value();
```

```
    mapNode["static"]["init_inventory"][std::to_string(w_mortar)]["item_name"] = "Mortar";
    mapNode["static"]["init_inventory"][std::to_string(w_mortar)]["supplies"] = findChild
<QSpinBox*>("mortar_ammo")->value();

    mapNode["static"]["init_inventory"][std::to_string(w_cluster)]["item_name"] = "Cluster
";
    mapNode["static"]["init_inventory"][std::to_string(w_cluster)]["supplies"] = findChil
d<QSpinBox*>("red_bomb_ammo")->value();

    mapNode["static"]["init_inventory"][std::to_string(w_banana)]["item_name"] = "Banana"
;
    mapNode["static"]["init_inventory"][std::to_string(w_banana)]["supplies"] = findChild
<QSpinBox*>("banana_ammo")->value();

    mapNode["static"]["init_inventory"][std::to_string(w_green_grenade)]["item_name"] =
"Grenade";
    mapNode["static"]["init_inventory"][std::to_string(w_green_grenade)]["supplies"] = fi
ndChild<QSpinBox*>("green_bomb_ammo")->value();

    mapNode["static"]["init_inventory"][std::to_string(w_holy_grenade)]["item_name"] = "
Holy bomb";
    mapNode["static"]["init_inventory"][std::to_string(w_holy_grenade)]["supplies"] = fin
dChild<QSpinBox*>("holy_bomb_ammo")->value();

    mapNode["static"]["init_inventory"][std::to_string(w_dynamite)]["item_name"] = "Dyna
mite";
    mapNode["static"]["init_inventory"][std::to_string(w_dynamite)]["supplies"] = findChi
ld<QSpinBox*>("dynamite_ammo")->value();

    mapNode["static"]["init_inventory"][std::to_string(w_air_strike)]["item_name"] = "Air
 Strike";
    mapNode["static"]["init_inventory"][std::to_string(w_air_strike)]["supplies"] = findC
hild<QSpinBox*>("fly_bombs_ammo")->value();

    mapNode["static"]["init_inventory"][std::to_string(w_teleport)]["item_name"] = "Telepo
rt";
    mapNode["static"]["init_inventory"][std::to_string(w_teleport)]["supplies"] = findChi
ld<QSpinBox*>("teleport_ammo")->value();

    mapNode["static"]["init_inventory"][std::to_string(w_bat)]["item_name"] = "Bat";
    mapNode["static"]["init_inventory"][std::to_string(w_bat)]["supplies"] = findChild<QS
pinBox*>("bat_ammo")->value();

    std::string map_path = DEFAULT_SAVED_MAPS_PATH + map_name + MAPS_EXT;

/*      std::stringstream ss;
    ss << mapNode;
    std::cout << ss.str() << std::endl; */

    launchEditor(mapNode, map_name);
}

void EditorLauncher::launchEditor(YAML::Node mapNode, std::string & map_name) {
    std::cout << "About to construct the_editor" << std::endl;
    Editor the_editor(mapNode, map_name, this->background_name, this->background
_path);
    std::cout << "Finish constructing the_editor" << std::endl;
    this->hide();
    int err_code;
    err_code = the_editor.start();
    if (err_code == 0) {
```

```
    }
    this->close();
}

void EditorLauncher::loadAndEdit(void) {
    std::cout << "Se carga un mapa existente para editarlo." << std::endl;
    QString map_path;
    map_path = QFileDialog::getOpenFileName(this, tr("Choose Map"), "/usr/etc/worms/ma
ps", tr("Tar gzipped (*.tar.gz)"));

    std::string str_map_path = map_path.toUtf8().constData();
    QFile f(map_path);
    QFileInfo file_info(f.fileName());
    QString file_name(file_info.fileName());
    std::string str_file_name = file_name.toUtf8().constData();
    size_t lastindex = str_file_name.find_first_of(".");
    std::string file_raw_name = str_file_name.substr(0, lastindex);


    std::cout << "El nombre del mapa es " << file_raw_name << std::endl;

    bool valid_map = validateChoosedMap(str_map_path);
    if (!valid_map) {
        QMessageBox msgBox;
        msgBox.setWindowTitle("Mapa invÃ¡lido.");
        std::string msg_response = "El mapa elegido para ediciÃ³n no es vÃ¡lido.";
        msgBox.setText(msg_response.c_str());
        msgBox.exec();
        return;
    }

    this->background_path = "/usr/etc/worms/temp/background.png";
    this->background_choosed = true;
    this->background_name = "background.png";

