```
#ifndef __BLOCKING_QUEUE_H__
#define __BLOCKING_QUEUE_H__

#include <mutex>
#include <condition_variable>
#include <queue>

template <typename T>
class Queue {
    private:
        std::queue<T> q;
            const unsigned int max_size;
        std::mutex mtx;
        std::condition_variable is_not_full;
        std::condition_variable is_not_empty;

        Queue(const Queue&) = delete;
        Queue& operator=(const Queue&) = delete;

    public:
        Queue(const unsigned int ms) : max_size(ms) {};

        void push(const T & val) {
            std::unique_lock<std::mutex> lck(mtx);
            if (q.empty()) {
                is_not_empty.notify_all();
            }

            while (q.size() >= this->max_size) {
                is_not_full.wait(lck);
            }

            q.push(val);
        }

        T pop(void) {
            std::unique_lock<std::mutex> lck(mtx);

            while (q.empty()) {
                is_not_empty.wait(lck);
            }

            const T val = q.front();
            q.pop();
            is_not_full.notify_all();

            return val;
        }

        size_t size(void) {
            return this->q.size();
        }
};

#endif
```

```
#include "event.h"
#include <iostream>
#include <string>
#include <sstream>
#include "yaml.h"
#include "types.h"

Event::Event(action_t a, size_t tid) {
    this->action = a;
    this->team_id = tid;
    this->eventNode["event"]["team_id"] = std::to_string(tid);
    this->eventNode["event"]["action"] = std::to_string(a);
}

Event::Event(action_t a, std::string & mn, size_t map_players_qty) {
    this->action = a;
    this->team_id = 0;
    this->eventNode["event"]["action"] = std::to_string(a);
    this->eventNode["event"]["match_name"] = mn;
    this->eventNode["event"]["map_players_qty"] = std::to_string(map_players_qty);
    this->eventNode["event"]["team_id"] = 0;
}

Event::Event(action_t a, std::string & mcn) {
    this->action = a;
    this->team_id = 0;
    this->eventNode["event"]["action"] = std::to_string(a);
    this->eventNode["event"]["creator_name"] = mcn;
    this->eventNode["event"]["team_id"] = 0;
}

Event::Event(action_t a, weapon_t w, size_t tid, int countdown, int power, int s
ight_angle) {
    this->action = a;
    this->team_id = tid;
    this->eventNode["event"]["team_id"] = std::to_string(tid);
    this->eventNode["event"]["action"] = std::to_string(a);
    this->eventNode["event"]["weapon"] = std::to_string(w);

    if (countdown != -1) {
        this->eventNode["event"]["countdown"] = std::to_string(countdown);
    }

    if (power != -1) {
        int powerConverted = (power / 150) + 10; // Hardcodeado
        this->eventNode["event"]["power"] = std::to_string(powerConverted);
    }

    if (sight_angle != -1) {
        this->eventNode["event"]["sight_angle"] = std::to_string(sight_angle);
    }
}

Event::Event(action_t a, weapon_t w, size_t tid, int remoteX, int remoteY) {
    this->action = a;
    this->team_id = tid;
    this->eventNode["event"]["team_id"] = std::to_string(tid);
    this->eventNode["event"]["action"] = std::to_string(a);
    this->eventNode["event"]["weapon"] = std::to_string(w);
    this->eventNode["event"]["remote_control_x"] = std::to_string(remoteX);
    this->eventNode["event"]["remote_control_y"] = std::to_string(remoteY);
}
```

```
Event::Event(YAML::Node & event) {
    this->eventNode = YAML::Clone(event);
    this->action = (action_t) event["event"]["action"].as<int>();
    this->team_id = event["event"]["team_id"].as<size_t>();
}

YAML::Node Event::getNode(void) {
    return this->eventNode;
}

bool Event::quit(void) {
    return (this->action == a_quitGame || this->action == a_quitLobby) ? true :
false;
}

bool Event::createMatch(void) {
    return this->action == a_createMatch ? true : false;
}

size_t Event::getTeamId(void) {
    return this->team_id;
}

bool Event::goToMatch(void) {
    return this->action == a_goToMatch;
}
```

```
#ifndef __EVENT_H__
#define __EVENT_H__

#include <iostream>
#include "types.h"
#include "yaml.h"
#include <string>

class Event {
    private:
        YAML::Node eventNode;
        action_t action;
        size_t team_id;
        std::string matchName;
    public:
        Event(action_t action = a_noEvent , size_t team_id = 0);
        Event(action_t, std::string &, size_t map_players_qty);
        Event(action_t, std::string &);
        // Las armas que no se pueden configurar
        // el countdown y o la potencia tienen
        // como parametro por default -1
        Event(action_t action, weapon_t, size_t, int countdown, int power, int s
ight_angle);

        // Para armas teledirigidas
        Event(action_t action, weapon_t, size_t, int remoteX, int remoteY);

        Event(YAML::Node &);
        bool quit(void);
        bool createMatch(void);
        bool goToMatch(void);
        YAML::Node getNode(void);
        size_t getTeamId(void);
};

#endif
```

```cpp
#include "paths.h"

Paths::Paths(const char * r) {
  if (r) {
    this->root = r;
  } else {
    struct stat sb;
    // Checkeamos si existe el directorio del instalador
    if (stat("/usr/var/worms/", &sb) == 0 && S_ISDIR(sb.st_mode)) {
      // Entonces el worms esta instalado
      this->root = "/usr/var/worms/";
    } else {
      this->root = "../../";
    }
  }

  /* GIRDERS PATHS */
  // Long
  this->PATH_GIRDER_LONG_90 = root + "resources/graphics/Weapons/grdl0.png";
  this->PATH_GIRDER_LONG_60 = root + "resources/graphics/Weapons/grdl1.png";
  this->PATH_GIRDER_LONG_45 = root + "resources/graphics/Weapons/grdl2.png";
  this->PATH_GIRDER_LONG_30 = root + "resources/graphics/Weapons/grdl3.png";
  this->PATH_GIRDER_LONG_0 = root + "resources/graphics/Weapons/grdl4.png";
  this->PATH_GIRDER_LONG_NEGATIVE_30 = root + "resources/graphics/Weapons/grdl5.png";
  this->PATH_GIRDER_LONG_NEGATIVE_45 = root + "resources/graphics/Weapons/grdl6.png";
  this->PATH_GIRDER_LONG_NEGATIVE_60 = root + "resources/graphics/Weapons/grdl7.png";
  this->PATH_GIRDER_LONG_NEGATIVE_90 = root + "resources/graphics/Weapons/grdl8.png";

  // Short
  this->PATH_GIRDER_SHORT_90 = root + "resources/graphics/Weapons/grds0.png";
  this->PATH_GIRDER_SHORT_60 = root + "resources/graphics/Weapons/grds1.png";
  this->PATH_GIRDER_SHORT_45 = root + "resources/graphics/Weapons/grds2.png";
  this->PATH_GIRDER_SHORT_30 = root + "resources/graphics/Weapons/grds3.png";
  this->PATH_GIRDER_SHORT_0 = root + "resources/graphics/Weapons/grds4.png";
  this->PATH_GIRDER_SHORT_NEGATIVE_30 = root + "resources/graphics/Weapons/grds5.png";
  this->PATH_GIRDER_SHORT_NEGATIVE_45 = root + "resources/graphics/Weapons/grds6.png";
  this->PATH_GIRDER_SHORT_NEGATIVE_60 = root + "resources/graphics/Weapons/grds7.png";
