```cpp
#include "air_strike.h"

#define AIR_STRIKE 3

View::AirStrike::AirStrike(SDL_Renderer * r, int ratioExplosion) :
  sprite(AIR_STRIKE, DEPENDENT_ON_GRADES),
        explosion(r, ratioExplosion, "Bazooka") {
  this->texture.loadFromFile(gPath.PATH_AIR_STRIKE, r);
  this->sprite.setSpriteSheet(&this->texture);

        this->exploded = false;
        this->finished = false;

        this->sound.setSound(gPath.PATH_SOUND_AIR_STRIKE);
  this->sound.playSound(0);
        this->ratioExplosion = ratioExplosion;

        this->playedAboutToExplode = false;
}

View::AirStrike::~AirStrike() {

}

void View::AirStrike::render(SDL_Renderer * r, int camX, int camY) {
  if (!this->exploded) {
                // Render AirStrike animation
                SDL_Rect clip = this->sprite.getNextClip(this->angleDirection);
                this->texture.render(r, this->x - camX, this->y - camY, &clip);
        } else {
                if (!this->explosion.finishedExplosion()) {
                        this->explosion.setX(this->x + this->getWidth() / 2);
                        this->explosion.setY(this->y + this->getHeight() / 2);
                        this->explosion.render(r, camX, camY);
                } else {
                        this->finished = true;
                }
        }
}
```



```cpp
#ifndef __AIR_STRIKE_H__
#define __AIR_STRIKE_H__

#include "explosion.h"
#include "rectangle_text.h"
#include "projectil.h"

namespace View {
  class AirStrike: public Projectil {
    private:
      SpriteAnimation sprite;
      Explosion explosion;

    public:
      AirStrike(SDL_Renderer * r, int ratioExplosion = 100);
      ~AirStrike();
      virtual void render(SDL_Renderer * r, int, int);
  };
}

#endif
```

jun 25, 18 20:09 **banana.cpp** Page 1/1

```
#include "banana.h"

#define BANANA_FPC 3

View::Banana::Banana(SDL_Renderer * r, int countdown, int ratioExplosion) :
  sprite(BANANA_FPC, INFINITE_GOING),
        explosion(r, ratioExplosion, "Banana"),
        countdownText(COUNTDOWN_TEXT_SIZE) {
  this->texture.loadFromFile(gPath.PATH_BANANA, r);
  this->sprite.setSpriteSheet(&this->texture);

        this->exploded = false;
        this->finished = false;

        this->sound.setSound(gPath.PATH_SOUND_THROW_PROJECTIL);
  this->sound.playSound(0);
        this->countdown = countdown;
        this->ratioExplosion = ratioExplosion;

        this->playedAboutToExplode = false;
}

View::Banana::~Banana() {

}

void View::Banana::render(SDL_Renderer * r, int camX, int camY) {
  if (!this->exploded) {
                // Render Banana animation
                SDL_Rect clip = this->sprite.getNextClip();
                this->texture.render(r, this->x - camX, this->y - camY, &clip);

                // Render countdown text
                this->countdownText.setText(r, std::to_string(this->countdown));
                this->countdownText.setX(this->x + this->texture.getWidth() +  t
his->countdownText.getWidth() / 2);
                this->countdownText.setY(this->y - this->countdownText.getHeight
() / 2);
                this->countdownText.render(r, camX, camY);
        } else {
                if (!this->explosion.finishedExplosion()) {
                        this->explosion.setX(this->x + this->getWidth() / 2);
                        this->explosion.setY(this->y + this->getHeight() / 2);
                        this->explosion.render(r, camX, camY);
                } else {
                        this->finished = true;
                }
        }
}
```