    YAML::Node map_node = YAML::LoadFile("/usr/etc/worms/temp/map.yml");
    map_node["static"]["background"]["file"] = this->background_path;
    launchEditor(map_node, file_raw_name);
}

bool EditorLauncher::validateChoosedMap(std::string & map_path) {
    std::string cmd_untar_map = "tar -xf " + map_path + " -C /usr/etc/worms/temp";
    if (std::system(cmd_untar_map.c_str()) < 0) {
        std::cout << "No se pudo descomprimir el mapa elegido para editar." << std::endl;
        return false;
    }

    struct stat buffer1;
    struct stat buffer2;
    std::string path_map_yml = "/usr/etc/worms/temp/map.yml";
    std::string path_map_bg = "/usr/etc/worms/temp/background.png";
    if (stat (path_map_yml.c_str(), &buffer1) != 0) {
        std::cout << "No se encontro el map.yml dentro del mapa elegido." << std::endl;
        return false;
    }
    if (stat (path_map_bg.c_str(), &buffer2) != 0) {
        std::cout << "No se encontrÃ³ el background.png dentro del mapa elegido." << std::endl;
        std::string cmd_clean = "rm /usr/etc/worms/temp/map.yml";
        std::system(cmd_clean.c_str());
        return false;
    }
    return true;
}

void EditorLauncher::createNewMap(void) {
    QMessageBox msgBox;
    msgBox.setWindowTitle("Crear nuevo mapa");
    msgBox.setText("PerderÃ¡ los cambios actuales. Â¿EstÃ¡ seguro que desea reiniciar la configuraciÃ³n act
ual?");
    msgBox.setStandardButtons(QMessageBox::Yes);
    msgBox.addButton(QMessageBox::No);
    msgBox.setDefaultButton(QMessageBox::Yes);
    if(msgBox.exec() == QMessageBox::No) {
        return;
    }
    findChild<QLineEdit*>("map_name")->clear();
    findChild<QLabel*>("label_background_path")->clear();
    findChild<QComboBox*>("background_options")->setCurrentIndex(0);
    findChild<QSpinBox*>("water_level")->setValue(DEFAULT_WATER_LEVEL);
    findChild<QSpinBox*>("teams_amount")->setValue(DEFAULT_TEAMS_AMOUNT);
    findChild<QSpinBox*>("worms_health")->setValue(DEFAULT_WORMS_HEALTH);

    findChild<QSpinBox*>("mortar_ammo")->setValue(DEFAULT_AMMO_QTY);
    findChild<QSpinBox*>("red_bomb_ammo")->setValue(DEFAULT_AMMO_QTY);
    findChild<QSpinBox*>("banana_ammo")->setValue(DEFAULT_AMMO_QTY);
    findChild<QSpinBox*>("green_bomb_ammo")->setValue(DEFAULT_AMMO_QTY);
    findChild<QSpinBox*>("holy_bomb_ammo")->setValue(DEFAULT_AMMO_QTY);
    findChild<QSpinBox*>("dynamite_ammo")->setValue(DEFAULT_AMMO_QTY);
    findChild<QSpinBox*>("fly_bombs_ammo")->setValue(DEFAULT_AMMO_QTY);
    findChild<QSpinBox*>("teleport_ammo")->setValue(DEFAULT_AMMO_QTY);
    findChild<QSpinBox*>("bat_ammo")->setValue(DEFAULT_AMMO_QTY);
    findChild<QSpinBox*>("bazooka_ammo")->setValue(DEFAULT_AMMO_QTY);

    this->background_path.clear();
    this->background_name.clear();
    this->background_choosed = false;
    this->background_mode.clear();
    this->water_level = DEFAULT_WATER_LEVEL;
    this->teams_amount = DEFAULT_TEAMS_AMOUNT;
    this->worms_health = DEFAULT_WORMS_HEALTH;