  this->PATH_GIRDER_SHORT_NEGATIVE_90 = root + "resources/graphics/Weapons/grds8.png";
  /* ------------------------------ */

  /* WATER PATHS */
  this->PATH_WATER_DEFAULT = root + "resources/graphics/water_gif.gif";
  this->PATH_WATER_2 = root + "resources/graphics/water_pattern_2.png";
  this->PATH_WATER_3 = root + "resources/graphics/water_pattern_3.png";
  this->PATH_LAVA = root + "resources/graphics/lava_pattern.jpg";
  /* --------------------------- */


  /* WEAPON ICONS */
  this->PATH_ICON_BAZOOKA = root + "resources/graphics/Weapon Icons/bazooka.1.png";
  this->PATH_ICON_MORTAR = root + "resources/graphics/Weapon Icons/mortar.1.png";
  this->PATH_ICON_GREEN_GRENADE = root + "resources/graphics/Weapon Icons/grenade.1.png";
  this->PATH_ICON_RED_GRENADE = root + "resources/graphics/Weapon Icons/cluster.1.png";
  this->PATH_ICON_BANANA = root + "resources/graphics/Weapon Icons/banana.1.png";
  this->PATH_ICON_HOLY_GRENADE = root + "resources/graphics/Weapon Icons/hgrenade.1.png";
  this->PATH_ICON_DYNAMITE = root + "resources/graphics/Weapon Icons/dynamite.1.png";
  this->PATH_ICON_BASEBALL = root + "resources/graphics/Weapon Icons/baseball.1.png";
  this->PATH_ICON_AIR_STRIKE = root + "resources/graphics/Weapon Icons/airstrke.1.png";
  this->PATH_ICON_TELEPORT = root + "resources/graphics/Weapon Icons/teleport.1.png";
  this->PATH_ICON_SHORT_GIRDER = root + "resources/graphics/Weapon Icons/girder.2.png";
  this->PATH_ICON_LONG_GIRDER = root + "resources/graphics/Weapon Icons/girders.1.png";
  /* ------------- */
```

```cpp

  /* WEAPONS */
  this->PATH_DYNAMITE = root + "resources/graphics/Weapons/dynamite.png";
  this->PATH_GREEN_GRENADE = root + "resources/graphics/Weapons/grenade.png";
  this->PATH_HOLY_GRENADE = root + "resources/graphics/Weapons/hgrenade.png";
  this->PATH_BANANA = root + "resources/graphics/Weapons/banana.png";
  this->PATH_BAZOOKA = root + "resources/graphics/Weapons/missile.png";
  this->PATH_AIR_STRIKE = root + "resources/graphics/Weapons/airmisil.png";
  this->PATH_CLUSTER = root + "resources/graphics/Weapons/cluster.png";
  this->PATH_MORTAR = root + "resources/graphics/Weapons/mortar.png";
  /* ----------------------------------- */

  /* WORM PATHS */
  this->PATH_PLAIN_WORM = root + "resources/graphics/Worms/plain_worm.png";
  this->PATH_WORM_BREATH_1 = root + "resources/graphics/Worms/wbrth1.png";
  this->PATH_WORM_BREATH_1_UP = root + "resources/graphics/Worms/wbrth1u.png";
  this->PATH_WORM_BREATH_1_DOWN = root + "resources/graphics/Worms/wbrth1d.png";
  this->PATH_WORM_WALK = root + "resources/graphics/Worms/wwalk.png";
  this->PATH_WORM_WALK_UP = root + "resources/graphics/Worms/wwalku.png";
  this->PATH_WORM_WALK_DOWN = root + "resources/graphics/Worms/wwalkd.png";
  this->PATH_WORM_ROLL = root + "resources/graphics/Worms/wroll.png";
  this->PATH_WORM_FALL_DN = root + "resources/graphics/Worms/wfall.png";
  this->PATH_WORM_JUMP = root + "resources/graphics/Worms/wjump.jpg";
  this->PATH_WORM_FLYING_1 = root + "resources/graphics/Worms/wfly1.png";
  this->PATH_WORM_FLYING_2 = root + "resources/graphics/Worms/wfly2.png";
  this->PATH_WORM_FLYING_3 = root + "resources/graphics/Worms/wfly3.png";
  this->PATH_WORM_DIE = root + "resources/graphics/Worms/wdie.png";

    // Pick weapons
    this->PATH_WORM_PICK_BAZOOKA = root + "resources/graphics/Worms/wbazlnk.png";
    this->PATH_WORM_PICK_BAZOOKA_UP = root + "resources/graphics/Worms/wbazlnku.png";
    this->PATH_WORM_PICK_BAZOOKA_DOWN = root + "resources/graphics/Worms/wbazlnkd.png";
    this->PATH_WORM_PICK_MORTAR = root + "resources/graphics/Worms/wbz2lnk.png";;
    this->PATH_WORM_PICK_MORTAR_UP = root + "resources/graphics/Worms/wbz2lnku.png";;
    this->PATH_WORM_PICK_MORTAR_DOWN = root + "resources/graphics/Worms/wbz2lnkd.png";;
    this->PATH_WORM_PICK_GREEN_GRENADE = root + "resources/graphics/Worms/wgrnlnk.png";
    this->PATH_WORM_PICK_GREEN_GRENADE_UP = root + "resources/graphics/Worms/wgrnlnku.png
";
    this->PATH_WORM_PICK_GREEN_GRENADE_DOWN = root + "resources/graphics/Worms/wgrnlnkd.p
ng";
    this->PATH_WORM_PICK_CLUSTER = root + "resources/graphics/Worms/wclslnk.png";
    this->PATH_WORM_PICK_CLUSTER_UP = root + "resources/graphics/Worms/wclslnku.png";
    this->PATH_WORM_PICK_CLUSTER_DOWN = root + "resources/graphics/Worms/wclslnkd.png";
    this->PATH_WORM_PICK_BANANA = root + "resources/graphics/Worms/wbanlnk.png";
    this->PATH_WORM_PICK_BANANA_UP = root + "resources/graphics/Worms/wbanlnku.png";
    this->PATH_WORM_PICK_BANANA_DOWN = root + "resources/graphics/Worms/wbanlnkd.png";
    this->PATH_WORM_PICK_HOLY_GRENADE = root + "resources/graphics/Worms/whgrlnk.png";
    this->PATH_WORM_PICK_HOLY_GRENADE_UP = root + "resources/graphics/Worms/whgrlnku.png"
;
    this->PATH_WORM_PICK_HOLY_GRENADE_DOWN = root + "resources/graphics/Worms/whgrlnkd.pn
g";
    this->PATH_WORM_PICK_AIR_STRIKE = root + "resources/graphics/Worms/wairlnk.png";
    this->PATH_WORM_PICK_AIR_STRIKE_UP = root + "resources/graphics/Worms/wairbaku.png";
    this->PATH_WORM_PICK_AIR_STRIKE_DOWN = root + "resources/graphics/Worms/wairbakd.png"
;
    this->PATH_WORM_PICK_DYNAMITE = root + "resources/graphics/Worms/wdynlnk.png";
    this->PATH_WORM_PICK_DYNAMITE_UP = root + "resources/graphics/Worms/wdynlnku.png";
    this->PATH_WORM_PICK_DYNAMITE_DOWN = root + "resources/graphics/Worms/wdynlnkd.png";
    this->PATH_WORM_PICK_BASEBALL = root + "resources/graphics/Worms/wbsblnk.png";
    this->PATH_WORM_PICK_BASEBALL_UP = root + "resources/graphics/Worms/wbsblnku.png";
    this->PATH_WORM_PICK_BASEBALL_DOWN = root + "resources/graphics/Worms/wbsblnkd.png";
    this->PATH_WORM_PICK_TELEPORT = root + "resources/graphics/Worms/wtellnk.png";
    this->PATH_WORM_PICK_TELEPORT_UP = root + "resources/graphics/Worms/wtellnku.png";
    this->PATH_WORM_PICK_TELEPORT_DOWN = root + "resources/graphics/Worms/wtellnkd.png";

    // Hide weapons
    this->PATH_WORM_HIDE_BAZOOKA = root + "resources/graphics/Worms/wbazbak.png";
    this->PATH_WORM_HIDE_BAZOOKA_UP = root + "resources/graphics/Worms/wbazbaku.png";
    this->PATH_WORM_HIDE_BAZOOKA_DOWN = root + "resources/graphics/Worms/wbazbakd.png";
    this->PATH_WORM_HIDE_MORTAR = root + "resources/graphics/Worms/wbz2bak.png";
    this->PATH_WORM_HIDE_MORTAR_UP = root + "resources/graphics/Worms/wbz2baku.png";
    this->PATH_WORM_HIDE_MORTAR_DOWN = root + "resources/graphics/Worms/wbz2bakd.png";
    this->PATH_WORM_HIDE_GREEN_GRENADE = root + "resources/graphics/Worms/wgrnbak.png";
    this->PATH_WORM_HIDE_GREEN_GRENADE_UP = root + "resources/graphics/Worms/wgrnbaku.png
";
    this->PATH_WORM_HIDE_GREEN_GRENADE_DOWN = root + "resources/graphics/Worms/wgrnbakd.