jun 25, 18 20:09 **banana.h** Page 1/1

```
#ifndef __BANANA_H__
#define __BANANA_H__

#include "explosion.h"
#include "rectangle_text.h"
#include "projectil.h"

namespace View {
  class Banana: public Projectil {
    private:
      SpriteAnimation sprite;
      Explosion explosion;
      RectangleText countdownText;
    public:
      Banana(SDL_Renderer * r, int countdown, int ratioExplosion = 100);
      ~Banana();
      virtual void render(SDL_Renderer * r, int, int);
  };
}

#endif
```

```cpp
#include "bazooka.h"

#define BAZOOKA_FPC 3

View::Bazooka::Bazooka(SDL_Renderer * r, int ratioExplosion, weapon_t weapon) :
  sprite(BAZOOKA_FPC, DEPENDENT_ON_GRADES),
        explosion(r, ratioExplosion, "Bazooka") {
        if (weapon == w_mortar) {
                this->texture.loadFromFile(gPath.PATH_MORTAR, r);
        } else {
                this->texture.loadFromFile(gPath.PATH_BAZOOKA, r);
        }

  this->sprite.setSpriteSheet(&this->texture);

        this->exploded = false;
        this->finished = false;

        this->sound.setSound(gPath.PATH_SOUND_THROW_PROJECTIL);
  this->sound.playSound(0);
        this->ratioExplosion = ratioExplosion;

        this->playedAboutToExplode = false;
}

View::Bazooka::~Bazooka() {

}

void View::Bazooka::render(SDL_Renderer * r, int camX, int camY) {
  if (!this->exploded) {
                // Render Bazooka animation
                SDL_Rect clip = this->sprite.getNextClip(this->angleDirection);
                this->texture.render(r, this->x - camX, this->y - camY, &clip);
        } else {
                if (!this->explosion.finishedExplosion()) {
                        this->explosion.setX(this->x + this->getWidth() / 2);
                        this->explosion.setY(this->y + this->getHeight() / 2);
                        this->explosion.render(r, camX, camY);
                } else {
                        this->finished = true;
                }
        }
}
```



```cpp
#ifndef __BAZOOKA_H__
#define __BAZOOKA_H__

#include "explosion.h"
#include "rectangle_text.h"
#include "projectil.h"
#include "types.h"

namespace View {
  class Bazooka: public Projectil {
    private:
      SpriteAnimation sprite;
      Explosion explosion;

    public:
      Bazooka(SDL_Renderer * r, int ratioExplosion = 100, weapon_t w = w_bazooka
);
      ~Bazooka();
      virtual void render(SDL_Renderer * r, int, int);
  };
}

#endif
```

```
#include "breathing.h"

View::Breathing::Breathing(View::Worm * worm, SDL_Renderer * r) {
  this->state = WS_BREATHING;
  this->context = worm;
  this->textures[NONE].loadFromFile(gPath.PATH_WORM_BREATH_1, r);
  this->textures[UP].loadFromFile(gPath.PATH_WORM_BREATH_1_UP, r);
  this->textures[DOWN].loadFromFile(gPath.PATH_WORM_BREATH_1_DOWN, r);
  this->sprites[NONE].setSpriteSheet(&this->textures[NONE]);
  this->sprites[UP].setSpriteSheet(&this->textures[UP]);
  this->sprites[DOWN].setSpriteSheet(&this->textures[DOWN]);
}

View::Breathing::~Breathing() {

}

void View::Breathing::render(SDL_Renderer * r, int camX, int camY, worm_inclinat
ion_t incl, bool mirrored, int angle) {
  SDL_Rect clip = this->sprites[incl].getNextClip();
  View::Texture & current = this->textures[incl];
  if (mirrored) {
    current.render(
      r,
      this->context->getX() - current.getWidth() / 2 - camX,
      this->context->getY() - current.getWidth() / 2 - camY,
      &clip,
      0,
      NULL,
      SDL_FLIP_HORIZONTAL
    );
  } else {
    current.render(
      r,
      this->context->getX() - current.getWidth() / 2 - camX,
      this->context->getY() - current.getWidth() / 2 - camY,
      &clip
    );
  }
}

void View::Breathing::resetAnimation(void) {
  std::map<worm_inclination_t, SpriteAnimation>::iterator it = this->sprites.beg
in();
  for (; it != this->sprites.end() ; it++) {
    it->second.reset();
  }
}
```



```
#ifndef __BREATHING_H__
#define __BREATHING_H__

#include <map>
#include "worm_state.h"
#include "sprite_animation.h"
#include "texture.h"
#include "worm.h"
#include "types.h"

namespace View {
  class Worm;

  class Breathing: public WormState {
    private:
      std::map<worm_inclination_t, View::Texture> textures;
      std::map<worm_inclination_t, View::SpriteAnimation> sprites;

    public:
      Breathing(View::Worm * context, SDL_Renderer * r);
      ~Breathing();
      virtual void render(SDL_Renderer *, int, int, worm_inclination_t, bool, in
t);
      virtual void resetAnimation(void);
  };
}


#endif
```

```
#include "camera.h"
#include <iostream>

#define OFFSET_NEAR 25
#define MOVE_PER_FRAME 10
#define MAX_MANUAL_QUIET_MS 1500

View::Camera::Camera(int camW, int camH, int levelW, int levelH) :
  width(camW), height(camH), levelWidth(levelW), levelHeight(levelH) {
    // Inicializamos la camara centrada al nivel
    this->camera = {
      (this->levelWidth - this->width) / 2,
      (this->levelHeight - this->height) / 2,
     this->width,
     this->height
    };

    this->movingLeft = false;
    this->movingRight = false;
    this->movingUp = false;
    this->movingDown = false;
    this->mode = CAMERA_AUTOMATIC;
}

View::Camera::~Camera() {}

int View::Camera::getX(void) const {
  return this->camera.x;
}

int View::Camera::getY(void) const {
  return this->camera.y;
}

SDL_Rect View::Camera::getCamera(void) const {
  return this->camera;
}

void View::Camera::setX(int x) {
  if (x < 0) {
    this->camera.x = 0;
    return;
  }

  if (x > this->levelWidth - this->width) {
    this->camera.x = this->levelWidth - this->width;
    return;
  }

  this->camera.x = x;
}

void View::Camera::setY(int y) {
  if (y < 0) {
    this->camera.y = 0;
    return;
  }

  if (y > this->levelHeight - this->height) {
    this->camera.y = this->levelHeight - this->height;
    return;
  }
```



```
  this->camera.y = y;
}

void View::Camera::setXY(int x, int y) {
  this->setX(x);
  this->setY(y);
  return;
}

void View::Camera::focus(const Drawable & d) {
  if (this->mode == CAMERA_MANUAL) {
    if (this->timer.getTicks() > MAX_MANUAL_QUIET_MS) {
      this->mode = CAMERA_AUTOMATIC;
      this->timer.stop();
    }
  }

  if (this->mode == CAMERA_AUTOMATIC) {
    this->setX(d.getX() - this->width / 2);
    this->setY(d.getY() - this->height / 2);
  }
}

void View::Camera::setManualMode(void) {
  this->mode = CAMERA_MANUAL;
  this->restartTimer();
}

void View::Camera::restartTimer(void) {
  this->timer.stop();
  this->timer.start();
}

void View::Camera::handleEvent(SDL_Event & e) {
  if (e.type == SDL_KEYDOWN) {
    if (e.key.keysym.sym == SDLK_LEFT) {
      this->setX(this->camera.x - 25);
      this->setManualMode();
    }
    if (e.key.keysym.sym == SDLK_RIGHT) {
      this->setX(this->camera.x + 25);
      this->setManualMode();
    }
    if (e.key.keysym.sym == SDLK_UP) {
      this->setY(this->camera.y - 25);
      this->setManualMode();
    }
    if (e.key.keysym.sym == SDLK_DOWN) {
      this->setY(this->camera.y + 25);
      this->setManualMode();
    }
  }
}

void View::Camera::updateCameraPosition(void) {
  int mouseX, mouseY;
  SDL_GetMouseState(&mouseX, &mouseY);

  if (
    mouseX > 0 + OFFSET_NEAR &&
    mouseX < this->width - OFFSET_NEAR &&
```

```
    mouseY > 0 + OFFSET_NEAR &&
    mouseY < this->height - OFFSET_NEAR
  ) {
    this->movingLeft = false;
    this->movingRight = false;
    this->movingUp = false;
    this->movingDown = false;
  } else {
    if (mouseX < 0 + OFFSET_NEAR || this->movingLeft) {
      this->setX(this->camera.x - MOVE_PER_FRAME);
      this->movingLeft = true;
      this->setManualMode();
    }
    if (mouseX > this->width - OFFSET_NEAR || this->movingRight) {
      this->setX(this->camera.x + MOVE_PER_FRAME);
      this->movingRight = true;
      this->setManualMode();
    }
    if (mouseY < 0 + OFFSET_NEAR || this->movingUp) {
      this->setY(this->camera.y - MOVE_PER_FRAME);
      this->movingUp = true;
      this->setManualMode();
    }
    if (mouseY > this->height -OFFSET_NEAR || this->movingDown) {
      this->setY(this->camera.y + MOVE_PER_FRAME);
      this->movingDown = true;
      this->setManualMode();
    }
    return;
  }
}
```



```
#ifndef __CAMERA_H__
#define __CAMERA_H__

#include <SDL2/SDL.h>
#include "drawable.h"
#include "sdl_timer.h"
#include "types.h"

namespace View {
  class Camera {
    private:
      SDL_Rect camera;
      camera_mode_t mode;
      Timer timer;

      // Dimensiones de la camara
      int width;
      int height;

      // Dimensiones del nivel
      int levelWidth;
      int levelHeight;

      bool movingLeft;
      bool movingRight;
      bool movingUp;
      bool movingDown;

      void setManualMode(void);
      void restartTimer(void);

    public:
      Camera(int camW, int camH, int levelW, int levelH);
      ~Camera();

      // Getters de la posicion de la camara
      int getX(void) const;
      int getY(void) const;

      // Getter del rectangulo de la camara
      SDL_Rect getCamera(void) const;

      // Seters de la posicion de la camara
      void setX(int);
      void setY(int);
      void setXY(int, int);

      // Centra la camara en un dibujable
      void focus(const Drawable &);

      // Mueve la camara si recibe el evento correspondiente
      void handleEvent(SDL_Event &);

      // Actualiza la posicion de la camara
      // respecto de donde
      // esta ubicado el mouse
      void updateCameraPosition(void);
  };
}

#endif
```

```
#include "client_configuration.h"

#define CHOP_ANGLE 3
#define MAX_SIGHT_ANGLE 90
#define MIN_SIGHT_ANGLE -90
#define MAX_TIME_SHOOTING 1500

#define VIEW_SHOOT_POWER_WIDTH 300
#define VIEW_SHOOT_POWER_HEIGHT 50

#define SCREEN_PADDING 10

#define MAX_WEAPONS 10
#define SCREEN_PERCENT_CLOCK 10
#define SCREEN_PERCENT_INVENTORY 50
#define SCREEN_PERCENT_SHOOT_POWER_HEIGHT 5
#define SCREEN_PERCENT_SHOOT_POWER_WIDTH 30

#define SCREEN_PERCENT_WIND_HEIGHT 5
#define SCREEN_PERCENT_WIND_WIDTH 30

#define SCREEN_PERCENT_TEAMS_HEALTH_HEIGHT 10
#define SCREEN_PERCENT_TEAMS_HEALTH_WIDTH 20

ClientConfiguration::ClientConfiguration(SDL_Renderer * r, int screenW, int scre
enH, const YAML::Node & staticMap, size_t teamId) :
  notice(screenW, screenH),
  renderer(r),
  teamId(teamId),
  shootPower(
    screenW / (100 / SCREEN_PERCENT_SHOOT_POWER_WIDTH),
    screenH / (100 / SCREEN_PERCENT_SHOOT_POWER_HEIGHT),
    MAX_TIME_SHOOTING
  ),
  clock(
    screenH / (100 / SCREEN_PERCENT_CLOCK),
    screenH / (100 / SCREEN_PERCENT_CLOCK)
  ),
  inventory(
    r,
    staticMap["init_inventory"]
  ),
  wind(
    r,
    screenW / (100 / SCREEN_PERCENT_WIND_WIDTH),
    screenH / (100 / SCREEN_PERCENT_WIND_HEIGHT)
  ),
  teamsHealth(
    r,
    screenW / (100 / SCREEN_PERCENT_TEAMS_HEALTH_WIDTH),
    screenH / (100 / SCREEN_PERCENT_TEAMS_HEALTH_HEIGHT),
    staticMap["teams_amount"].as<int>(),
    staticMap["worms_health"].as<int>(),
    staticMap["max_worms"].as<int>()
  ) {

  int clockX = SCREEN_PADDING + this->clock.getWidth() / 2;
  int clockY = screenH - SCREEN_PADDING - this->clock.getHeight() / 2;
  this->clock.setX(clockX);
  this->clock.setY(clockY);

  int shootX = screenW - SCREEN_PADDING - this->shootPower.getWidth() / 2;
```



```
  int shootY = screenH - SCREEN_PADDING - this->shootPower.getHeight() / 2 - thi
s->wind.getHeight() - SCREEN_PADDING;
  this->shootPower.setX(shootX);
  this->shootPower.setY(shootY);

  int windX = screenW - SCREEN_PADDING - this->wind.getWidth() / 2;
  int windY = screenH - SCREEN_PADDING - this->wind.getHeight() / 2;
  this->wind.setX(windX);
  this->wind.setY(windY);

  int teamsHealthX = SCREEN_PADDING + this->clock.getWidth() + SCREEN_PADDING +
this->teamsHealth.getWidth() / 2;
  int teamsHealthY = screenH - SCREEN_PADDING - this->teamsHealth.getHeight() /
2;
  this->teamsHealth.setX(teamsHealthX);
  this->teamsHealth.setY(teamsHealthY);

  this->inventory.setIconSide(screenH / (100 / SCREEN_PERCENT_INVENTORY) / MAX_W
EAPONS);

  this->sightAngle = 0;
  this->weaponsCountdown = 5;
  this->wormDataConfig = ALL;
  this->shooting = false;
  this->shooted = false;
  this->powerShoot = -1;
  this->shootingSound.setSound(gPath.PATH_SOUND_THROW_POWER_UP);

  this->remoteControlX = 0;
  this->remoteControlY = 0;

  this->wormProtagonicId = 1;

  this->beginTurn.setSound(gPath.PATH_SOUND_BEGIN_TURN);
  this->beginTurnPlayed = false;

  this->music.setMusic(gPath.PATH_MUSIC_DEFAULT);
  this->music.playMusic();
}

ClientConfiguration::~ClientConfiguration() {

}

void ClientConfiguration::handleEvent(SDL_Event & e) {
  this->inventory.handleEvent(e);
  weapon_t weapon = this->inventory.getSelectedWeapon();

  if (e.type == SDL_MOUSEBUTTONDOWN) {
    if (e.button.button == SDL_BUTTON_LEFT) {
      if (weapon == w_air_strike || weapon == w_teleport) {
        this->shooted = true;
        SDL_GetMouseState(&this->remoteControlX, &this->remoteControlY);
      }
    }
  }

  if (e.type == SDL_KEYDOWN) {
    SDL_Keycode code = e.key.keysym.sym;
    if (code == SDLK_PLUS || code == SDLK_KP_PLUS) {
      this->music.increaseMusicVolume();
    }
```

```
  if (code == SDLK_MINUS || code == SDLK_KP_MINUS) {
    this->music.decreaseMusicVolume();
  }

              if (code == SDLK_1) {
                      this->weaponsCountdown = 1;
    this->notice.showFlashNotice(this->renderer, "Countdown set to 1 second");
              }

              if (code == SDLK_2) {
                      this->weaponsCountdown = 2;
    this->notice.showFlashNotice(this->renderer, "Countdown set to 2 seconds");
              }

              if (code == SDLK_3) {
                      this->weaponsCountdown = 3;
    this->notice.showFlashNotice(this->renderer, "Countdown set to 3 seconds");
              }

              if (code == SDLK_4) {
                      this->weaponsCountdown = 4;
    this->notice.showFlashNotice(this->renderer, "Countdown set to 4 seconds");
              }

              if (code == SDLK_5) {
                      this->weaponsCountdown = 5;
    this->notice.showFlashNotice(this->renderer, "Countdown set to 5 seconds");
              }

  if (code == SDLK_w) {
    if (this->sightAngle + CHOP_ANGLE > MAX_SIGHT_ANGLE) {
      this->sightAngle = MAX_SIGHT_ANGLE;
    } else {
      this->sightAngle += CHOP_ANGLE;
    }
  }

  if (code == SDLK_s) {
    if (this->sightAngle - CHOP_ANGLE < MIN_SIGHT_ANGLE) {
      this->sightAngle = MIN_SIGHT_ANGLE;
    } else {
      this->sightAngle -= CHOP_ANGLE;
    }
        }

  if (code == SDLK_DELETE) {
    if (this->wormDataConfig == ALL) {
      this->wormDataConfig = ONLY_HEALTH;
      return;
    }

    if (this->wormDataConfig == ONLY_HEALTH) {
      this->wormDataConfig = NO_DATA;
      return;
    }

    if (this->wormDataConfig == NO_DATA) {
      this->wormDataConfig = ALL;
      return;
    }
  }
```



```
    if (code == SDLK_h) {
      this->teamsHealth.toggleHide();
    }

    if (code == SDLK_SPACE && weapon != w_air_strike && weapon != w_teleport) {
      if (!this->shootingTimer.isStarted()) {
        this->shootingSound.playSound(0);
        this->shootingTimer.start();
        this->shooting = true;
      }
    }
  }

  if (this->shootingTimer.isStarted()) {
    if (this->shootingTimer.getTicks() >= MAX_TIME_SHOOTING) {
      this->shootingSound.stopSound();
      this->powerShoot = MAX_TIME_SHOOTING;
      this->shootingTimer.stop();
      this->shooting = false;
      this->shooted = true;
    }
  }

  if (e.type == SDL_KEYUP) {
    SDL_Keycode code = e.key.keysym.sym;
    if (code == SDLK_SPACE && this->shooting) {
      this->shootingSound.stopSound();
      this->shooting = false;
      this->shooted = true;
      this->powerShoot = this->shootingTimer.getTicks();
      this->shootingTimer.stop();
    }
  }