}
```

```
#ifndef EDITOR_LAUNCHER_H
#define EDITOR_LAUNCHER_H

#include <QMainWindow>
#include <string>
#include "yaml.h"

namespace Ui {
class EditorLauncher;
}

class EditorLauncher : public QMainWindow
{
    Q_OBJECT

public:
    explicit EditorLauncher(QWidget *parent = 0);
    ~EditorLauncher();

private:
    Ui::EditorLauncher *ui;
    std::string background_path;
    std::string background_name;
    bool background_choosed;
    std::string background_mode;
    int water_level;
    int teams_amount;
    int worms_health;
    std::map<int, size_t> weapons_ammo;
    size_t mortar_ammo;
    size_t red_bomb_ammo;
    size_t banana_ammo;
    size_t green_bomb_ammo;
    size_t holy_bomb_ammo;
    size_t dynamite_ammo;
    size_t fly_bombs_ammo;
    size_t teleport_ammo;
    size_t bat_ammo;

    void connectEvents(void);
    void chooseBackground(void);
    void goCreate(void);
    void launchEditor(YAML::Node, std::string &);
    void loadAndEdit(void);
    void createNewMap(void);
    bool validateChoosedMap(std::string &);
    void removeTempFiles(void);
};

#endif // EDITOR_LAUNCHER_H
```

```
#include <iostream>
#include <QApplication>
#include "editor_launcher.h"
#include <SDL2/SDL.h>
#include <string>
#include <vector>
#include "window_game.h"
#include "girder_long.h"
#include "paths.h"
#include "girder_short.h"
#include "inventory.h"
#include "inventory_editor.h"
#include "inventory_weapons.h"
#include "map_game.h"
#include "worm.h"
#include "yaml.h"

#define ARGC_DEFAULT 1
#define ARGC_FILE_CONFIG 2

// Variable global
Paths gPath;

void validateArgs(int, char*[], YAML::Node & map);

int main(int argc, char * argv[]) {
    QApplication a(argc, argv);
    EditorLauncher w;
    w.show();

    return a.exec();
}
```

```
#include "map_game.h"

#define DEFAULT_SAVED_MAPS_PATH "/usr/etc/worms/maps/"

View::MapGame::MapGame(YAML::Node & map) :
map(map) {
  this->index = 0;
  this->stateIndex = 0;
}

View::MapGame::~MapGame() {
  for(int i = this->mapStates.size() - 1; i >= 0; --i) {
    delete this->mapStates[i];
    this->mapStates[i] = nullptr;
  }
}

void View::MapGame::initializeStates() {
  this->mapStates.push_back(new MapState());

  if (!this->map["dynamic"]) {
    return;
  }

  std::cout << "EXISTE DYNAMIC" << std::endl;

  const YAML::Node& shortGirders = this->map["static"]["short_girders"];
  const YAML::Node& longGirders = this->map["static"]["long_girders"];
  const YAML::Node& wormsTeams = this->map["dynamic"]["worms_teams"];

  std::cout << "NODOS CREADO" << std::endl;
  int x = 0;
  int y = 0;

  std::stringstream ss;
  ss << this->map;
  //std::cout << ss.str().c_str() << std::endl;

  for (YAML::const_iterator it = shortGirders.begin(); it != shortGirders.end();
 ++it) {
    std::cout << "ITERANDO SOBRE SHORT GIRDERS" << std::endl;
    const YAML::Node & shortGirder = *it;

    std::cout << "TOMANDO DATOS DE SHORT GIRDER" << std::endl;
    x = shortGirder["x"].as<int>();
    y = shortGirder["y"].as<int>();
    std::cout << x << " " << y << std::endl;
    degrees_t degrees = (degrees_t) shortGirder["angle"].as<int>();
    std::cout << "DEGREES " << degrees << std::endl;
    addShortGirder(degrees, x , y);
    std::cout << "SHORT GIRDER AGREGADA" << std::endl;
  }

  for (YAML::const_iterator it = longGirders.begin(); it != longGirders.end(); +
+it) {
    const YAML::Node & longGirder = *it;
    x = longGirder["x"].as<int>();
    y = longGirder["y"].as<int>();
    degrees_t degrees = (degrees_t) longGirder["angle"].as<int>();
    addLongGirder(degrees, x, y);
  }
  int tid = 0;
  std::string name;
  int health = 0;