png";
    this->PATH_WORM_HIDE_CLUSTER = root + "resources/graphics/Worms/wclsbak.png";
    this->PATH_WORM_HIDE_CLUSTER_UP = root + "resources/graphics/Worms/wclsbaku.png";
    this->PATH_WORM_HIDE_CLUSTER_DOWN = root + "resources/graphics/Worms/wclsbakd.png";
    this->PATH_WORM_HIDE_BANANA = root + "resources/graphics/Worms/wbanbak.png";
    this->PATH_WORM_HIDE_BANANA_UP = root + "resources/graphics/Worms/wbanbaku.png";
    this->PATH_WORM_HIDE_BANANA_DOWN = root + "resources/graphics/Worms/wbanbakd.png";
    this->PATH_WORM_HIDE_HOLY_GRENADE = root + "resources/graphics/Worms/whgrbak.png";
    this->PATH_WORM_HIDE_HOLY_GRENADE_UP = root + "resources/graphics/Worms/whgrbaku.png"
;
    this->PATH_WORM_HIDE_HOLY_GRENADE_DOWN = root + "resources/graphics/Worms/whgrbakd.pn
g";
    this->PATH_WORM_HIDE_DYNAMITE = root + "resources/graphics/Worms/wdynbak.png";
    this->PATH_WORM_HIDE_DYNAMITE_UP = root + "resources/graphics/Worms/wdynbaku.png";
    this->PATH_WORM_HIDE_DYNAMITE_DOWN = root + "resources/graphics/Worms/wdynbakd.png";
    this->PATH_WORM_HIDE_AIR_STRIKE = root + "resources/graphics/Worms/wairbak.png";
    this->PATH_WORM_HIDE_AIR_STRIKE_UP = root + "resources/graphics/Worms/wairbaku.png";
    this->PATH_WORM_HIDE_AIR_STRIKE_DOWN = root + "resources/graphics/Worms/wairbakd.png"
;
    this->PATH_WORM_HIDE_BASEBALL = root + "resources/graphics/Worms/wbsbbak.png";
    this->PATH_WORM_HIDE_BASEBALL_UP = root + "resources/graphics/Worms/wbsbbaku.png";
    this->PATH_WORM_HIDE_BASEBALL_DOWN = root + "resources/graphics/Worms/wbsbbakd.png";
    this->PATH_WORM_HIDE_BASEBALL_SHOOTED = root + "resources/graphics/Worms/wbsbbk2.png
";
    this->PATH_WORM_HIDE_BASEBALL_SHOOTED_UP = root + "resources/graphics/Worms/wbsbbk2
u.png";
    this->PATH_WORM_HIDE_BASEBALL_SHOOTED_DOWN = root + "resources/graphics/Worms/wbsbb
k2d.png";
    this->PATH_WORM_HIDE_TELEPORT = root + "resources/graphics/Worms/wtelbak.png";
    this->PATH_WORM_HIDE_TELEPORT_UP = root + "resources/graphics/Worms/wtelbaku.png";
    this->PATH_WORM_HIDE_TELEPORT_DOWN = root + "resources/graphics/Worms/wtelbakd.png";

    // Pointing
    this->PATH_WORM_POINTING_BAZOOKA = root + "resources/graphics/Worms/wbaz.png";
    this->PATH_WORM_POINTING_BAZOOKA_UP = root + "resources/graphics/Worms/wbazu.png";
    this->PATH_WORM_POINTING_BAZOOKA_DOWN = root + "resources/graphics/Worms/wbazd.png";
    this->PATH_WORM_POINTING_MORTAR = root + "resources/graphics/Worms/wbaz2.png";
    this->PATH_WORM_POINTING_MORTAR_UP = root + "resources/graphics/Worms/wbaz2u.png";
    this->PATH_WORM_POINTING_MORTAR_DOWN = root + "resources/graphics/Worms/wbaz2d.png";

    this->PATH_WORM_POINTING_GREEN_GRENADE = root + "resources/graphics/Worms/wthrgrn.png
";
    this->PATH_WORM_POINTING_GREEN_GRENADE_UP = root + "resources/graphics/Worms/wthrgrn
u.png";
    this->PATH_WORM_POINTING_GREEN_GRENADE_DOWN = root + "resources/graphics/Worms/wthr
grnd.png";
    this->PATH_WORM_POINTING_CLUSTER = root + "resources/graphics/Worms/wthrcls.png";
    this->PATH_WORM_POINTING_CLUSTER_UP = root + "resources/graphics/Worms/wthrclsu.png";
```

```
    this->PATH_WORM_POINTING_CLUSTER_DOWN = root + "resources/graphics/Worms/wthrclsd.png"
;
    this->PATH_WORM_POINTING_BANANA = root + "resources/graphics/Worms/wthrban.png";
    this->PATH_WORM_POINTING_BANANA_UP = root + "resources/graphics/Worms/wthrbanu.png";
    this->PATH_WORM_POINTING_BANANA_DOWN = root + "resources/graphics/Worms/wthrband.png"
;
    this->PATH_WORM_POINTING_HOLY_GRENADE = root + "resources/graphics/Worms/wthrhgr.png"
;
    this->PATH_WORM_POINTING_HOLY_GRENADE_UP = root + "resources/graphics/Worms/wthrhgr.p
ng";
    this->PATH_WORM_POINTING_HOLY_GRENADE_DOWN = root + "resources/graphics/Worms/wthrhg
r.png";
    this->PATH_WORM_POINTING_BASEBALL = root + "resources/graphics/Worms/wbsbaim.png";
    this->PATH_WORM_POINTING_BASEBALL_UP = root + "resources/graphics/Worms/wbsbaimu.png
";
    this->PATH_WORM_POINTING_BASEBALL_DOWN = root + "resources/graphics/Worms/wbsbaimd.p
ng";

    // Shooting
    this->PATH_WORM_SHOOTING_AIR_STRIKE = root + "resources/graphics/Worms/wairtlk.png";
    this->PATH_WORM_SHOOTING_AIR_STRIKE_UP = root + "resources/graphics/Worms/wairtlku.png
";
    this->PATH_WORM_SHOOTING_AIR_STRIKE_DOWN = root + "resources/graphics/Worms/wairtlkd.
png";
    this->PATH_WORM_SHOOTING_BASEBALL = root + "resources/graphics/Worms/wbsbswn.png";
    this->PATH_WORM_SHOOTING_BASEBALL_UP = root + "resources/graphics/Worms/wbsbswnu.png
";
    this->PATH_WORM_SHOOTING_BASEBALL_DOWN = root + "resources/graphics/Worms/wbsbswnd.p
ng";

  /* ----------------------------------- */

  /* EFFECTS */
  this->PATH_EXPLOSION_EFFECT = root + "resources/graphics/Effects/firehit.png";
  this->PATH_DEFAULT_SIGHT = root + "resources/graphics/Misc/crshairr.png";
  this->PATH_WIND_LEFT = root + "resources/graphics/Misc/windl.png";
  this->PATH_WIND_RIGHT = root + "resources/graphics/Misc/windr.png";
  this->PATH_GRAVE_1 = root + "resources/graphics/Misc/grave1.png";
  this->PATH_GRAVE_2 = root + "resources/graphics/Misc/grave2.png";
  this->PATH_GRAVE_3 = root + "resources/graphics/Misc/grave3.png";
  this->PATH_GRAVE_4 = root + "resources/graphics/Misc/grave4.png";
  this->PATH_GRAVE_5 = root + "resources/graphics/Misc/grave5.png";
  this->PATH_GRAVE_6 = root + "resources/graphics/Misc/grave6.png";
  this->PATH_SAVE_ICON = root + "resources/graphics/Misc/save_icon.png";
  this->PATH_EXIT_ICON = root + "resources/graphics/Misc/exit_icon.png";
  /* ---------------------------- */

  /* FONTS */
  this->PATH_FONT_WORM_DATA = root + "resources/fonts/arial.ttf";
  this->PATH_FONT_GROBOLD = root + "resources/fonts/GROBOLD.ttf";
  this->PATH_FONT_VERDANA_BOLD = root + "resources/fonts/verdanab.ttf";
  this->PATH_FONT_ARIAL_BOLD = root + "resources/fonts/arialb.ttf";

  /* SOUND_EFFECTS */
  this->PATH_MUSIC_DEFAULT = root + "resources/sounds/music.mp3";
  this->PATH_SOUND_DYNAMITE = root + "resources/sounds/Effects/FUSE.WAV";
  this->PATH_SOUND_GIRDER = root + "resources/sounds/Effects/GIRDERIMPACT.WAV";
  this->PATH_SOUND_TELEPORT = root + "resources/sounds/Effects/TELEPORT.WAV";
  this->PATH_SOUND_THROW_PROJECTIL = root + "resources/sounds/Effects/THROWRELEASE.WAV
";
  this->PATH_SOUND_HOLY = root + "resources/sounds/Effects/HOLYGRENADE.WAV";
  this->PATH_SOUND_TIME_TRICK = root + "resources/sounds/Effects/TIMERTICK.WAV";
  this->PATH_SOUND_THROW_POWER_UP = root + "resources/sounds/Effects/THROWPOWERUP.WAV
";
  this->PATH_SOUND_WORM_WALKING = root + "resources/sounds/Effects/Walk−Expand.wav";
  this->PATH_SOUND_WORM_WALKING_EXPAND = root + "resources/sounds/Effects/Walk−Compress.wa
v";
  this->PATH_SOUND_AIR_STRIKE = root + "resources/sounds/Effects/Airstrike.wav";

    // Worms voices
    this->PATH_SOUND_LAUGH = root + "resources/sounds/Voices/Spanish/LAUGH.WAV";
    this->PATH_SOUND_FATALITY = root + "resources/sounds/Voices/Spanish/FATALITY.WAV";
    this->PATH_SOUND_HURRY = root + "resources/sounds/Voices/Spanish/HURRY.WAV";
    this->PATH_SOUND_BYE = root + "resources/sounds/Voices/Spanish/BYEBYE.WAV";
    this->PATH_SOUND_DIE = root + "resources/sounds/Voices/Spanish/OHDEAR.WAV";
    this->PATH_SOUND_BEGIN_TURN = root + "resources/sounds/Voices/Spanish/YESSIR.WAV";
    this->PATH_SOUND_JUMP_1 = root + "resources/sounds/Voices/Spanish/JUMP1.WAV";
    this->PATH_SOUND_JUMP_2 = root + "resources/sounds/Voices/Spanish/JUMP2.WAV";

    // Worms hit