}

int ClientConfiguration::getWeaponsCountdown(void) const {
  return this->weaponsCountdown;
}

bool ClientConfiguration::hasShooted(void) const {
  return this->shooted;
}

int ClientConfiguration::getPowerShoot(void) {
  int pshoot = this->powerShoot;
  this->shooted = false;
  this->powerShoot = -1;
  return pshoot;
}

void ClientConfiguration::render(SDL_Renderer * r) {
  if (this->shooting) {
    this->shootPower.render(r, this->shootingTimer.getTicks());
  }

  this->notice.render(r);

  this->wind.render(r, 0, 0);

  this->inventory.render(r);
  this->clock.render(r, 0, 0);
```

```
  this->teamsHealth.render(r, 0, 0);
}

weapon_t ClientConfiguration::getSelectedWeapon(void) {
  return this->inventory.getSelectedWeapon();
}

void ClientConfiguration::update(const YAML::Node & gameStatus, const YAML::Node
 & inventory) {
  size_t teamWithTurn = gameStatus["team_turn"].as<size_t>();
  if (this->teamId == teamWithTurn && !this->beginTurnPlayed) {
    this->beginTurn.playSound(0);
    this->beginTurnPlayed = true;
  }

  if (this->teamId != teamWithTurn && this->beginTurnPlayed) {
    this->beginTurnPlayed = false;
  }

  int newTime = gameStatus["turn_timeleft"].as<int>();
  int windForce = gameStatus["wind_force"].as<int>();
  this->wormProtagonicId = gameStatus["protagonic_worm"].as<int>();
  const YAML::Node & teamsHealthNode = gameStatus["teams_health"];

  if (newTime) {
    this->clock.toggleHide(false);
  } else {
    this->clock.toggleHide(true);
  }

  this->clock.setTime(newTime);
  this->wind.setWindPower(windForce);
  this->teamsHealth.update(teamsHealthNode);
  this->inventory.update(inventory);
}

int ClientConfiguration::getSightAngle(void) {
  return this->sightAngle;
}

worm_data_cfg_t ClientConfiguration::getWormDataConfiguration(void) {
  return this->wormDataConfig;
}

int ClientConfiguration::getRemoteControlX(void) {
  this->shooted = false;
  return this->remoteControlX;
}

int ClientConfiguration::getRemoteControlY(void) {
  this->shooted = false;
  return this->remoteControlY;
}

size_t ClientConfiguration::getWormProtagonicId(void) {
  return this->wormProtagonicId;
}
```



```
#ifndef __CLIENT_CONFIGURATION_H__
#define __CLIENT_CONFIGURATION_H__

#include <iostream>
#include <SDL2/SDL.h>
#include "clock.h"
#include "inventory_weapons.h"
#include "types.h"
#include "sdl_timer.h"
#include "sound_effect.h"
#include "shoot_power.h"
#include "paths.h"
#include "teams_health.h"
#include "wind.h"
#include "yaml.h"
#include "flash_notice.h"

class ClientConfiguration {
  private:
    worm_data_cfg_t wormDataConfig;
    int weaponsCountdown;
    int sightAngle;
    int powerShoot;
    int remoteControlX;
    int remoteControlY;
    bool shooting;
    bool shooted;
    bool beginTurnPlayed;
    size_t wormProtagonicId;
    size_t teamId;
    Timer shootingTimer;
    SoundEffect shootingSound;
    View::ShootPower shootPower;
    View::Clock clock;
    View::WeaponsInventory inventory;
    View::Wind wind;
    View::TeamsHealth teamsHealth;
    SDL_Renderer * renderer;
    FlashNotice notice;

    SoundEffect beginTurn;
    SoundEffect music;

  public:
    ClientConfiguration(SDL_Renderer *, int, int, const YAML::Node &, size_t);
    ~ClientConfiguration();

    void handleEvent(SDL_Event &);

    int getWeaponsCountdown(void) const;
    bool hasShooted(void) const;
    int getPowerShoot(void);
    void render(SDL_Renderer *);

    // Retorna el weapon_t seleccionado
    weapon_t getSelectedWeapon(void);

    // Retorna el worm_data_cfg_t configurado
    worm_data_cfg_t getWormDataConfiguration(void);

    // Retorna el sight_angle configurado
    int getSightAngle(void);
```

```
    // Retorna la posicion X/Y del teledirigido
    int getRemoteControlX(void);
    int getRemoteControlY(void);

    // Updatea la configuracion que muestra el cliente
    void update(const YAML::Node &, const YAML::Node &);

    // Devuelve el id del worm protagonico
    size_t getWormProtagonicId(void);
};

#endif
```



```
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_mixer.h>
#include <qt5/QtWidgets/QMessageBox>
#include <fstream>
#include <string>
#include <sstream>
#include <unistd.h>
#include "client_game.h"
#include "protocol.h"
#include "event_sender.h"
#include "model_receiver.h"
#include "window_game.h"
#include "camera.h"
#include "socket.h"
#include "socket_error.h"
#include "protocol_error.h"
#include "blocking_queue.h"
#include "sdl_timer.h"
#include "client_configuration.h"
#include "types.h"
#include "yaml.h"
#include "worms_status.h"
#include "projectiles.h"
#include "inventory.h"
#include "client_settings.h"

#define TIE_GAME_CODE 0
#define CONSTANT_WAIT 100/6
#define MAX_QUEUE_MODELS 256
#define MAP_RECEIVED_NAME "/usr/etc/worms/temp/map.tar.gz"
#define MAP_YML_PATH "/usr/etc/worms/temp/map.yml"

extern ClientSettings gClientSettings;

ClientGame::ClientGame(Protocol * prt, size_t tid) :
protocol(prt),
events(MAX_QUEUE_MODELS),
team_id(tid) {
        std::string map_received_name(MAP_RECEIVED_NAME);
        std::fstream file_map(map_received_name, std::fstream::out | std::fstrea
m::binary | std::fstream::trunc);
        std::cout << "Esperando mapa del sevidor." << std::endl;
    this->protocol->rcvFile(file_map);
        std::cout << "Mapa recibido del servidor." << std::endl;
        file_map.close();
        std::string cmd_unzip_tar_gz = "tar -xf " + map_received_name + " -C /usr/etc/w
orms/temp";
        std::system(cmd_unzip_tar_gz.c_str());
        this->mapNode = YAML::LoadFile(MAP_YML_PATH);
        this->creator = false;
}

ClientGame::~ClientGame(void) {
        removeTempFiles();
}

ClientGame::ClientGame(Protocol * prt, size_t tid, std::string & mp) :
protocol(prt),
events(MAX_QUEUE_MODELS),
```

```
team_id(tid) {
        std::string cmd_unzip_tar_gz = "tar -xf " + mp + " -C /usr/etc/worms/temp";
        std::system(cmd_unzip_tar_gz.c_str());
        this->mapNode = YAML::LoadFile(MAP_YML_PATH);
        this->creator = true;
}

void ClientGame::removeTempFiles(void) {
        if (!this->creator) {
                std::string map_received_name(MAP_RECEIVED_NAME);
                std::string cmd_rm_map_yml = "rm /usr/etc/worms/temp/map.yml /usr/etc/worms/t
emp/background.png " + map_received_name;
                std::system(cmd_rm_map_yml.c_str());
        } else {
                std::string cmd_rm_map_yml = "rm /usr/etc/worms/temp/map.yml /usr/etc/worms/t
emp/background.png";
                std::system(cmd_rm_map_yml.c_str());
        }
}

void ClientGame::startGame(void) {
        EventSender event_sender(this->protocol, events);
        YAML::Node staticMap = this->mapNode["static"];
        YAML::Node dynamicMap = this->mapNode["dynamic"];
        YAML::Node wormsNode = dynamicMap["worms_teams"];

        ProtectedDynamics pdynamics(dynamicMap);
        ModelReceiver model_receiver(this->protocol, pdynamics);

        // Creo la pantalla con dichas cosas estÃ¡ticas.
        View::WindowGame mainWindow(staticMap, gClientSettings.RESOLUTION_WIDTH,

        gClientSettings.RESOLUTION_HIGH, gClientSettings.FULL_SCREEN);
        SDL_Renderer * renderer = mainWindow.getRenderer();
        View::Camera camera(mainWindow.getScreenWidth(), mainWindow.getScreenHei
ght(),
                                                mainWindow.getBgWidth(), mainWin
dow.getBgHeight());

        ClientConfiguration cfg(
                renderer,
                mainWindow.getScreenWidth(),
                mainWindow.getScreenHeight(),
                staticMap,
                this->team_id
        );

    View::WormsStatus worms(wormsNode, renderer);
        // Lanzo threads de enviar eventos y de recibir modelos
        event_sender.start();
        model_receiver.start();

        std::cout << "Iniciando game loop." << std::endl;
    gameLoop(camera, mainWindow, renderer, pdynamics, worms, cfg);
        std::cout << "Fin de game loop." << std::endl;

        // Salimos del ciclo del juego, enviamos evento de que nos fuimos.;

        Event event(a_quitGame, this->team_id);
        this->events.push(event);
        usleep(1000000);
        // Stop y Join de threads
```



```
        event_sender.stop();
        event_sender.join();

        model_receiver.stop();
        model_receiver.join();

}

void ClientGame::gameLoop(View::Camera & camera, View::WindowGame & mainWindow,
SDL_Renderer * renderer,
ProtectedDynamics & pdynamics, View::WormsStatus & worms, ClientConfiguration &
cfg) try {
        bool quit = false;
        SDL_Event e;
        int timeLostSleeping = 0;
        int ti;
        int tf;
        int updateCount = 0;
        int renderCount = 0;
        bool defeated_msg_showed = false;

    View::Projectiles projectiles;
        bool match_finished = false;
        while (!quit && !match_finished) {
                ti = SDL_GetTicks();
                while (SDL_PollEvent(&e) != 0) {
                        if (e.type == SDL_QUIT)
                                quit = true;

                        cfg.handleEvent(e);

                        if (cfg.hasShooted()) {
                                weapon_t weapon = cfg.getSelectedWeapon();
                                if (weapon != w_air_strike && weapon != w_telepo
rt) {
                                        Event event(a_shoot, weapon, this->team_
id, cfg.getWeaponsCountdown(), cfg.getPowerShoot(), cfg.getSightAngle());
                                        this->events.push(event);
                                } else {
                                        Event event(a_shoot, weapon, this->team_
id, cfg.getRemoteControlX() + camera.getX(), cfg.getRemoteControlY() + camera.ge
tY());
                                        this->events.push(event);
                                }
                        }

                        if (e.type == SDL_KEYDOWN) {
                                if (e.key.keysym.sym == SDLK_w) {
                                        Event event(a_pointUp, this->team_id);
                                        this->events.push(event);
                                }
                                if (e.key.keysym.sym == SDLK_s) {
                                        Event event(a_pointDown, this->team_id);
                                        this->events.push(event);
                                }
                                if (e.key.keysym.sym == SDLK_a) {
                                        Event event(a_moveLeft, this->team_id);
                                        this->events.push(event);
                                }
                                if (e.key.keysym.sym == SDLK_d) {
                                        Event event(a_moveRight, this->team_id);
                                        this->events.push(event);
```

```
                                }
                                if (e.key.keysym.sym == SDLK_RETURN) {
                                        Event event(a_frontJump, this->team_id);
                                        this->events.push(event);
                                }
                                if (e.key.keysym.sym == SDLK_BACKSPACE) {
                                        Event event(a_backJump, this->team_id);
                                        this->events.push(event);
                                }

                                camera.handleEvent(e);
                        }
                }

                camera.updateCameraPosition();

                const View::Projectil * projectilProtagonic = projectiles.getPro
jectilProtagonic();
                const View::Worm * protagonicWorm = worms.getWormView(cfg.getWor
mProtagonicId());
                const View::Worm * wormAffectedByExplosion = worms.getWormAffect
edByExplosion();
                if (wormAffectedByExplosion) {
                        camera.focus(*wormAffectedByExplosion);
                }else if (projectilProtagonic) {
                        camera.focus(*projectilProtagonic);
                } else if (protagonicWorm) {
                        camera.focus(*protagonicWorm);
                }

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

                // Dibujamos cosas estÃ¡ticas
                mainWindow.render(camera);

                // Procesamiento de los snapshoots
                bool thereIsModel = pdynamics.popModel();
                while (thereIsModel) {
                        updateCount++;
                        worms.update(pdynamics.getWorms());
                        worms.updateWormProtagonic(pdynamics.getWormProtagonicId
());
                        projectiles.update(renderer, pdynamics.getProjectiles())
;
                        thereIsModel = pdynamics.popModel();
                }
                match_finished  = pdynamics.finishedMatch();
                if (match_finished) {
                        size_t team_winner = pdynamics.getWinnerTeam();
                        if (team_winner == this->team_id) {
                                QMessageBox msgBox;
                                msgBox.setWindowTitle("Ganaste.");
                                msgBox.setText("Sos el rey de los gusanos!");
                                msgBox.exec();
                        } else if (team_winner == TIE_GAME_CODE) {
                                QMessageBox msgBox;
                                msgBox.setWindowTitle("Empate.");
                                msgBox.setText("Fue una partida muy reÃ±ida!");
                                msgBox.exec();
                        } else {
                                QMessageBox msgBox;
```



```
                                msgBox.setWindowTitle("Partida terminada.");
                                msgBox.setText("TerminÃ³ la partida. Una pena que hayas perdi
do.");
                                msgBox.exec();
                        }
                        return;
                }

                renderCount++;

                if (pdynamics.hasGameStatus()) {
                        cfg.update(pdynamics.getGameStatus(), pdynamics.getTeamI
nventory(this->team_id));
                        worms.updateWormsClientConfiguration(cfg);
                        if (!defeated_msg_showed && pdynamics.teamDefeated(this-
>team_id)) {
                                QMessageBox msgBox;
                                msgBox.setWindowTitle("Perdiste.");
                                msgBox.setText("Tu equipo ha perdido. Â¡MÃ¡s suerte la prÃ³xi
ma!");
                                msgBox.exec();
                                defeated_msg_showed = true;
                        }
                }

                // Dibujamos cosas dinÃ¡micas
                // Gusanos
                worms.render(renderer, camera);

                // Proyectiles
                projectiles.render(renderer, camera);

                // El agua va sobre todo menos el inventario
                mainWindow.renderWater(camera);

                // Dibujamos los objetos propios del cliente
                cfg.render(renderer);

                SDL_RenderPresent(renderer);
                tf = SDL_GetTicks();

                int to_sleep = CONSTANT_WAIT - (tf-ti) - timeLostSleeping;
                if (to_sleep < 0) {
                        timeLostSleeping = 0;
                } else {
                        SDL_Delay(to_sleep);
                        timeLostSleeping = SDL_GetTicks() - (tf + to_sleep);
                }
        }
} catch(const SocketError & e) {
        std::cout << e.what() << std::endl;
} catch(const std::exception & e) {
                std::cout << e.what() << std::endl;
}
```