  for (YAML::const_iterator it = wormsTeams.begin(); it != wormsTeams.end(); ++i
t) {
    tid = it->first.as<int>();
    const YAML::Node& wormsNode = it->second["worms"];
    for (YAML::const_iterator worms = wormsNode.begin(); worms != wormsNode.end(
); worms++) {
      const YAML::Node& worm = *worms;
      name = worm["name"].as<std::string>();
      health = worm["health"].as<int>();
      x = worm["x"].as<int>();
      y = worm["y"].as<int>();
      addWormInTeam(tid, name, health, x, y);
    }
  }
}

void View::MapGame::createMapToSave() {
    mapToSave["static"]["background"] = YAML::Clone(map["static"]["background"]);
    mapToSave["static"]["water_level"] = map["static"]["water_level"];
    mapToSave["static"]["teams_amount"] = map["static"]["teams_amount"];
    mapToSave["static"]["worms_health"] = map["static"]["worms_health"];
    mapToSave["static"]["init_inventory"] = YAML::Clone(map["static"]["init_inventory"]);
}

void View::MapGame::setRenderer(SDL_Renderer * renderer) {
  this->renderer = renderer;
}

void View::MapGame::render(SDL_Renderer * renderer, int camX, int camY) {
    if (this->mapStates.size() != 0) {
    this->mapStates[stateIndex]->render(renderer, camX, camY);
  }
}
/* Add methods */
void View::MapGame::addShortGirder(degrees_t degrees, int x, int y) {
  this->updateIndex();
  std::cout << "INDEX UPDATED" << std::endl;
  MapState* previousState = this->mapStates.back();
  MapState* newState = new MapState();
  newState->operator=(previousState);
  newState->addShortGirder(renderer, degrees, x, y);
  this->mapStates.push_back(newState);
}

void View::MapGame::addLongGirder(degrees_t degrees, int x, int y) {
  this->updateIndex();
  MapState* previousState = this->mapStates.back();
  MapState* newState = new MapState();
  newState->operator=(previousState);
  newState->addLongGirder(renderer, degrees, x, y);
  this->mapStates.push_back(newState);
}

void View::MapGame::addWormInTeam(int teamId, std::string & name, int health, in
t x, int y) {
  this->updateIndex();
  MapState* previousState = this->mapStates.back();
  MapState* newState = new MapState();
```

```
  newState->operator=(previousState);
  newState->addWorm(renderer, teamId, name, health, x, y);
  this->mapStates.push_back(newState);
}

void View::MapGame::setPreviousState(View::EditorInventory & inv) {
  if (this->stateIndex) {
    this->stateIndex--;
    inv.updateWormsTeamSupplies(this->mapStates[this->stateIndex]->getWorms());
  }
}

void View::MapGame::setNextState(View::EditorInventory & inv) {
  if (this->stateIndex != this->mapStates.size() - 1) {
    this->stateIndex++;
    inv.updateWormsTeamSupplies(this->mapStates[this->stateIndex]->getWorms());
  }
}

void View::MapGame::updateIndex(void) {
  this->stateIndex++;

  if (this->stateIndex != this->mapStates.size()) {
    std::vector<MapState*>::iterator it = this->mapStates.begin() + this->stateI
ndex;
    for (; it != this->mapStates.end();) {
      delete *it;
      *it = nullptr;
      it = this->mapStates.erase(it);
    }
  }
}

void View::MapGame::printCurrentState(void) {
  // std::cout << *this->mapStates[this->stateIndex] << std::endl;
}

int View::MapGame::getNextWormId(void) {
  // int newId = 1;
  // YAML::Node * state = this->mapStates[this->stateIndex];
  // const YAML::Node & teams = (*state)["dynamic"]["worms_teams"];
  // YAML::const_iterator it = teams.begin();

  // for (; it != teams.end() ; it++) {
  //   newId += (it->second)["worms"].size();
  // }