    this->PATH_SOUND_NOOO = root + "resources/sounds/Voices/Spanish/NOOO.WAV";
    this->PATH_SOUND_OOFF_1 = root + "resources/sounds/Voices/Spanish/OOFF1.WAV";
    this->PATH_SOUND_OOFF_2 = root + "resources/sounds/Voices/Spanish/OOFF2.WAV";
    this->PATH_SOUND_OOFF_3 = root + "resources/sounds/Voices/Spanish/OOFF3.WAV";
    this->PATH_SOUND_OW_1 = root + "resources/sounds/Voices/Spanish/OW1.WAV";
    this->PATH_SOUND_OW_2 = root + "resources/sounds/Voices/Spanish/OW2.WAV";
    this->PATH_SOUND_OW_3 = root + "resources/sounds/Voices/Spanish/OW3.WAV";
    // About to explode
    this->PATH_SOUND_WHAT_THE = root + "resources/sounds/Voices/Spanish/WHATTHE.WAV";
    this->PATH_SOUND_UH_OH = root + "resources/sounds/Voices/Spanish/UH−OH.WAV";
    this->PATH_SOUND_TAKE_COVER = root + "resources/sounds/Voices/Spanish/TAKECOVER.WAV
";
    this->PATH_SOUND_RUN_AWAY = root + "resources/sounds/Voices/Spanish/RUNAWAY.WAV";

  // Explosions
  this->PATH_SOUND_EXPLOSION_1 = root + "resources/sounds/Effects/Explosion1.wav";
  this->PATH_SOUND_EXPLOSION_2 = root + "resources/sounds/Effects/Explosion2.wav";
  this->PATH_SOUND_EXPLOSION_3 = root + "resources/sounds/Effects/Explosion3.WAV";
  /* −−−−−−−−−−−−−− */
}
```

```
#ifndef __PATHS_H__
#define __PATHS_H__

#include <string>
#include <sys/stat.h>

class Paths {
  private:
    std::string root;

  public:
    /* GIRDERS PATHS */
    // Long
    std::string PATH_GIRDER_LONG_90;
    std::string PATH_GIRDER_LONG_60;
    std::string PATH_GIRDER_LONG_45;
    std::string PATH_GIRDER_LONG_30;
    std::string PATH_GIRDER_LONG_0;
    std::string PATH_GIRDER_LONG_NEGATIVE_30;
    std::string PATH_GIRDER_LONG_NEGATIVE_45;
    std::string PATH_GIRDER_LONG_NEGATIVE_60;
    std::string PATH_GIRDER_LONG_NEGATIVE_90;

    // Short
    std::string PATH_GIRDER_SHORT_90;
    std::string PATH_GIRDER_SHORT_60;
    std::string PATH_GIRDER_SHORT_45;
    std::string PATH_GIRDER_SHORT_30;
    std::string PATH_GIRDER_SHORT_0;
    std::string PATH_GIRDER_SHORT_NEGATIVE_30;
    std::string PATH_GIRDER_SHORT_NEGATIVE_45;
    std::string PATH_GIRDER_SHORT_NEGATIVE_60;
    std::string PATH_GIRDER_SHORT_NEGATIVE_90;
    /* −−−−−−−−−−−−−−−−−−−−−−−−−−−−−− */

    /* WORM PATHS */
    std::string PATH_PLAIN_WORM;
    std::string PATH_WORM_BREATH_1;
    std::string PATH_WORM_BREATH_1_UP;
    std::string PATH_WORM_BREATH_1_DOWN;
    std::string PATH_WORM_WALK;
    std::string PATH_WORM_WALK_UP;
    std::string PATH_WORM_WALK_DOWN;
    std::string PATH_WORM_ROLL;
    std::string PATH_WORM_FALL_DN;
    std::string PATH_WORM_JUMP;
    std::string PATH_WORM_FLYING_1;
    std::string PATH_WORM_FLYING_2;
    std::string PATH_WORM_FLYING_3;
    std::string PATH_WORM_DIE;

      // Pick weapons
      std::string PATH_WORM_PICK_BAZOOKA;
      std::string PATH_WORM_PICK_BAZOOKA_UP;
      std::string PATH_WORM_PICK_BAZOOKA_DOWN;
      std::string PATH_WORM_PICK_MORTAR;
      std::string PATH_WORM_PICK_MORTAR_UP;
      std::string PATH_WORM_PICK_MORTAR_DOWN;
      std::string PATH_WORM_PICK_GREEN_GRENADE;
      std::string PATH_WORM_PICK_GREEN_GRENADE_UP;
      std::string PATH_WORM_PICK_GREEN_GRENADE_DOWN;
      std::string PATH_WORM_PICK_CLUSTER;
      std::string PATH_WORM_PICK_CLUSTER_UP;
      std::string PATH_WORM_PICK_CLUSTER_DOWN;
      std::string PATH_WORM_PICK_BANANA;
      std::string PATH_WORM_PICK_BANANA_UP;
      std::string PATH_WORM_PICK_BANANA_DOWN;
      std::string PATH_WORM_PICK_HOLY_GRENADE;
      std::string PATH_WORM_PICK_HOLY_GRENADE_UP;
      std::string PATH_WORM_PICK_HOLY_GRENADE_DOWN;
      std::string PATH_WORM_PICK_DYNAMITE;
      std::string PATH_WORM_PICK_DYNAMITE_UP;
      std::string PATH_WORM_PICK_DYNAMITE_DOWN;
      std::string PATH_WORM_PICK_AIR_STRIKE;
      std::string PATH_WORM_PICK_AIR_STRIKE_UP;
      std::string PATH_WORM_PICK_AIR_STRIKE_DOWN;
      std::string PATH_WORM_PICK_BASEBALL;
      std::string PATH_WORM_PICK_BASEBALL_UP;
      std::string PATH_WORM_PICK_BASEBALL_DOWN;
      std::string PATH_WORM_PICK_TELEPORT;
      std::string PATH_WORM_PICK_TELEPORT_UP;
      std::string PATH_WORM_PICK_TELEPORT_DOWN;

      // Hide weapons
      std::string PATH_WORM_HIDE_BAZOOKA;
      std::string PATH_WORM_HIDE_BAZOOKA_UP;
      std::string PATH_WORM_HIDE_BAZOOKA_DOWN;
      std::string PATH_WORM_HIDE_MORTAR;
      std::string PATH_WORM_HIDE_MORTAR_UP;
      std::string PATH_WORM_HIDE_MORTAR_DOWN;
      std::string PATH_WORM_HIDE_GREEN_GRENADE;
      std::string PATH_WORM_HIDE_GREEN_GRENADE_UP;
      std::string PATH_WORM_HIDE_GREEN_GRENADE_DOWN;
      std::string PATH_WORM_HIDE_CLUSTER;
      std::string PATH_WORM_HIDE_CLUSTER_UP;
      std::string PATH_WORM_HIDE_CLUSTER_DOWN;
      std::string PATH_WORM_HIDE_BANANA;
      std::string PATH_WORM_HIDE_BANANA_UP;
      std::string PATH_WORM_HIDE_BANANA_DOWN;
      std::string PATH_WORM_HIDE_HOLY_GRENADE;
      std::string PATH_WORM_HIDE_HOLY_GRENADE_UP;
      std::string PATH_WORM_HIDE_HOLY_GRENADE_DOWN;
      std::string PATH_WORM_HIDE_DYNAMITE;
      std::string PATH_WORM_HIDE_DYNAMITE_UP;
      std::string PATH_WORM_HIDE_DYNAMITE_DOWN;
      std::string PATH_WORM_HIDE_AIR_STRIKE;
      std::string PATH_WORM_HIDE_AIR_STRIKE_UP;
      std::string PATH_WORM_HIDE_AIR_STRIKE_DOWN;
      std::string PATH_WORM_HIDE_BASEBALL;
      std::string PATH_WORM_HIDE_BASEBALL_UP;
      std::string PATH_WORM_HIDE_BASEBALL_DOWN;
      std::string PATH_WORM_HIDE_BASEBALL_SHOOTED;
      std::string PATH_WORM_HIDE_BASEBALL_SHOOTED_UP;
      std::string PATH_WORM_HIDE_BASEBALL_SHOOTED_DOWN;
      std::string PATH_WORM_HIDE_TELEPORT;
      std::string PATH_WORM_HIDE_TELEPORT_UP;
      std::string PATH_WORM_HIDE_TELEPORT_DOWN;

      // Pointing
      std::string PATH_WORM_POINTING_BAZOOKA;
      std::string PATH_WORM_POINTING_BAZOOKA_UP;
      std::string PATH_WORM_POINTING_BAZOOKA_DOWN;
      std::string PATH_WORM_POINTING_MORTAR;
      std::string PATH_WORM_POINTING_MORTAR_UP;
```

```
      std::string PATH_WORM_POINTING_MORTAR_DOWN;
      std::string PATH_WORM_POINTING_GREEN_GRENADE;
      std::string PATH_WORM_POINTING_GREEN_GRENADE_UP;
      std::string PATH_WORM_POINTING_GREEN_GRENADE_DOWN;
      std::string PATH_WORM_POINTING_CLUSTER;
      std::string PATH_WORM_POINTING_CLUSTER_UP;
      std::string PATH_WORM_POINTING_CLUSTER_DOWN;
      std::string PATH_WORM_POINTING_BANANA;
      std::string PATH_WORM_POINTING_BANANA_UP;
      std::string PATH_WORM_POINTING_BANANA_DOWN;
      std::string PATH_WORM_POINTING_HOLY_GRENADE;
      std::string PATH_WORM_POINTING_HOLY_GRENADE_UP;
      std::string PATH_WORM_POINTING_HOLY_GRENADE_DOWN;
      std::string PATH_WORM_POINTING_BASEBALL;
      std::string PATH_WORM_POINTING_BASEBALL_UP;
      std::string PATH_WORM_POINTING_BASEBALL_DOWN;

      // Shooting
      std::string PATH_WORM_SHOOTING_AIR_STRIKE;
      std::string PATH_WORM_SHOOTING_AIR_STRIKE_UP;
      std::string PATH_WORM_SHOOTING_AIR_STRIKE_DOWN;
      std::string PATH_WORM_SHOOTING_BASEBALL;
      std::string PATH_WORM_SHOOTING_BASEBALL_UP;
      std::string PATH_WORM_SHOOTING_BASEBALL_DOWN;

    /* ----------------------------------- */

    /* WATER PATHS */
    std::string PATH_WATER_DEFAULT;
    std::string PATH_WATER_2;
    std::string PATH_WATER_3;
    std::string PATH_LAVA;
    /* --------------------------- */


    /* WEAPON ICONS */
    std::string PATH_ICON_BAZOOKA;
    std::string PATH_ICON_MORTAR;
    std::string PATH_ICON_GREEN_GRENADE;
    std::string PATH_ICON_RED_GRENADE;