jun 25, 18 20:09 **client_game.h** Page 1/1

```
#ifndef __CLIENT_GAME_H__
#define __CLIENT_GAME_H__

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_mixer.h>
#include "protocol.h"
#include "blocking_queue.h"
#include "event_sender.h"
#include "camera.h"
#include "window_game.h"
#include "worms_status.h"
#include "protected_dynamics.h"
#include <string>

class ClientGame {
    private:
        Protocol * protocol;
        Queue<Event> events;
        size_t team_id;
        YAML::Node mapNode;
        bool creator;

        void removeTempFiles(void);

    public:
        ClientGame(Protocol *, size_t, std::string &);
        ClientGame(Protocol *, size_t);
        ~ClientGame(void);
        void startGame(void);
        void gameLoop(View::Camera &, View::WindowGame &, SDL_Renderer *, Protec
tedDynamics &, View::WormsStatus &, ClientConfiguration &);
};

#endif
```

jun 25, 18 20:09 **client_lobby.cpp** Page 1/8

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <QtGui/QCloseEvent>
#include <qt5/QtWidgets/QMessageBox>
#include <QFileDialog>
#include <QTableWidget>
#include "QStackedWidget"
#include "client_lobby.h"
#include "ui_clientlobby.h"
#include "socket.h"
#include "socket_error.h"
#include "event.h"
#include "types.h"
#include "waiting_match.h"
#include "client_game.h"
#include "client_settings.h"

#define PAGE_CONNECTION_INDEX 0
#define PAGE_LOBBY_INDEX 1
#define PAGE_MATCH_CREATE 2
#define PAGE_WAITING_MATCH_INDEX 3
#define PAGE_JOINED_WAITING_MATCH_INDEX 4

extern ClientSettings gClientSettings;

ClientLobby::ClientLobby(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::ClientLobby)
{
    ui->setupUi(this);
    connectEvents();
    this->pages = findChild<QStackedWidget*>(WIDGET_PAGES);
    this->pages->setCurrentIndex(PAGE_CONNECTION_INDEX);
    this->protocol = nullptr;
    this->waiting_match = nullptr;
}

ClientLobby::~ClientLobby()
{
    cleanLobby();
    if (this->protocol != nullptr) {
        delete this->protocol;
    }
    delete ui;
}

void ClientLobby::connectEvents(void) {
    QPushButton* cleanTextBoxes = findChild<QPushButton*>(WIDGET_BUTTON_CLEAN_LO
GIN);
    QObject::connect(cleanTextBoxes, &QPushButton::clicked,
                     this, &ClientLobby::cleanTextBoxes);

    QPushButton* connectButton = findChild<QPushButton*>(WIDGET_BUTTON_CONNECT_L
OGIN);
    QObject::connect(connectButton, &QPushButton::clicked,
                     this, &ClientLobby::connectToServer);

    QPushButton* createButton = findChild<QPushButton*>(WIDGET_BUTTON_CREATE_GAM
E);
    QObject::connect(createButton, &QPushButton::clicked,
                     this, &ClientLobby::createMatch);
```

```
    QPushButton* exitLobbyButton = findChild<QPushButton*>(WIDGET_BUTTON_EXIT_LO
BBY);
    QObject::connect(exitLobbyButton, &QPushButton::clicked,
                     this, &ClientLobby::exitLobby);

    QPushButton* joinButton = findChild<QPushButton*>(WIDGET_BUTTON_JOIN_GAME);
    QObject::connect(joinButton, &QPushButton::clicked,
                     this, &ClientLobby::joinMatch);

    QPushButton* refreshButton = findChild<QPushButton*>(WIDGET_BUTTON_REFRESH_L
OBBY);
    QObject::connect(refreshButton, &QPushButton::clicked,
                     this, &ClientLobby::refreshLobby);