  // return newId;
}

void View::MapGame::saveAs(std::string mapName, std::string bgName, std::string
bgPath) {
  this->mapToSave.reset();
  createMapToSave();
  addMaxWormsAmount();
  addShortGirdersToMap();
  addLongGirdersToMap();
  addWormsToMap();
  std::ofstream fout("/usr/etc/worms/maps/map.yml", std::ofstream::trunc);
  std::string bg_name = "background.png";
  this->mapToSave["static"]["background"]["file"] = bg_name;
  addInventoryToTeams();
  fout << mapToSave;
  /* std::cout << "This map" << std::endl;
  std::cout << this->map << std::endl; */
  fout.close();

  std::string maps_path(DEFAULT_SAVED_MAPS_PATH);
  std::string cmd_cp_background = "cp \"" + bgPath + "\" " + maps_path + "background.
png";
  std::system(cmd_cp_background.c_str());

  struct stat buffer;
  std::string map_path = maps_path + mapName + ".tar.gz";
  std::cout << "Chequeando si existe el archivo " << map_path << std::endl;
  if (stat(map_path.c_str(), &buffer) == 0) {
    std::string cmd_rm_previous_map = "rm " + map_path;
    std::system(cmd_rm_previous_map.c_str());
    std::cout << "Mapa previo removido." << std::endl;
  }

  std::string cmd_tar_gz = "tar -zcf \"" + maps_path + mapName + ".tar.gz\" --directory=" +
 maps_path + " map.yml background.png";
  std::system(cmd_tar_gz.c_str());
  std::string cmd_rmv_temp = "rm " + maps_path + "background.png " + maps_path + "ma
p.yml";
  std::system(cmd_rmv_temp.c_str());
  std::cout << "Mapa guardado!" << std::endl;
}

void View::MapGame::addLongGirdersToMap() {
  std::map<int, View::GirderLong*> longGirders = this->mapStates[this->stateInde
x]->getLongGirders();
  int longGirderCounter = 1;
  std::map<int, View::GirderLong*>::const_iterator longGirder;
  for (longGirder = longGirders.begin(); longGirder != longGirders.end(); ++long
Girder) {
    YAML::Node newGirderNode;
    newGirderNode["id"] = longGirderCounter;
    newGirderNode["x"] = longGirder->second->getX();
    newGirderNode["y"] = longGirder->second->getY();
    newGirderNode["angle"] = (int) longGirder->second->getCurrentDegrees();
    this->mapToSave["static"]["long_girders"].push_back(newGirderNode);
    longGirderCounter++;
  }
}

void View::MapGame::addShortGirdersToMap() {
  std::map<int, View::GirderShort*> shortGirders = this->mapStates[this->stateIn
dex]->getShortGirders();
  int shortGirderCounter = 1;
  std::map<int, View::GirderShort*>::const_iterator shortGirder;
  for (shortGirder = shortGirders.begin(); shortGirder != shortGirders.end(); ++
shortGirder) {
    YAML::Node newGirderNode;
    newGirderNode["id"] = shortGirderCounter;
    newGirderNode["x"] = shortGirder->second->getX();
    newGirderNode["y"] = shortGirder->second->getY();
    newGirderNode["angle"] = (int) shortGirder->second->getCurrentDegrees();
    this->mapToSave["static"]["short_girders"].push_back(newGirderNode);
    shortGirderCounter++;
  }
}
```

```
void View::MapGame::addWormsToMap() {
  std::map<size_t, std::vector<View::Worm*>> worms = this->mapStates[this->state
Index]->getWorms();
  int wormCounter = 1;
  std::map<std::size_t, std::vector<View::Worm*>>::const_iterator worm;
  for (worm = worms.begin(); worm != worms.end(); ++worm) {
    std::vector<View::Worm*>::const_iterator worm_it;
    for (worm_it = worm->second.begin(); worm_it != worm->second.end(); worm_it+
+) {
      YAML::Node newWorm;
      newWorm["id"] = wormCounter;
      newWorm["name"] = (*worm_it)->getName();
      newWorm["health"] = (*worm_it)->getHealth();
      newWorm["x"] = (*worm_it)->getX();
      newWorm["y"] = (*worm_it)->getY();
      newWorm["sight_angle"] = 0;
      newWorm["status"]["grounded"] = 0;
      newWorm["status"]["falling"] = 1;
      newWorm["status"]["mirrored"] = 0;
      newWorm["status"]["walking"] = 0;
      this->mapToSave["dynamic"]["worms_teams"][worm->first]["worms"].push_back(new
Worm);
      wormCounter++;
    }
  }
}