    std::string PATH_ICON_BANANA;
    std::string PATH_ICON_HOLY_GRENADE;
    std::string PATH_ICON_DYNAMITE;
    std::string PATH_ICON_BASEBALL;
    std::string PATH_ICON_AIR_STRIKE;
    std::string PATH_ICON_TELEPORT;
    std::string PATH_ICON_SHORT_GIRDER;
    std::string PATH_ICON_LONG_GIRDER;
    /* ------------- */

    /* WEAPONS */
    std::string PATH_DYNAMITE;
    std::string PATH_GREEN_GRENADE;
    std::string PATH_HOLY_GRENADE;
    std::string PATH_BANANA;
    std::string PATH_BAZOOKA;
    std::string PATH_AIR_STRIKE;
    std::string PATH_CLUSTER;
    std::string PATH_MORTAR;
    /* ---------------------------- */

    /* EFFECTS */
    std::string PATH_EXPLOSION_EFFECT;
    std::string PATH_DEFAULT_SIGHT;
    std::string PATH_WIND_LEFT;
    std::string PATH_WIND_RIGHT;
    std::string PATH_GRAVE_1;
    std::string PATH_GRAVE_2;
    std::string PATH_GRAVE_3;
    std::string PATH_GRAVE_4;
    std::string PATH_GRAVE_5;
    std::string PATH_GRAVE_6;
    std::string PATH_SAVE_ICON;
    std::string PATH_EXIT_ICON;

    /* ---------------------------- */

    /* FONTS */
    std::string PATH_FONT_WORM_DATA;
    std::string PATH_FONT_GROBOLD;
    std::string PATH_FONT_VERDANA_BOLD;
    std::string PATH_FONT_ARIAL_BOLD;

    /* SOUND EFFECTS */
    std::string PATH_MUSIC_DEFAULT;
    std::string PATH_SOUND_DYNAMITE;
    std::string PATH_SOUND_GIRDER;
    std::string PATH_SOUND_TELEPORT;
    std::string PATH_SOUND_THROW_PROJECTIL;
    std::string PATH_SOUND_HOLY;
    std::string PATH_SOUND_THROW_POWER_UP;
    std::string PATH_SOUND_WORM_WALKING;
    std::string PATH_SOUND_WORM_WALKING_EXPAND;
    std::string PATH_SOUND_AIR_STRIKE;

    std::string PATH_SOUND_TIME_TRICK;

      // Worms voices
      std::string PATH_SOUND_LAUGH;
      std::string PATH_SOUND_FATALITY;
      std::string PATH_SOUND_HURRY;
      std::string PATH_SOUND_BYE;
      std::string PATH_SOUND_DIE;
      std::string PATH_SOUND_BEGIN_TURN;
      std::string PATH_SOUND_JUMP_1;
      std::string PATH_SOUND_JUMP_2;

        // Worms hit
        std::string PATH_SOUND_NOOO;
        std::string PATH_SOUND_OOFF_1;
        std::string PATH_SOUND_OOFF_2;
        std::string PATH_SOUND_OOFF_3;
        std::string PATH_SOUND_OW_1;
        std::string PATH_SOUND_OW_2;
        std::string PATH_SOUND_OW_3;
        // About to explosions
        std::string PATH_SOUND_WHAT_THE;
        std::string PATH_SOUND_UH_OH;
        std::string PATH_SOUND_TAKE_COVER;
        std::string PATH_SOUND_RUN_AWAY;

    // Explosions
    std::string PATH_SOUND_EXPLOSION_1;
    std::string PATH_SOUND_EXPLOSION_2;
```

```
    std::string PATH_SOUND_EXPLOSION_3;
    /* ----------------------------- */

    Paths(const char * r = NULL);


};

extern Paths gPath;

#endif
```

```
#include <iostream>
#include <cstring>
#include <fstream>
#include <sstream>
#include <arpa/inet.h>
#include "protocol.h"
#include "socket.h"
#include "socket_error.h"
#include "protocol_error.h"
#include "event.h"

#define MSG_PROTOCOL_CLOSE_PEER "El servidor cerrÃ³ el socket. "

Protocol::Protocol(SocketReadWrite socket) : skt(std::move(socket)) {
}

void Protocol::getPlayerName(std::string & name) {
    uint32_t length;

    this->skt.getBuffer((uchar*)&length, 4);
    length = ntohl(length);

    uchar * buffer = new uchar[length+1];

    this->skt.getBuffer(buffer, length);
    std::string ret_name((char*)buffer, (int)length);
    name = ret_name;
    delete[] buffer;
}

void Protocol::sendName(std::string const & name) const {
    uint32_t length = name.size();
    length = htonl(length);
    this->skt.sendBuffer((const uchar*)&length, 4);
    this->skt.sendBuffer((const uchar*)name.c_str(), name.size());
}

void Protocol::sendFile(std::fstream & file) const {
    uint32_t length;

    file.seekg(0, std::ios::end);
    length = file.tellg();
    file.seekg(0, std::ios_base::beg);
    uint32_t net_length = htonl(length);
    this->skt.sendBuffer((const uchar*)&net_length, 4);

    int remain = length;
    int readed = 0;
    uchar file_chop[FILE_TRANSFER_CHOP_SIZE];
    while (remain) {
        if (remain < FILE_TRANSFER_CHOP_SIZE) {
            file.read((char*)file_chop, remain);
            readed = remain;
        } else {
            file.read((char*)file_chop, FILE_TRANSFER_CHOP_SIZE);
            readed = FILE_TRANSFER_CHOP_SIZE;
        }
        remain -= readed;
        this->skt.sendBuffer(file_chop, readed);
    }
}
```

```
void Protocol::rcvFile(std::fstream & file) const {
    uint32_t length = 0;
    this->skt.getBuffer((uchar*)&length, 4);
    length = ntohl(length);

    if (length == 0) {
            std::stringstream msg;
            msg << MSG_PROTOCOL_CLOSE_PEER << std::strerror(errno);
            throw ProtocolError(msg.str());
    }

    int remain = length;
    int received = 0;
    uchar file_chop[FILE_TRANSFER_CHOP_SIZE];
    while (remain) {
        if (remain < FILE_TRANSFER_CHOP_SIZE) {
            received = this->skt.getBuffer(file_chop, remain);
        } else {
            received = this->skt.getBuffer(file_chop, FILE_TRANSFER_CHOP_SIZE);
        }
        remain -= received;
        file.write((char*)file_chop, received);
    }
}

void Protocol::sendExit(void) {
    this->skt.shutDown();
}

void Protocol::rcvGameMap(YAML::Node & mapNode) {
    uint32_t node_size = 0;
    skt.getBuffer((uchar *) &node_size, 4);
    node_size = ntohl(node_size);
    uchar * buffer = new uchar[node_size+1];
    skt.getBuffer(buffer, node_size);
    buffer[node_size] = '\0';
    std::string text_node((char*) buffer);
    delete[] buffer;
    mapNode = YAML::Load(text_node);
}

void Protocol::sendGameMap(YAML::Node & mapNode) {
    std::stringstream map_dump;
    map_dump << mapNode;
    uint32_t node_size = map_dump.str().length();
    uint32_t net_node_size = htonl(node_size);
    skt.sendBuffer((const uchar *) &net_node_size, 4);
    skt.sendBuffer((const uchar *) map_dump.str().c_str(), node_size);
}

void Protocol::sendGameMapAsString(std::stringstream & map_dump) {
    uint32_t size = map_dump.str().length();
    uint32_t net_size = htonl(size);
    skt.sendBuffer((const uchar*) &net_size, 4);
    skt.sendBuffer((const uchar*) map_dump.str().c_str(), size);
}

void Protocol::sendEvent(Event event) {
    YAML::Node nodeEvent = event.getNode();
    this->sendGameMap(nodeEvent);
}
```

```
Event Protocol::rcvEvent(void) {
    YAML::Node eventNode;
    this->rcvGameMap(eventNode);
    Event event(eventNode);
    return event;
}

void Protocol::sendModel(YAML::Node & modelNode) {
    this->sendGameMap(modelNode);
}

void Protocol::rcvModel(YAML::Node & modelNode) {
    this->rcvGameMap(modelNode);
}

void Protocol::sendGameStatus(YAML::Node & gameStatusNode) {
    this->sendGameMap(gameStatusNode);
}

void Protocol::rcvGameStatus(YAML::Node & gameStatusNode) {
    this->rcvGameMap(gameStatusNode);
}

void Protocol::rcvMsg(YAML::Node & msgNode) {
    this->rcvGameMap(msgNode);
}

void Protocol::sendMsg(YAML::Node & msgNode) {
    this->sendGameMap(msgNode);
}
```

```
#include "protocol_error.h"
#include <string>
#include <iostream>

ProtocolError::ProtocolError(const std::string & msg) {
    this->msg = msg;
}

const char * ProtocolError::what(void) const noexcept {
    return this->msg.c_str();
}

ProtocolError::~ProtocolError(void) noexcept {
}
```

```
#ifndef __PROTOCOL_ERROR_H__
#define __PROTOCOL_ERROR_H__

#include <exception>
#include <string>

/*
Clase para generar excepciones del tipo 'protocol'.