    QPushButton* createGameButton = findChild<QPushButton*>(WIDGET_BUTTON_START_
WAITING_MATCH);
    QObject::connect(createGameButton, &QPushButton::clicked,
                     this, &ClientLobby::waitForPlayersOnCreatedMatch);

    QPushButton* backLobbyButton = findChild<QPushButton*>(WIDGET_BUTTON_BACK_TO
_LOBBY);
    QObject::connect(backLobbyButton, &QPushButton::clicked,
                     this, &ClientLobby::backLobby);

    QPushButton* chooseMapsFolderButton = findChild<QPushButton*>(WIDGET_BUTTON_
CHOOSE_MAP);
    QObject::connect(chooseMapsFolderButton, &QPushButton::clicked,
                     this, &ClientLobby::chooseMap);

    QPushButton* refreshWaitingPlayersButton = findChild<QPushButton*>(WIDGET_BU
TTON_REFRESH_WAITING_PLAYERS);
    QObject::connect(refreshWaitingPlayersButton, &QPushButton::clicked,
                     this, &ClientLobby::feedWaitingPlayers);;

    QPushButton* startWaitingMatchButton = findChild<QPushButton*>(WIDGET_BUTTON
_START_MATCH);
    QObject::connect(startWaitingMatchButton, &QPushButton::clicked,
                     this, &ClientLobby::startWaitingMatch);

    QPushButton* cancelWaitingMatchButton = findChild<QPushButton*>(WIDGET_BUTTO
N_CANCEL_WAITING_MATCH);
    QObject::connect(cancelWaitingMatchButton, &QPushButton::clicked,
                     this, &ClientLobby::cancelWaitingMatch);

    QPushButton* exitWaitingMatchButton = findChild<QPushButton*>(WIDGET_BUTTON_
EXIT_WAITING_MATCH);
    QObject::connect(exitWaitingMatchButton, &QPushButton::clicked,
                     this, &ClientLobby::exitWaitingMatch);
}

void ClientLobby::cleanTextBoxes(void) {
    /* std::cout << "Clear Text." << std::endl; */
    QLineEdit* playerNameText = findChild<QLineEdit*>(WIDGET_TEXT_PLAYER_NAME);
    playerNameText->clear();
    QLineEdit* ipText = findChild<QLineEdit*>(WIDGET_TEXT_IP);
    ipText->clear();
    QLineEdit* portText = findChild<QLineEdit*>(WIDGET_TEXT_PORT);
    portText->clear();
}

void ClientLobby::connectToServer(void) {
```



```
    QLineEdit* ipText = findChild<QLineEdit*>(WIDGET_TEXT_IP);
    QLineEdit* portText = findChild<QLineEdit*>(WIDGET_TEXT_PORT);
    QLineEdit* pnameText = findChild<QLineEdit*>(WIDGET_TEXT_PLAYER_NAME);
    if (ipText->text().isEmpty() || portText->text().isEmpty() || pnameText->tex
t().isEmpty()) {
        return;
    }
    std::string ip = ipText->displayText().toUtf8().constData();
    std::string port = portText->displayText().toUtf8().constData();
    this->player_name = pnameText->displayText().toUtf8().constData();
    /* std::cout << "El usuario intenta conectarse a " << ip << ":" << port << s
td::endl; */

    try {
        SocketConnection skt(ip, port);
        this->protocol = new Protocol(std::move(skt));
        /* std::cout << "Socket creado para conexiÃ³n con servidor " << skt.sock
fd << std::endl; */
    } catch (const SocketError & e) {
        std::cout << e.what() << std::endl;
        QMessageBox msgBox;
        msgBox.setWindowTitle("Error de conexiÃ³n.");
        msgBox.setText("No se pudo conectar con el servidor. Por favor, chequee ip y puerto.");
        msgBox.exec();
        return;
    }
    /* std::cout << "Conexion con servidor establecida" << std::endl; */

    goLobby();
}

void ClientLobby::goLobby(void) {
    this->protocol->sendName(this->player_name);
    // Recibe el nuevo nombre (si hubo colisiÃ³n) que le asigna el servidor.
    this->protocol->getPlayerName(this->player_name);
    /* std::cout << "El servidor me bautizÃ³ como: " << this->player_name << std
::endl; */
    QLabel* playerName = findChild<QLabel*>(WIDGET_TEXT_PLAYER_NAME_LOBBY);
    playerName->setText(this->player_name.c_str());

    this->pages->setCurrentIndex(PAGE_LOBBY_INDEX);

    feedLobby();
}

void ClientLobby::createMatch(void) {
    this->pages->setCurrentIndex(PAGE_MATCH_CREATE);
}

void ClientLobby::exitLobby(void) {
    /* std::cout << "Me voy del lobby" << std::endl; */
    cleanLobby();
    this->pages->setCurrentIndex(PAGE_CONNECTION_INDEX);
    Event new_event(a_quitLobby, 1);
    this->protocol->sendEvent(new_event);
    delete this->protocol;
    this->protocol = nullptr;
}

void ClientLobby::joinMatch(void) {
    /* std::cout << "Me uno a una partida!" << std::endl; */
    QTableWidget * matchsList = findChild<QTableWidget*>(WIDGET_GAMES_TABLE);
```

```
    int index_selected = matchsList->selectionModel()->currentIndex().row();

    if (index_selected < 0) {
        QMessageBox msgBox;
        msgBox.setWindowTitle("Seleccione una partida.");
        msgBox.setText("Por favor, seleccione una partida de la lista.");
        msgBox.exec();
        return;
    }

    std::string match_creator_name;
    match_creator_name = matchsList->item(index_selected,1)->text().toUtf8().con
stData();
   /*  std::cout << "El creador de la partida a joinearse es: " << match_creator
_name << std::endl; */

    Event new_event(a_joinWaitingMatch, match_creator_name);
    this->protocol->sendEvent(new_event);

    YAML::Node join_match_response;
    this->protocol->rcvMsg(join_match_response);
    if (join_match_response["code"].as<int>() == 0) {
        refreshLobby();
        /* std::cout << "La partida esta llena, no se puede acceder." << std::en
dl; */
        std::string msg = join_match_response["msg"].as<std::string>();
        QMessageBox msgBox;
        msgBox.setWindowTitle("No se puede unir a partida:");
        msgBox.setText(msg.c_str());
        msgBox.exec();
        return;
    } else if (join_match_response["code"].as<int>() == 1) {
        /* std::cout << "Hay lugar en la partida, accediendo!." << std::endl; */
        this->pages->setCurrentIndex(PAGE_JOINED_WAITING_MATCH_INDEX);
        QLabel* gameCreator = findChild<QLabel*>(WIDGET_TEXT_GAME_CREATOR);
        gameCreator->setText(match_creator_name.c_str());

        this->waiting_match = new WaitingMatch(this->protocol, this->pages);
        this->waiting_match->start();

    }
}

void ClientLobby::refreshLobby(void) {
    /* std::cout << "Refresh del lobby" << std::endl; */
    cleanLobby();
    /* std::cout << "Lobby limpiado" << std::endl; */
    Event new_event(a_refreshLobby, 1);
    this->protocol->sendEvent(new_event);
    feedLobby();
}

void ClientLobby::waitForPlayersOnCreatedMatch(void) {
    QLineEdit * matchName = findChild<QLineEdit*>(WIDGET_TEXT_GAME_NAME);
    std::string matchNameStr = matchName->text().toUtf8().constData();

    if (matchName->text().isEmpty()) {
        QMessageBox msgBox;
        msgBox.setWindowTitle("Partida invÃ¡lida.");
        msgBox.setText("Por favor, ingrese un nombre a la partida.");
        msgBox.exec();
        return;
```



```
    }

    if (this->map_game_path.size() <= 0) {
        QMessageBox msgBox;
        msgBox.setWindowTitle("Partida invÃ¡lida.");
        msgBox.setText("Por favor, seleccione un mapa de la lista.");
        msgBox.exec();
        return;
    }

    /* std::cout << "Lanzo una partida en espera!" << std::endl; */
    this->pages->setCurrentIndex(PAGE_WAITING_MATCH_INDEX);
    Event new_event(a_createMatch, matchNameStr, this->map_players_qty);
    this->protocol->sendEvent(new_event);
}

void ClientLobby::backLobby(void) {
    this->pages->setCurrentIndex(PAGE_LOBBY_INDEX);
    QLabel* currentMapPath = findChild<QLabel*>(WIDGET_TEXT_CHOOSED_MAP);
    currentMapPath->setText("Not selected. Please, choose a map!");
    this->map_game_path.clear();
    refreshLobby();
}

void ClientLobby::cleanLobby(void) {
    QTableWidget * matchsList = findChild<QTableWidget*>(WIDGET_GAMES_TABLE);
    while (matchsList->rowCount() > 0) {
        matchsList->removeRow(0);
    }
}

void ClientLobby::feedLobby(void) {
    /* std::cout << "Alimentando lobby!" << std::endl; */
    QTableWidget * matchsList = findChild<QTableWidget*>(WIDGET_GAMES_TABLE);
    YAML::Node gameStatus;
    this->protocol->rcvGameStatus(gameStatus);

    std::stringstream ss;
    ss << gameStatus;
    std::cout << ss.str() << std::endl;



    YAML::Node::const_iterator it;
    for (it = gameStatus["waiting_games"].begin(); it != gameStatus["waiting_games"].
end(); it++) {

        matchsList->insertRow(matchsList->rowCount());

        QTableWidgetItem * table_game_name = new QTableWidgetItem((*it)["match_na
me"].as<std::string>().c_str());
        table_game_name->setFlags(table_game_name->flags() ^ Qt::ItemIsEditable)
;
        matchsList->setItem(matchsList->rowCount()-1, 0, table_game_name);

        QTableWidgetItem * table_game_creator = new QTableWidgetItem((*it)["creat
or"].as<std::string>().c_str());
        table_game_creator->setFlags(table_game_creator->flags() ^ Qt::ItemIsEdi
table);
        matchsList->setItem(matchsList->rowCount()-1, 1, table_game_creator);

        QTableWidgetItem * table_actual_players = new QTableWidgetItem((*it)["joi
```

```
ned_players"].as<std::string>().c_str());
        table_actual_players->setFlags(table_actual_players->flags() ^ Qt::ItemI
sEditable);
        matchsList->setItem(matchsList->rowCount()-1, 2, table_actual_players);