void View::MapGame::addMaxWormsAmount(void) {
  size_t max = 0;
  std::map<std::size_t, std::vector<View::Worm*>>::const_iterator it;
  std::map<size_t, std::vector<View::Worm*>> worms = this->mapStates[this->state
Index]->getWorms();
  for (it = worms.begin(); it != worms.end(); ++it) {
    if (it->second.size() > max) {
      max = it->second.size();
    }
  }
  this->mapToSave["static"]["max_worms"] = max;
}

void View::MapGame::addInventoryToTeams() {
  YAML::iterator it = mapToSave["dynamic"]["worms_teams"].begin();

  for (; it != mapToSave["dynamic"]["worms_teams"].end() ; it++) {
      it->second["inventory"] = YAML::Clone(this->mapToSave["static"]["init_inventory"]);
  }
}

bool View::MapGame::hasWorms() {
  std::map<size_t, std::vector<View::Worm*>> worms = this->mapStates[this->state
Index]->getWorms();
  if (worms.size() < 2) return false;
  std::map<size_t, std::vector<View::Worm*>>::iterator it;
  for (it = worms.begin(); it != worms.end(); ++it) {
    if (it->second.size() == 0) {
      return false;
    }
  }
  return true;
}

int View::MapGame::amountWormsTeam(int teamId) {
  std::map<size_t, std::vector<View::Worm*>> worms = this->mapStates[this->state
Index]->getWorms();
  return worms[teamId].size();
}
```

```
#ifndef __MAP_GAME_H__
#define __MAP_GAME_H__

#include <SDL2/SDL.h>
#include <vector>
#include <map>
#include <string>
#include <fstream>
#include "girder_long.h"
#include "girder_short.h"
#include "inventory_editor.h"
#include "girder.h"
#include "worm.h"
#include "yaml.h"
#include "map_state.h"

namespace View {
  class EditorInventory;

  class MapGame {
    private:

      size_t statIndex;

      std::vector<MapState*> mapStates;
      unsigned int stateIndex;
      SDL_Renderer * renderer;
      YAML::Node & map;
      YAML::Node mapToSave;
      unsigned int index;

      // Obtiene el id del proximo
      // worm a agregar
      int getNextWormId(void);
      void addInventoryToTeams();
      void addLongGirdersToMap();
      void addShortGirdersToMap();
      void addWormsToMap();
      void addMaxWormsAmount(void);
      void updateIndex();

    public:
      // Constructor, recibe el nodo YAML
      // donde guardara toda la informacion del mapa
      MapGame(YAML::Node &);

      // Destructor, libera los items dibujados
      ~MapGame();

      // Dibuja lo que ya fue clickeado por el usuario
      void render(SDL_Renderer * r, int camX, int camY);

      // Agrega una short girder en la posicion del mapa indicada
      void addShortGirder(degrees_t, int, int);

      // Agrega una long girder en la posicion del mapa indicada
      void addLongGirder(degrees_t, int, int);

      // Agrega un worm
      void addWormInTeam(int, std::string &, int, int, int);

      // Establece el estado anterior (si hay)
      void setPreviousState(View::EditorInventory &);

      // Establece el estado posterior (si hay)
      void setNextState(View::EditorInventory &);

      // Imprime el estado actual
      void printCurrentState(void);

      // Guarda el mapa en la carpeta de mapas del servidor
      // bajo el nombre indicado
      void saveAs(std::string, std::string, std::string);

      bool hasWorms();

      void setRenderer(SDL_Renderer * renderer);
      void initializeStates();
      void createMapToSave();
      int amountWormsTeam(int);
  };
}

#endif
```

```
#include "map_state.h"

MapState::MapState() {
  this->newLongGirder = nullptr;
  this->newWorm = nullptr;
  this->newShortGirder = nullptr;
}