*/
class ProtocolError : public std::exception {
    private:
        std::string msg;
    public:
        explicit ProtocolError(const std::string &);
        virtual const char * what(void) const noexcept;
        virtual ~ProtocolError(void) noexcept;
};

#endif
```

```
#ifndef __PROTOCOL_H__
#define __PROTOCOL_H__

#include <string>
#include "socket.h"
#include "yaml.h"
#include "types.h"
#include "event.h"

#define PLAYER_NAME_LENGTH_LIMIT 20
#define FILENAME_LENGTH_LIMIT 255
#define FILE_TRANSFER_CHOP_SIZE 255


class Protocol {
    private:
        SocketReadWrite skt;

    public:
        explicit Protocol(SocketReadWrite);
        void getPlayerName(std::string &);
        void sendName(std::string const &) const;
        void rcvFile(std::fstream & file) const;
        void sendFile(std::fstream & file) const;
        void sendExit(void);
        void rcvGameMap(YAML::Node &);
        void sendGameMap(YAML::Node &);
        void sendEvent(Event);
        Event rcvEvent(void);
        void sendModel(YAML::Node &);
        void rcvModel(YAML::Node &);
        void sendSnapshot(std::string const &) const;
        void rcvSnapshot(std::string &);
        void sendGameStatus(YAML::Node &);
        void rcvGameStatus(YAML::Node &);
        void rcvMsg(YAML::Node &);
        void sendMsg(YAML::Node &);
        void sendGameMapAsString(std::stringstream & map_dump);
};

#endif
```

```
#include "sdl_timer.h"

Timer::Timer() {
  //Initialize the variables
  mStartTicks = 0;
  mPausedTicks = 0;

  mPaused = false;
  mStarted = false;
}

Timer::~Timer() {}

void Timer::start() {
  //Start the timer
  mStarted = true;

  //Unpause the timer
  mPaused = false;

  //Get the current clock time
  mStartTicks = SDL_GetTicks();
        mPausedTicks = 0;
}

void Timer::stop() {
  //Stop the timer
  mStarted = false;

  //Unpause the timer
  mPaused = false;

        //Clear tick variables
        mStartTicks = 0;
        mPausedTicks = 0;
}

void Timer::pause() {
  //If the timer is running and isn't already paused
  if (mStarted && !mPaused) {
    //Pause the timer
    mPaused = true;

    //Calculate the paused ticks
    mPausedTicks = SDL_GetTicks() - mStartTicks;
    mStartTicks = 0;
  }
}

void Timer::unpause() {
  //If the timer is running and paused
  if (mStarted && mPaused) {
    //Unpause the timer
    mPaused = false;

    //Reset the starting ticks
    mStartTicks = SDL_GetTicks() - mPausedTicks;

    //Reset the paused ticks
    mPausedTicks = 0;
  }
}
```

```

size_t Timer::getTicks() {
        //The actual timer time
        size_t time = 0;

  //If the timer is running
  if (mStarted) {
    //If the timer is paused
    if (mPaused) {
      //Return the number of ticks when the timer was paused
      time = mPausedTicks;
    } else {
      //Return the current time minus the start time
      time = SDL_GetTicks() - mStartTicks;
    }
  }

  return time;
}

bool Timer::isStarted() {
        //Timer is running and paused or unpaused
  return mStarted;
}

bool Timer::isPaused() {
        //Timer is running and paused
  return mPaused && mStarted;
}
```

```
#ifndef __SDL_TIMER_H__
#define __SDL_TIMER_H__

#include <SDL2/SDL.h>

// fuente: http://lazyfoo.net/tutorials/SDL/23_advanced_timers/index.php

class Timer {
  private:
    //The clock time when the timer started
    int mStartTicks;

    //The ticks stored when the timer was paused
    int mPausedTicks;

    //The timer status
    bool mPaused;
    bool mStarted;

  public:
    //Initializes variables
    Timer();
    ~Timer();

    //The various clock actions
    void start();
    void stop();
    void pause();
    void unpause();

    //Gets the timer's time
    size_t getTicks();

    //Checks the status of the timer
    bool isStarted();
    bool isPaused();
};

#endif
```

```
#include "socket.h"
#include "socket_error.h"
#include "types.h"
#include <iostream>
#include <string>
#include <cstring>
#include <sstream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>

SocketBase::SocketBase(int sockfd) {
    if (sockfd == SOCKET_INVALID_STATE) {
        std::stringstream msg;
        msg << MSG_SOCKET_ERROR_INVALID_CREATE << " " << sockfd;
        throw SocketError(msg.str());
    }
    this->sockfd = sockfd;
}

int SocketBase::getSockFd(void) const {
    return this->sockfd;
}

void SocketBase::SktClose(void) {
    ::close(this->sockfd);
    this->sockfd = SOCKET_INVALID_STATE;
}

SocketBase::~SocketBase(void) {
    if (this->sockfd != SOCKET_INVALID_STATE) {
        ::shutdown(getSockFd(), SHUT_RDWR);
        SktClose();
    }
}

SocketBase::SocketBase(SocketBase && move) noexcept :
 sockfd(SOCKET_INVALID_STATE) {
    int t = this->sockfd;
    this->sockfd = move.sockfd;
    move.sockfd = t;
}

bool SocketBase::validState(void) const {
    return (this->sockfd == SOCKET_INVALID_STATE ? false : true);
}

SocketReadWrite::SocketReadWrite(int sockfd) : SocketBase(sockfd) {
}

size_t SocketReadWrite::getBuffer(uchar * buffer, int size) const {
    if (getSockFd() == SOCKET_INVALID_STATE) {
        std::stringstream msg;
        msg << MSG_SOCKET_ERROR_INVALID_READ << " " << getSockFd();
        throw SocketError(msg.str());
    }
    bool skt_closed = false;
    int rcv_bytes = 0;
    int rcv_status, remain;
    while (skt_closed == false && rcv_bytes < size) {
        remain = size - rcv_bytes;
        rcv_status =
        ::recv(getSockFd(), &buffer[rcv_bytes], remain, MSG_NOSIGNAL);
        if (rcv_status > 0) {
            rcv_bytes += rcv_status;
        } else if (rcv_status == 0) {
            skt_closed = true;
        } else if (rcv_status == -1) {
            std::stringstream msg;
            msg << MSG_SOCKET_ERROR_READ << " " << rcv_bytes
            << std::strerror(errno);
            throw SocketError(msg.str());
        }
    }

    if (rcv_bytes > 0)
        return rcv_bytes;

    if (skt_closed)
        return 0;

    return 0;
}

void SocketReadWrite::sendBuffer(const uchar * buffer, int size) const {
    if (getSockFd() == SOCKET_INVALID_STATE) {
        std::stringstream msg;
        msg << MSG_SOCKET_ERROR_INVALID_WRITE << " " << getSockFd();
        throw SocketError(msg.str());
    }

    int remain = size;
    int send_status = 0;

    while (remain) {
        send_status =
        ::send(getSockFd(), &buffer[remain-size], remain, MSG_NOSIGNAL);
        if (send_status == -1) {
            std::stringstream msg;
            msg << MSG_SOCKET_ERROR_WRITE << " " << size-remain
            << std::strerror(errno);
            throw SocketError(msg.str());
        } else {
            remain -= send_status;
        }
    }
}

void SocketReadWrite::shutDown(void) {
    int errcode = 0;
    errcode = ::shutdown(getSockFd(), SHUT_RDWR);
    if (errcode) {
        std::stringstream msg;
        throw SocketError(msg.str());
    }
}

int SocketConnection::_get_valid_socket
(std::string const & host, std::string const & port) {
    struct ::addrinfo hints;
    memset(&hints, 0, sizeof(struct ::addrinfo));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
```

```
    hints.ai_flags = 0;