        QTableWidgetItem * table_max_players = new QTableWidgetItem((*it)["require
d_players"].as<std::string>().c_str());
        table_max_players->setFlags(table_max_players->flags() ^ Qt::ItemIsEdita
ble);
        matchsList->setItem(matchsList->rowCount()-1, 3, table_max_players);
    }
}

void ClientLobby::chooseMap(void) {
    QString map_path;
    map_path = QFileDialog::getOpenFileName(this, tr("Choose a map"), "/usr/etc/worms/
maps", tr("Tar gzipped (*.tar.gz)"));
    if (map_path.length() > 0) {
        this->map_game_path = map_path.toUtf8().constData();
        /* std::cout << "El mapa elegido es " << this->map_game_path << std::end
l; */
        QLabel* currentMapPath = findChild<QLabel*>(WIDGET_TEXT_CHOOSED_MAP);
        currentMapPath->setText(this->map_game_path.c_str());
        std::string cmd_mkdir = "mkdir temp_map_folder";
        std::string cmd_unzip = "tar -xvf \"" +  this->map_game_path + "\" -C ./temp_map
_folder/";
        std::system(cmd_mkdir.c_str());
        std::system(cmd_unzip.c_str());
        YAML::Node mapNode = YAML::LoadFile("temp_map_folder/map.yml");
        this->map_players_qty = mapNode["dynamic"]["worms_teams"].size();
        std::string cmd_rm_temp_dir = "rm -fr ./temp_map_folder";
        std::system(cmd_rm_temp_dir.c_str());
    } else {
        /* std::cout << "No se eligio un mapa." << std::endl; */
    }
}

void ClientLobby::feedWaitingPlayers(void) {
    /* std::cout << "Alimento la lista de jugadores en espera." << std::endl; */
    Event new_event(a_refreshWaitingList);
    this->protocol->sendEvent(new_event);
    YAML::Node waiting_players_list;
    this->protocol->rcvMsg(waiting_players_list);

    std::stringstream ss;
    ss << waiting_players_list;
    std::cout << ss.str() << std::endl;

    QListWidget * waitingPlayersList = findChild<QListWidget*>(WIDGET_LIST_WAITI
NG_PLAYERS);

    waitingPlayersList->clear();

    YAML::Node::const_iterator it;
    for (it = waiting_players_list["waiting_players"].begin(); it != waiting_players
_list["waiting_players"].end(); it++) {
        std::string waiting_player_name = (*it).as<std::string>();
        if (waiting_player_name == this->player_name) continue;

        QListWidgetItem * new_list_widget = new QListWidgetItem;
        new_list_widget->setText(tr(waiting_player_name.c_str()));
        waitingPlayersList->insertItem(1, new_list_widget);
```



```
    }
}

void ClientLobby::startWaitingMatch(void) {
    /* std::cout << "Comienzo juego" << std::endl; */
    Event new_event(a_startMatch);
    YAML::Node response;
    this->protocol->sendEvent(new_event);
    /* std::cout << "Esperando respuesta del server..." << std::endl; */
    this->protocol->rcvMsg(response);
    /* std::cout << "Respuesta del server recibida." << std::endl; */

    if (response["code"].as<int>() == 1) {
        /* std::cout << "El servidor me dio el OK para iniciar la partida."; */
        size_t team_id = response["team_id"].as<size_t>();
        /* std::cout << "Me asignÃ³ el team id " << team_id << std::endl; */
        std::fstream map_file(this->map_game_path, std::fstream::in | std::fstre
am::binary);
        this->protocol->sendFile(map_file);
        map_file.close();
        /* std::cout << "Aca instancio un juego cliente y lo lanzo pasandole el
protocolo." << std::endl; */
        ClientGame the_game(this->protocol, team_id, this->map_game_path);
        the_game.startGame();
        backLobby();
    } else {
        /* std::cout << "La partida no puede comenzar" << std::endl; */
        feedWaitingPlayers();
        QMessageBox msgBox;
        msgBox.setWindowTitle("No se puede iniciar partida.");
        std::string msg_response = response["msg"].as<std::string>();
        msgBox.setText(msg_response.c_str());
        msgBox.exec();
    }
}

// Invocada cuando el CREADOR de una partida en espera cancela dicha partida...
void ClientLobby::cancelWaitingMatch(void) {
    /* std::cout << "Cancelo juego en espera." << std::endl; */
    Event new_event(a_rmWaitingMatch);
    this->protocol->sendEvent(new_event);
    backLobby();
}

// Invocada cuando un participante no-creador de una partida en espera se va de
dicha partida en espera
void ClientLobby::exitWaitingMatch(void) {
    /* std::cout << "Me voy de una waiting match siendo un invitado." << std::en
dl; */
    if (this->waiting_match->isRunning()) {
        Event new_event(a_exitWaitingMatch);
        this->protocol->sendEvent(new_event);
    }
    this->waiting_match->stop();
    this->waiting_match->join();
    delete this->waiting_match;
    this->waiting_match = nullptr;
    backLobby();
}

void ClientLobby::closeEvent(QCloseEvent * event) {
```

```
    if (this->waiting_match) {
        exitWaitingMatch();
    }
    if (this->protocol) {
        exitLobby();
    }
    std::cout << "Cerrando cliente." << std::endl;
}
```



```
#ifndef CLIENT_LOBBY_H
#define CLIENT_LOBBY_H

#include <QMainWindow>
#include <QListWidgetItem>
#include <QtGui/QCloseEvent>
#include "QStackedWidget"
#include "protocol.h"
#include "waiting_match.h"

#define WIDGET_PAGES "pages_client"
#define WIDGET_BUTTON_CLEAN_LOGIN "button_clean"
#define WIDGET_BUTTON_CONNECT_LOGIN "button_connect"
#define WIDGET_BUTTON_CREATE_GAME "button_create"
#define WIDGET_BUTTON_EXIT_LOBBY "button_exit_lobby"
#define WIDGET_BUTTON_JOIN_GAME "button_join"
#define WIDGET_BUTTON_REFRESH_LOBBY "button_refresh"
#define WIDGET_BUTTON_START_WAITING_MATCH "button_start"
#define WIDGET_BUTTON_BACK_TO_LOBBY "button_back_lobby"
#define WIDGET_BUTTON_CHOOSE_MAP "button_changue_maps_folder"
#define WIDGET_BUTTON_REFRESH_WAITING_PLAYERS "button_refresh_waiting_players"
#define WIDGET_BUTTON_START_MATCH "button_start_waiting_match"
#define WIDGET_BUTTON_CANCEL_WAITING_MATCH "button_cancel_waiting_match"
#define WIDGET_BUTTON_EXIT_WAITING_MATCH "button_exit_waiting_match"

#define WIDGET_TEXT_PLAYER_NAME "text_player_name"
#define WIDGET_TEXT_IP "text_ip"
#define WIDGET_TEXT_PORT "text_port"
#define WIDGET_TEXT_PLAYER_NAME_LOBBY "playerName"
#define WIDGET_TEXT_GAME_CREATOR "text_game_creator"
#define WIDGET_TEXT_GAME_NAME "text_game_name"
#define WIDGET_TEXT_CHOOSED_MAP "text_current_map_path"

#define WIDGET_LIST_WAITING_PLAYERS "list_waiting_players"

#define WIDGET_GAMES_TABLE "table_matchs"


namespace Ui {
class ClientLobby;
}

class ClientLobby : public QMainWindow
{
    Q_OBJECT

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

private:
    Ui::ClientLobby *ui;
    Protocol * protocol;
    std::string player_name;
    QList<QListWidgetItem*> lobby_games;
    QStackedWidget * pages;
    WaitingMatch * waiting_match;

    std::string map_game_path;
    int map_players_qty;

    void connectEvents(void);
```

```
    void cleanTextBoxes(void);
    void connectToServer(void);
    void hideConnectionWindow(void);
    void showConnectionWindow(void);
    void goLobby(void);
    void createMatch(void);
    void exitLobby(void);
    void joinMatch(void);
    void refreshLobby(void);
    void feedLobby(void);
    void cleanLobby(void);
    void waitForPlayersOnCreatedMatch(void);
    void backLobby(void);
    void chooseMap(void);
    void feedWaitingPlayers(void);
    void startWaitingMatch(void);
    void cancelWaitingMatch(void);
    void exitWaitingMatch(void);
    void closeEvent(QCloseEvent *);
};

#endif
```



```
#include "client_settings.h"

ClientSettings::ClientSettings(void) {
    this->FULL_SCREEN = FULL_SCREEN_DEFAULT;
    this->RESOLUTION_HIGH = RESOLUTION_HIGH_DEFAULT;
    this->RESOLUTION_WIDTH = RESOLUTION_WIDTH_DEFAULT;
    this->SOUND_FX = SOUND_FX_DEFAULT;
}
```

```
#ifndef __CLIENT_SETTINGS_H__
#define __CLIENT_SETTINGS_H__

#define RESOLUTION_HIGH_DEFAULT 760
#define RESOLUTION_WIDTH_DEFAULT 1024
#define SOUND_FX_DEFAULT 1
#define FULL_SCREEN_DEFAULT 0

class ClientSettings {
    public:
        ClientSettings(void);

        int RESOLUTION_HIGH;
        int RESOLUTION_WIDTH;
        bool SOUND_FX;
        bool FULL_SCREEN;
};

#endif
```



```
#include "clock.h"

#define HURRY_TIME 11

View::Clock::Clock(int width, int height) :
  font(gPath.PATH_FONT_ARIAL_BOLD, height - PADDING * 2) {

  this->x = 0;
  this->y = 0;
  this->width = width;
  this->height = height;

  this->hide = false;
  this->hurrySound.setSound(gPath.PATH_SOUND_HURRY);
  this->timeTrickSound.setSound(gPath.PATH_SOUND_TIME_TRICK);
}