MapState::~MapState() {
  if (this->newShortGirder) {
    delete this->newShortGirder;
    this->newShortGirder = nullptr;
  }
  // } else if (this->newWorm) {
  //    delete this->newWorm;
  //    this->newWorm = nullptr;
   else if (this->newLongGirder) {
    delete this->newLongGirder;
    this->newLongGirder = nullptr;
  }
}

std::map<int, View::GirderShort*> MapState::getShortGirders() {
  return this->shortGirders;
}

std::map<int, View::GirderLong*> MapState::getLongGirders() {
  return this->longGirders;
}

std::map<std::size_t, std::vector<View::Worm*>> MapState::getWorms() {
  return this->worms;
}

void MapState::operator=(MapState* mapState) {
  this->shortGirders = mapState->getShortGirders();
  this->longGirders = mapState->getLongGirders();
  this->worms = mapState->getWorms();
}

void MapState::addShortGirder(SDL_Renderer* renderer, degrees_t degrees, int x,
int y) {
  this->newShortGirder = new View::GirderShort(renderer, degrees);
  newShortGirder->setX(x);
  newShortGirder->setY(y);
  this->shortGirders.insert(std::pair<int, View::GirderShort*>(this->shortGirder
s.size() + 1,
    this->newShortGirder));
}

void MapState::addLongGirder(SDL_Renderer* renderer, degrees_t degrees, int x, i
nt y) {
  this->newLongGirder = new View::GirderLong(renderer, degrees);
  newLongGirder->setX(x);
  newLongGirder->setY(y);

  this->longGirders.insert(std::pair<int, View::GirderLong*>(this->longGirders.s
ize() + 1,
  this->newLongGirder));
}

void MapState::addWorm(SDL_Renderer* renderer, int teamId, std::string & name, i
nt health, int x, int y) {
  this->newWorm = new View::Worm(renderer, name, teamId, health);
  newWorm->setX(x);
  newWorm->setY(y);
  this->worms[teamId].push_back(this->newWorm);
}


void MapState::render(SDL_Renderer * renderer, int camX, int camY) {
  // Render short girders
  std::map<int, View::GirderShort*>::iterator shortGirder;
  for (shortGirder = this->shortGirders.begin(); shortGirder != this->shortGirde
rs.end(); ++shortGirder) {
    shortGirder->second->render(renderer, camX, camY);
  }

  // Render long girders
  std::map<int, View::GirderLong*>::iterator longGirder;
  for (longGirder = this->longGirders.begin(); longGirder != this->longGirders.e
nd(); ++longGirder) {
    longGirder->second->render(renderer, camX, camY);
  }

  // Render worms
  std::map<std::size_t, std::vector<View::Worm*>>::iterator worm;
  for (worm = worms.begin(); worm != worms.end(); ++worm) {
    std::vector<View::Worm*>::iterator worm_it;
    for (worm_it = worm->second.begin(); worm_it != worm->second.end(); worm_it+
+) {
      (*worm_it)->render(renderer, camX, camY);
    }
  }
}
```

```
#ifndef __MAP_STATE_H__
#define __MAP_STATE_H__

#include <map>
#include "girder_long.h"
#include "girder_short.h"
#include "worm.h"
#include <SDL2/SDL.h>

class MapState {
private:
    std::map<int, View::GirderShort*> shortGirders;
    std::map<int, View::GirderLong*> longGirders;
        std::map<std::size_t, std::vector<View::Worm*>> worms;

    View::GirderShort* newShortGirder;
    View::GirderLong* newLongGirder;
    View::Worm* newWorm;
public:
    MapState();
    ~MapState();
    void render(SDL_Renderer* renderer, int camX, int camY);
    void operator=(MapState* mapState);
    std::map<int, View::GirderShort*> getShortGirders();
    std::map<int, View::GirderLong*> getLongGirders();
    std::map<std::size_t, std::vector<View::Worm*>> getWorms();
    void addShortGirder(SDL_Renderer* rendeder, degrees_t degrees, int x, int y)
;
    void addLongGirder(SDL_Renderer* renderer, degrees_t degrees, int x, int y);
    void addWorm(SDL_Renderer* renderer, int teamId, std::string & name, int hea
lth, int x, int y);
};

#endif
```

Table of Contents