    int errcode = 0;
    errcode = ::getaddrinfo
    (host.c_str(), port.c_str(), &hints, &this->results);
    if (errcode) {
        std::stringstream msg;
        msg << MSG_SOCKET_ERROR_ADDRINFO << std::strerror(errno);
        throw SocketError(msg.str());
    }

    errcode = 0;
    int newsockfd = -1;

    for (this->result = this->results;
    this->result != NULL;
    this->result = this->result->ai_next) {
        newsockfd = ::socket
        (this->result->ai_family, this->result->ai_socktype,
         this->result->ai_protocol);
        if (newsockfd == -1) {
            continue;
        } else {
            break;
        }
    }
    return newsockfd;
}

SocketConnection::SocketConnection
(std::string const & host, std::string const & port)
 : SocketReadWrite(_get_valid_socket(host, port)) {
    int errcode =
    ::connect(getSockFd(), this->result->ai_addr, this->result->ai_addrlen);
    ::freeaddrinfo(this->results);
    this->result = NULL;
    this->results = NULL;

    if (errcode == -1) {
        SktClose();
        std::stringstream msg;
        msg << MSG_SOCKET_ERROR_CONNECT << " " << host << " " << port;
        throw SocketError(msg.str());
    }
}

int SocketListener::_get_valid_socket_listener(std::string const & port) {
    struct ::addrinfo hints;

    memset(&hints, 0, sizeof(struct ::addrinfo));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    int errcode = 0;
    errcode = ::getaddrinfo(NULL, port.c_str(), &hints, &this->result);
    if (errcode) {
        std::stringstream msg;
        msg << MSG_SOCKET_ERROR_ADDRINFO << std::strerror(errno);
        throw SocketError(msg.str());
    }
    return ::socket(this->result->ai_family, this->result->ai_socktype,
    this->result->ai_protocol);
}

SocketListener::SocketListener(std::string const & port) :
SocketBase(_get_valid_socket_listener(port)){
    int skopt = 1;
    ::setsockopt(getSockFd(), SOL_SOCKET, SO_REUSEADDR, &skopt, sizeof(skopt));

    int errcode =
    ::bind(getSockFd(), this->result->ai_addr, this->result->ai_addrlen);
    if (errcode == -1) {
        SktClose();
        std::stringstream msg;
        msg << MSG_SOCKET_ERROR_BIND << " "
        << port << "." << std::strerror(errno);
        throw SocketError(msg.str());
    }

    ::freeaddrinfo(this->result);
    this->result = NULL;

    errcode = ::listen(getSockFd(), SOCKET_LISTENER_MAX_CONNECTION_WAITING);
    if (errcode == -1) {
        SktClose();
        std::stringstream msg;
        msg << MSG_SOCKET_ERROR_LISTEN << " "
        << port << "." << std::strerror(errno);
        throw SocketError(msg.str());
    }
}

SocketReadWrite SocketListener::accept_connection(void) {
    if (getSockFd() == SOCKET_INVALID_STATE) {
        std::stringstream msg;
        msg << MSG_SOCKET_ERROR_ACCEPT_INVALID;
        throw SocketError(msg.str());
    }

    int newsockfd = ::accept(getSockFd(), NULL, NULL);
    if (newsockfd == -1) {
        std::stringstream msg;
        msg << MSG_SOCKET_ERROR_ACCEPT << " " << std::strerror(errno);
        throw SocketError(msg.str());
    }

    return std::move(SocketReadWrite(newsockfd));
}

void SocketListener::stopListening(void) {
    int errcode = 0;
    errcode = ::shutdown(getSockFd(), SHUT_RDWR);
    if (errcode) {
        std::stringstream msg;
        msg << MSG_SOCKET_ERROR_SHUTDOWN << std::strerror(errno);
        throw SocketError(msg.str());
    }
}
```

```
#include "socket_error.h"
#include <string>
#include <iostream>

SocketError::SocketError(const std::string & msg) {
    this->msg = msg;
}

const char * SocketError::what(void) const noexcept {
    return this->msg.c_str();
}

SocketError::~SocketError(void) noexcept {
}
```

```
#ifndef __COMMON_SOCKET_ERROR_H__
#define __COMMON_SOCKET_ERROR_H__

#include <exception>
#include <string>

/*
Clase para generar excepciones del tipo 'socket'.
*/
class SocketError : public std::exception {
    private:
        std::string msg;
    public:
        explicit SocketError(const std::string &);
        virtual const char * what(void) const noexcept;
        virtual ~SocketError(void) noexcept;
};

#endif
```

```
#ifndef __COMMON_SOCKET_H__
#define __COMMON_SOCKET_H__

#include <string>
#include "types.h"

#define SOCKET_INVALID_STATE -1
#define SOCKET_INITIAL_STATE 0
#define SOCKET_LISTENER_MAX_CONNECTION_WAITING 10

#define MSG_SOCKET_ERROR_INVALID_CREATE "Se intenta crear un socket invalido."
#define MSG_SOCKET_ERROR_INVALID_READ "Se intenta leer un socket invalido."
#define MSG_SOCKET_ERROR_READ "−1 en recv() de socket tras recibir (bytes):"
#define MSG_SOCKET_ERROR_INVALID_WRITE "Escritura en un socket invalido."
#define MSG_SOCKET_ERROR_WRITE "−1 en send() de socket tras enviar (bytes):"
#define MSG_SOCKET_ERROR_SHUTDOWN "Error en shutdown de socket:"
#define MSG_SOCKET_ERROR_ADDRINFO "Error en socket getaddrinfo():"
#define MSG_SOCKET_ERROR_CONNECT "No se pudo conectar socket TCP al ip−port:"
#define MSG_SOCKET_ERROR_BIND "Error en bind de socket en puerto: "
#define MSG_SOCKET_ERROR_LISTEN "Error en listen() de socket en puerto:"
#define MSG_SOCKET_ERROR_ACCEPT_INVALID "accept() en un socket invalido."
#define MSG_SOCKET_ERROR_ACCEPT "Error de socket tras un accept():"

typedef struct addrinfo net_addrinfo;

/*
Base para cualquier tipo de socket. Contiene el file descriptor y funciones basi
cas para cualquier socket como
un constructor (por parametro y por movimiento), destructor, cierre y pregunta s
i esta en estado valido (fd != −1).
No es valido copiar un socket, por lo que dichos metodos estan anulados. Cualqui
er tipo de socket heredara de este
socket base.
*/
class SocketBase {
    private:

    protected:
        explicit SocketBase(int fd);
        int getSockFd(void) const;

    public:
        int sockfd;
        SocketBase(SocketBase && move) noexcept;
        virtual ~SocketBase(void);
        void SktClose(void);
        bool validState(void) const;
        explicit SocketBase(SocketBase const&) = delete;
        SocketBase& operator=(SocketBase const&) = delete;
};

/*
Este tipo de socket es un socket que ya esta conectado con un par y tiene los me
todos para enviar y recibir buffers de informacion.
Tambien puede hacer shutdown para indicarle al par que termino de operar. Su con
sturctor llama al constructor base, pasandole un file
descriptor valido.
*/
class SocketReadWrite : public SocketBase {
    private:
    protected:
    public:
        explicit SocketReadWrite(int sockfd);
        size_t getBuffer(uchar * buffer, int size) const;
        void sendBuffer(const uchar * buffer, int size) const;
        void shutDown(void);
};

/*
Este tipo de socket se construira exitosamente cuando pueda conectarse al IP-POR
T que se pasen como parametro. Hereda de
SocketReadWrite por lo que una vez construido (y conectado) puede enviar y recib
ir informacion con su par.
Su metodo _get_valid_socket() itera en los addr results buscando una conexion va
lida, y retorna un file descriptor valido,
luego SocketConnection invoca al constructor de su padre con dicho file descript
or valido, y posteriormente se conecta con el par.