View::Clock::~Clock(void) {

}

int View::Clock::getWidth(void) const {
        return this->width;
}

int View::Clock::getHeight(void) const {
        return this->height;
}

int View::Clock::getX(void) const {
        return this->x;
}

int View::Clock::getY(void) const {
        return this->y;
}

void View::Clock::setX(int x) {
        this->x = x - this->width / 2;
}

void View::Clock::setY(int y) {
        this->y = y - this->height / 2;
}

void View::Clock::setTime(int newTime) {
  if (newTime == HURRY_TIME && this->time != newTime) {
    this->hurrySound.playSound(0);
  }

  if (newTime < HURRY_TIME && this->time != newTime) {
    this->timeTrickSound.playSound(0);
  }

  this->time = newTime;
}

void View::Clock::toggleHide(bool newState) {
  this->hide = newState;
}

void View::Clock::render(SDL_Renderer * r, int x, int y) {
```

```
  if (!this->hide) {
    SDL_Color color = {255, 255, 255, 0};

    if (this->time < HURRY_TIME) {
      color = {255, 0, 0, 0};
    }

    this->timeTexture.loadFromRenderedText(r, this->font, std::to_string(this->t
ime), color);

    // Black rect
    SDL_Rect blackRect = {
      this->x,
      this->y,
      this->width,
      this->height,
    };
    SDL_SetRenderDrawColor(r, 0x00, 0x00, 0x00, 0xFF);
    SDL_RenderFillRect(r, &blackRect);

    // Render time text
    this->timeTexture.render(
      r,
      this->x + this->width / 2 - this->timeTexture.getWidth() / 2,
      this->y + this->height / 2 - this->timeTexture.getHeight() / 2
    );
  }
}
```



```
#ifndef __CLOCK_H__
#define __CLOCK_H__

#include <SDL2/SDL.h>
#include "drawable.h"
#include "font.h"
#include "paths.h"
#include "sound_effect.h"
#include "texture.h"

#define TEXT_SIZE 40
#define PADDING 5

namespace View {
  class Clock: public Drawable {
    private:
      Font font;
      int time;
      Texture timeTexture;
      SoundEffect hurrySound;
      SoundEffect timeTrickSound;
      bool hide;



    public:
      Clock(int width, int height);
      ~Clock();
      void setTime(int);
      void toggleHide(bool);

      virtual void render(SDL_Renderer *, int, int);
      virtual int getWidth(void) const;
      virtual int getHeight(void) const;
      virtual int getX(void) const;
      virtual int getY(void) const;
      virtual void setX(int);
      virtual void setY(int);
  };
}

#endif
```

```cpp
#include "cluster.h"

#define CLUSTER_FPC 3

View::Cluster::Cluster(SDL_Renderer * r, int countdown, int ratioExplosion) :
        sprite(CLUSTER_FPC, INFINITE_GOING),
        explosion(r, ratioExplosion, "Grenade"),
        countdownText(COUNTDOWN_TEXT_SIZE) {
  this->texture.loadFromFile(gPath.PATH_CLUSTER, r);
  this->sprite.setSpriteSheet(&this->texture);

        this->exploded = false;
        this->finished = false;
        this->playedAboutToExplode = false;

        this->sound.setSound(gPath.PATH_SOUND_THROW_PROJECTIL);
  this->sound.playSound(0);
        this->countdown = countdown;
        this->ratioExplosion = ratioExplosion;
}

View::Cluster::~Cluster() {

}

void View::Cluster::render(SDL_Renderer * r, int camX, int camY) {
        if (!this->exploded) {
                // Render Cluster animation
                SDL_Rect clip = this->sprite.getNextClip();
                this->texture.render(r, this->x - camX, this->y - camY, &clip);

                // Render countdown text
                this->countdownText.setText(r, std::to_string(this->countdown));
                this->countdownText.setX(this->x + this->texture.getWidth() +  t
his->countdownText.getWidth() / 2);
                this->countdownText.setY(this->y - this->countdownText.getHeight
() / 2);
                this->countdownText.render(r, camX, camY);
        } else {
                if (!this->explosion.finishedExplosion()) {
                        this->explosion.setX(this->x + this->getWidth() / 2);
                        this->explosion.setY(this->y + this->getHeight() / 2);
                        this->explosion.render(r, camX, camY);
                } else {
                        this->finished = true;
                }
        }
}
```



```cpp
#ifndef __CLUSTER_H__
#define __CLUSTER_H__

#include <SDL2/SDL.h>
#include "explosion.h"
#include "paths.h"
#include "projectil.h"
#include "rectangle_text.h"


namespace View {
  class Cluster: public Projectil {
    private:
      SpriteAnimation sprite;
      Explosion explosion;
      RectangleText countdownText;

    public:
      Cluster(SDL_Renderer * r, int countdown, int ratioExplosion = 100);
      ~Cluster();

      virtual void render(SDL_Renderer * r, int, int);
  };
}

#endif
```

```
#include "dead.h"

#define DEAD_RATIO_EXPLOSION 50

View::Dead::Dead(View::Worm * worm, SDL_Renderer * r) :
  explosion(r, DEAD_RATIO_EXPLOSION, "Bazooka"){
  this->state = WS_DEAD;
  this->context = worm;

  this->textureDying.loadFromFile(gPath.PATH_WORM_DIE, r);
  this->spriteDying.setSpriteSheet(&this->textureDying);
  this->spriteDying.changeSpriteType(ONLY_GOING);

  const char * graves[] = {
    //gPath.PATH_GRAVE_1.c_str(),
    gPath.PATH_GRAVE_2.c_str(),
    gPath.PATH_GRAVE_3.c_str()
    //gPath.PATH_GRAVE_4.c_str(),
    //gPath.PATH_GRAVE_5.c_str(),
    //gPath.PATH_GRAVE_6.c_str()
  };

  const char * sounds[] = {
    gPath.PATH_SOUND_DIE.c_str(),
    gPath.PATH_SOUND_BYE.c_str()
  };

  this->textureGrave.loadFromFile(graves[rand() % 2], r);
  this->spriteGrave.setSpriteSheet(&this->textureGrave);
  this->spriteGrave.changeSpriteType(INFINITE_GOING_AND_BACK);

  this->sound.setSound(sounds[rand() % 2]);
  this->soundPlayed = false;

  this->dying = true;
}

View::Dead::~Dead() {

}

void View::Dead::render(SDL_Renderer * r, int camX, int camY, worm_inclination_t
 incl, bool mirrored, int angle) {
  if (!this->soundPlayed) {
    this->soundPlayed = true;
    this->sound.playSound(0);
  }

  if (this->dying) {
    if (!this->spriteDying.finished()) {
      SDL_Rect clip = this->spriteDying.getNextClip();
      View::Texture & current = this->textureDying;
      if (mirrored) {
        current.render(
          r,
          this->context->getX() - current.getWidth() / 2 - camX,
          this->context->getY() - current.getWidth() / 2 - camY,
          &clip,
          0,
          NULL,
          SDL_FLIP_HORIZONTAL
        );
```



```
      } else {
        current.render(
          r,
          this->context->getX() - current.getWidth() / 2 - camX,
          this->context->getY() - current.getWidth() / 2 - camY,
          &clip
        );
      }
    } else {
      this->dying = false;
    }
  } else if (!this->explosion.finishedExplosion()) {
    this->explosion.setX(this->context->getX());
    this->explosion.setY(this->context->getY());
    this->explosion.render(r, camX, camY);
  } else {
    this->context->setAffectedByExplosion(false);
    SDL_Rect clip = this->spriteGrave.getNextClip();
    this->textureGrave.render(
      r,
      this->context->getX() - this->textureGrave.getWidth() / 2 - camX,
      this->context->getY() - this->textureGrave.getWidth() / 2 - camY,
      &clip
    );
  }
}

void View::Dead::resetAnimation(void) {
  this->spriteDying.reset();
  this->spriteGrave.reset();
}
```

jun 25, 18 20:09 **dead.h** Page 1/1

```
#ifndef __DEAD_H__
#define __DEAD_H__

#include <map>
#include "worm_state.h"
#include "sprite_animation.h"
#include "sound_effect.h"
#include "texture.h"
#include "worm.h"
#include "types.h"
#include "explosion.h"

namespace View {
  class Worm;

  class Dead: public WormState {
    private:
      Texture textureDying;
      SpriteAnimation spriteDying;
      Explosion explosion;

      Texture textureGrave;
      SpriteAnimation spriteGrave;

      SoundEffect sound;
      bool soundPlayed;

      bool dying;

    public:
      Dead(View::Worm * context, SDL_Renderer * r);
      ~Dead();
      virtual void render(SDL_Renderer *, int, int, worm_inclination_t, bool, in
t angle);
      virtual void resetAnimation(void);
  };
}


#endif
```

jun 25, 18 20:09 **drawable.h** Page 1/1

```
#ifndef __DRAWABLE_H__
#define __DRAWABLE_H__

#include <SDL2/SDL.h>

namespace View {
  class Drawable {
    protected:
      int x;
      int y;
      int width;
      int height;
    public:
      virtual void render(SDL_Renderer *, int, int) = 0;
      virtual int getWidth(void) const = 0;
      virtual int getHeight(void) const = 0;
      virtual int getX(void) const = 0;
      virtual int getY(void) const = 0;
      virtual void setX(int) = 0;
      virtual void setY(int) = 0;
  };
}

#endif
```

jun 25, 18 20:09 **dynamite.cpp** Page 1/1

```cpp
#include "dynamite.h"
#define DYNAMITE_FPC 3

View::Dynamite::Dynamite(SDL_Renderer * r, int countdown, int ratioExplosion) :
        sprite(DYNAMITE_FPC, INFINITE_GOING_AND_BACK),
        explosion(r, ratioExplosion, "Dynamite"),
        countdownText(COUNTDOWN_TEXT_SIZE) {
  this->texture.loadFromFile(gPath.PATH_DYNAMITE, r);
  this->sprite.setSpriteSheet(&this->texture);

        this->exploded = false;
        this->finished = false;

        const char * laughs[] = {
                gPath.PATH_SOUND_LAUGH.c_str(),
                gPath.PATH_SOUND_FATALITY.c_str()
        };

        this->laugh.setSound(laughs[rand() % 2]);
        this->laugh.playSound(0);

        this->sound.setSound(gPath.PATH_SOUND_DYNAMITE);
        this->sound.playSound();
        this->countdown = countdown;
        this->ratioExplosion = ratioExplosion;
}