*/
class SocketConnection : public SocketReadWrite {
    private:
        net_addrinfo * result;
        net_addrinfo * results;
        int _get_valid_socket(std::string const &, std::string const &);
    protected:
    public:
        SocketConnection(std::string const &, std::string const &);
};

/*
Este tipo de socket representa un 'socket listener' o 'socket aceptador', propio
 de un servidor. Con la macro SOCKET_LISTENER_MAX_CONNECTION_WAITING
se definen la cantidad de conexiones en espera que puede tener. Una vez construi
do este socket, queda escuchando conexiones en <port>. El metodo
accept_connection() es bloqueante y espera una nueva conexion. Al recibir una co
nexion, retorna con move semantics un socket del tipo SocketReadWrite,
que esta listo para enviar y recibir datos con su par.
*/
class SocketListener : public SocketBase {
    private:
        net_addrinfo * result;
        size_t max_connections_waiting =
        SOCKET_LISTENER_MAX_CONNECTION_WAITING;
        int _get_valid_socket_listener(std::string const &);
    protected:
    public:
        explicit SocketListener(std::string const & port);
        SocketReadWrite accept_connection(void);
        void stopListening(void);
};


#endif
```

```
#include "sound_effect.h"
#include <limits.h>

int GLOBAL_CHANNEL_COUNTER = 1;

SoundEffect::SoundEffect(void) {
  this->sound = NULL;
  this->music = NULL;

  this->playingMusic = false;
  this->playingSound = false;

  if (GLOBAL_CHANNEL_COUNTER == INT_MAX) {
    GLOBAL_CHANNEL_COUNTER = 1;
  }

  this->channel = GLOBAL_CHANNEL_COUNTER++;

  Mix_AllocateChannels(Mix_AllocateChannels(-1) + 1);
}

SoundEffect::~SoundEffect() {
  this->freeSound();
  this->freeMusic();
}

void SoundEffect::freeSound(void) {
  this->playingSound = false;
  Mix_HaltChannel(this->channel);
  if (this->sound) {
    Mix_FreeChunk(this->sound);
    this->sound = NULL;
  }
}

void SoundEffect::freeMusic(void) {
  if (this->music) {
    Mix_HaltChannel(this->channel);
    Mix_FreeMusic(this->music);
    this->music = NULL;
  }
}

void SoundEffect::setSound(std::string path) {
  this->freeSound();
  this->sound = Mix_LoadWAV(path.c_str());
  if (!this->sound) {
    throw View::Exception("%s %s. SDL_Error: %s.", ERR_MSG_LOADING_SOUND, path.c_str
(), Mix_GetError());
  }
}

void SoundEffect::setMusic(std::string path) {
  this->freeMusic();
  this->music = Mix_LoadMUS(path.c_str());
  if (!this->music) {
    throw View::Exception("%s %s. SDL_Error: %s.", ERR_MSG_LOADING_SOUND, path.c_str
(), Mix_GetError());
  }
}

void SoundEffect::playSound(int loops) {
  if (Mix_Playing(this->channel) == 0) {
    this->playingSound = false;
  }

  if (!this->playingSound && this->sound) {
    Mix_PlayChannel(this->channel, this->sound, loops);
    this->playingSound = true;
  }
}

void SoundEffect::stopSound(void) {
  Mix_HaltChannel(this->channel);
}

void SoundEffect::playMusic(int loops) {
  if (!Mix_PlayingMusic()) {
    this->playingMusic = false;
    if (this->music) {
      Mix_PlayMusic(this->music, loops);
      this->playingMusic = true;
    }
  }

  if (Mix_PausedMusic()) {
    Mix_ResumeMusic();
  }
}

void SoundEffect::increaseMusicVolume(int inc) {
  this->musicVolume = Mix_VolumeMusic(-1);
  this->musicVolume + inc > MIX_MAX_VOLUME ? Mix_VolumeMusic(MIX_MAX_VOLUME) : M
ix_VolumeMusic(this->musicVolume + inc);
}

void SoundEffect::decreaseMusicVolume(int dec) {
  this->musicVolume = Mix_VolumeMusic(-1);
  this->musicVolume - dec < 0 ? Mix_VolumeMusic(0) : Mix_VolumeMusic(this->music
Volume - dec);
}

bool SoundEffect::isPlaying(void) {
  return Mix_Playing(this->channel) != 0;
}
```

```
#ifndef __SOUND_EFFECT_H__
#define __SOUND_EFFECT_H__

#include <SDL2/SDL.h>
#include <SDL2/SDL_mixer.h>
#include <string>
#include "view_exceptions.h"

extern int GLOBAL_CHANNEL_COUNTER;

class SoundEffect {
  private:
    Mix_Chunk * sound;
    Mix_Music * music;

    bool playingSound;
    bool playingMusic;

    int channel;
    int musicVolume;

    // Libera la memoria del sonido
    void freeSound(void);

    // Libera la memoria de la musica
    void freeMusic(void);

    void channelFinished(int);

  public:
    // Constructor por defecto
    SoundEffect();

    // Destructor, libera el Mix_Chunk
    ~SoundEffect();

    // Setea los archivos de sonidos
    void setSound(std::string);
    void setMusic(std::string);

    // Reproduce el sonido loops+1
    // veces. -1: reproduce eternamente
    // hasta que se lo detenga.
    void playSound(int loops = -1);

    // Reproduce la musica loops+1
    // veces. -1: reproduce eternamente
    // hasta que se lo detenga.
    void playMusic(int loops = -1);

    // Aumenta el volumen de la musica
    void increaseMusicVolume(int inc = 10);

    // Disminuye el volumen de la musica
    void decreaseMusicVolume(int dec = 10);

    // Para el sonido
    void stopSound(void);

    // Checkea si esta playing
    bool isPlaying(void);
};
```

```
#endif
```

```
#include "thread.h"

Thread::Thread(void) {
}

Thread::Thread(Thread && other_moved) {
    this->thread = std::move(other_moved.thread);
}

Thread & Thread::operator=(Thread && other_copyed) {
    this->thread = std::move(other_copyed.thread);
    return * this;
}

Thread::~Thread(void) {
}

void Thread::join(void) {
    this->thread.join();
}

void Thread::start(void) {
    this->thread = std::thread(&Thread::run, this);
}
```

```
#ifndef __THREAD_HPP__
#define __THREAD_HPP__

#include <thread>

/*
Clase abstracta que encapsula un thread. La idea es que los objetos vivos herede
n de esta clase e implementen los metodos virtuales run() (el mas importante,
ya que 'dispara' el thread), isRunning (que sirve para saber cuando un thread te
rmino de ejecutarse), y getId (esto si deseamos que cada thread tenga un
ID unico, quiza con fines de debugging).
*/
class Thread {
    private:
        std::thread thread;
    public:
        Thread(void);
        Thread(Thread &&);
        Thread & operator=(Thread && other);
        virtual ~Thread(void);
        virtual void run(void) = 0;
        virtual bool isRunning(void) const = 0;
        virtual size_t getId(void) const = 0;
        void join(void);
        void start(void);
        Thread(const Thread &) = delete;
        Thread & operator=(const Thread &) = delete;
};

#endif
```

```
#ifndef __COMMON_TYPES_H__
#define __COMMON_TYPES_H__

/*
 Se definen tipos de datos comunes al cliente y el servidor.
*/
typedef enum {
    a_noEvent,
    a_refreshLobby,
    a_createMatch,
    a_rmWaitingMatch,
    a_joinWaitingMatch,
    a_exitWaitingMatch,
    a_refreshWaitingList,
    a_startMatch,
    a_goToMatch,
    a_quitLobby,
    a_moveLeft,
    a_moveRight,
    a_stopMoving,
    a_pointUp,
    a_pointDown,
    a_pickWeapon,
    a_shoot,
    a_frontJump,
    a_backJump,
    a_selectWeapon,
    a_pointFlyShoot,
    a_choose1SecDeton,
    a_choose2SecDeton,
    a_choose3SecDeton,
    a_choose4SecDeton,
    a_choose5SecDeton,
    a_showWeaponMenu,
    a_changeWorm,
    a_quitGame
} action_t;

typedef enum {
    w_null = -1,
    w_bazooka = 0,
    w_mortar = 1,
    w_cluster = 2,
    w_banana = 4,
    w_green_grenade = 3,
    w_holy_grenade = 5,
    w_dynamite = 7,
    w_air_strike = 6,
    w_bat = 8,
    w_teleport = 9
} weapon_t;

typedef enum {
    quited,
    lobby,
    joined,
    creator,
    on_match
} client_status_t;

typedef enum {
    ALL,
    ONLY_HEALTH,
    NO_DATA
} worm_data_cfg_t;

typedef enum {
    WS_BREATHING,
    WS_WALKING,
    WS_FALLING,
    WS_FLYING,
    WS_PICK_WEAPON,
    WS_DEAD
} view_worm_state_t;

typedef enum {
    NONE,
    UP,
    DOWN
} worm_inclination_t;

typedef enum {
    CAMERA_AUTOMATIC,
    CAMERA_MANUAL
} camera_mode_t;

typedef unsigned char uchar;

#endif
```

Table of Contents