View::Dynamite::~Dynamite() {

}

void View::Dynamite::render(SDL_Renderer * r, int camX, int camY) {
        if (!this->exploded) {
                // Render dynamite animation
                SDL_Rect clip = this->sprite.getNextClip();
                this->texture.render(r, this->x - camX, this->y - camY, &clip);

                // Render countdown text
                this->countdownText.setText(r, std::to_string(this->countdown));
                this->countdownText.setX(this->x + this->texture.getWidth() +  t
his->countdownText.getWidth() / 2);
                this->countdownText.setY(this->y - this->countdownText.getHeight
() / 2);
                this->countdownText.render(r, camX, camY);
        } else {
                this->sound.stopSound();
                if (!this->explosion.finishedExplosion()) {
                        this->explosion.setX(this->x + this->getWidth() / 2);
                        this->explosion.setY(this->y + this->getHeight() / 2);
                        this->explosion.render(r, camX, camY);
                } else {
                        this->finished = true;
                }
        }
}
```

jun 25, 18 20:09 **dynamite.h** Page 1/1

```cpp
#ifndef __DYNAMITE_H__
#define __DYNAMITE_H__

#include <SDL2/SDL.h>
#include <SDL2/SDL_mixer.h>
#include <string>
#include "drawable.h"
#include "explosion.h"
#include "paths.h"
#include "projectil.h"
#include "rectangle_text.h"
#include "sprite_animation.h"
#include "sound_effect.h"
#include "texture.h"


namespace View {
  class Dynamite: public Projectil {
    private:
      SpriteAnimation sprite;
      Explosion explosion;
      RectangleText countdownText;

      SoundEffect laugh;

    public:
      Dynamite(SDL_Renderer *, int c, int ratio = 100);
      ~Dynamite();

      virtual void render(SDL_Renderer * r, int, int);
  };
}

#endif
```

jun 26, 18 12:23 **event_sender.cpp** Page 1/1

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include "socket.h"
#include "event_sender.h"
#include "blocking_queue.h"
#include "thread.h"
#include "protocol.h"
#include "types.h"
#include "event.h"

EventSender::EventSender(Protocol * p, Queue<Event> & e) :
protocol(p),
events(e) {
    keep_runing = true;
}

EventSender::~EventSender(void) {

}

bool EventSender::isRunning(void) const {
    return true;
}

size_t EventSender::getId(void) const {
    return 0;
}

void EventSender::run(void) {
    while (keep_runing) {
        Event event = this->events.pop();
        YAML::Node evento = event.getNode();
        if (!event.quit()) {
            this->protocol->sendEvent(event);
        } else {
            std::cout << "Enviando evento de quit." << std::endl;
            this->protocol->sendEvent(event);
            return;
        }
    }
}

void EventSender::stop(void) {
    this->keep_runing = false;
}
```

jun 25, 18 20:09 **event_sender.h** Page 1/1

```cpp
#ifndef __EVENT_SENDER_H__
#define __EVENT_SENDER_H__

#include "thread.h"
#include "protocol.h"
#include "blocking_queue.h"
#include "socket.h"
#include "types.h"
#include "event.h"

class EventSender : public Thread {
    private:
        Protocol * protocol;
        Queue<Event> & events;
        bool keep_runing;

        virtual bool isRunning(void) const;
        virtual size_t getId(void) const;
    public:
        EventSender(Protocol *, Queue<Event> &);
        ~EventSender(void);
        virtual void run(void);
        void stop(void);
};

#endif
```

```
#include "explosion.h"

View::Explosion::Explosion(SDL_Renderer * r, int ratio, std::string weapon) :
  sprite(EXPLOSION_FPC, ONLY_GOING) {
  this->texture.loadFromFile(gPath.PATH_EXPLOSION_EFFECT, r);
  this->width = ratio * 2;
  this->height = ratio * 2;
  this->x = 0;
  this->y = 0;

  this->sprite.setSpriteSheet(&this->texture);
  if (weapon == "Bazooka") {
    this->sound.setSound(gPath.PATH_SOUND_EXPLOSION_3);
  } else if (weapon == "Grenade") {
    this->sound.setSound(gPath.PATH_SOUND_EXPLOSION_2);
  } else {
    this->sound.setSound(gPath.PATH_SOUND_EXPLOSION_1);
  }
  this->soundPlayed = false;
}

View::Explosion::~Explosion() {

}

int View::Explosion::getX(void) const {
        return this->x;
}

int View::Explosion::getY(void) const {
        return this->y;
}

int View::Explosion::getWidth(void) const {
  return this->width;
}

int View::Explosion::getHeight(void) const {
  return this->height;
}

void View::Explosion::setX(int x) {
        this->x = x - this->width / 2;
}

void View::Explosion::setY(int y) {
        this->y = y - this->height / 2;
}

void View::Explosion::render(SDL_Renderer * r, int camX, int camY) {
  if (!this->sprite.finished()) {
    if (!this->soundPlayed) {
      this->sound.playSound(0);
      this->soundPlayed = true;
    }
                SDL_Rect clip = this->sprite.getNextClip();
                this->texture.render(r, this->x - camX, this->y - camY, this->wi
dth, this->height, &clip);
        }
}

bool View::Explosion::finishedExplosion(void) {
```



```
  return this->sprite.finished();
}
```

jun 25, 18 20:09 **explosion.h** Page 1/1

```
#ifndef __EXPLOSION_H__
#define __EXPLOSION_H__

#include <SDL2/SDL.h>
#include "drawable.h"
#include "paths.h"
#include "sprite_animation.h"
#include "sound_effect.h"
#include "texture.h"

#define EXPLOSION_FPC 3

namespace View {
  class Explosion: public Drawable {
    private:
      Texture texture;
      SpriteAnimation sprite;
      SoundEffect sound;
      bool soundPlayed;
    public:
      Explosion(SDL_Renderer * r, int ratio, std::string weapon = "Bazooka");
      ~Explosion();

      bool finishedExplosion(void);

      virtual void render(SDL_Renderer *, int, int);
      virtual int getWidth(void) const;
      virtual int getHeight(void) const;
      virtual int getX(void) const;
      virtual int getY(void) const;
      virtual void setX(int);
      virtual void setY(int);
  };
}

#endif
```

jun 25, 18 20:09 **falling.cpp** Page 1/1

```
#include "falling.h"

View::Falling::Falling(View::Worm * worm, SDL_Renderer * r) {
  this->state = WS_FALLING;
  this->context = worm;
  this->texture.loadFromFile(gPath.PATH_WORM_ROLL, r);
  this->sprite.setSpriteSheet(&this->texture);
  this->sprite.changeSpriteType(INFINITE_GOING);
}

View::Falling::~Falling() {

}

void View::Falling::render(SDL_Renderer * r, int camX, int camY, worm_inclinatio
n_t incl, bool mirrored, int angle) {
  SDL_Rect clip = this->sprite.getNextClip();
  View::Texture & current = this->texture;
  if (mirrored) {
    current.render(
      r,
      this->context->getX() - current.getWidth() / 2 - camX,
      this->context->getY() - current.getWidth() / 2 - camY,
      &clip,
      0,
      NULL,
      SDL_FLIP_HORIZONTAL
    );
  } else {
    current.render(
      r,
      this->context->getX() - current.getWidth() / 2 - camX,
      this->context->getY() - current.getWidth() / 2 - camY,
      &clip
    );
  }
}

void View::Falling::resetAnimation(void) {
  this->sprite.reset();
}
```

jun 25, 18 20:09 **falling.h** Page 1/1

```
#ifndef __FALLING_H__
#define __FALLING_H__

#include <map>
#include "sprite_animation.h"
#include "texture.h"
#include "worm_state.h"
#include "worm.h"
#include "types.h"

namespace View {
  class Worm;

  class Falling: public WormState {
    private:
      Texture texture;
      SpriteAnimation sprite;

    public:
      Falling(View::Worm * context, SDL_Renderer * r);
      ~Falling();
      virtual void render(SDL_Renderer *, int, int, worm_inclination_t, bool, in
t);
      virtual void resetAnimation(void);
  };
}


#endif
```

jun 25, 18 21:23 **flash_notice.cpp** Page 1/2

```
#include "flash_notice.h"

#define FLASH_NOTICE_PADDING 5
#define MAX_TIME_DISPLAYING 1500

FlashNotice::FlashNotice(void) :
  notice(30) {
}

FlashNotice::FlashNotice(int sw, int sh, int percentWindow) :
  notice(sh / (100 / percentWindow)) {
  this->screenWidth = sw;
  this->screenHeight = sh;
}

FlashNotice::~FlashNotice() {

}

void FlashNotice::showFlashNotice(SDL_Renderer * r, const std::string & notice)
{
  SDL_Color black = {0,0,0,0};
  SDL_Color white = {255,255,255,255};
  this->notice.setBackgroundColor(black);
  this->notice.setTextColor(white);
  this->notice.setText(r, notice);
  if (this->timer.isStarted()) {
    this->timer.stop();
  }
  this->timer.start();
  this->notice.setX(this->screenWidth / 2);
  this->notice.setY(FLASH_NOTICE_PADDING + this->notice.getHeight() / 2);
}

void FlashNotice::showFlashError(SDL_Renderer * r, const std::string & notice) {
  SDL_Color black = {0,0,0,0};
  SDL_Color red = {255,0,0,0};
  this->notice.setBackgroundColor(black);
  this->notice.setTextColor(red);
  this->notice.setText(r, notice);
  if (this->timer.isStarted()) {
    this->timer.stop();
  }
  this->timer.start();
  this->notice.setX(this->screenWidth / 2);
  this->notice.setY(FLASH_NOTICE_PADDING + this->notice.getHeight() / 2);
}

void FlashNotice::render(SDL_Renderer * r) {
  if (this->timer.getTicks() < MAX_TIME_DISPLAYING && this->timer.isStarted()) {
    this->notice.render(r, 0, 0);
  } else {
    if (this->timer.isStarted()) {
      this->timer.stop();
    }
  }
}

void FlashNotice::setScreenWidth(int w) {
  this->screenWidth = w;
}
```

```
void FlashNotice::setScreenHeight(int h) {
  this->screenHeight = h;
}
```



```
#ifndef __FLASH_NOTICE_H__
#define __FLASH_NOTICE_H__

#include <SDL2/SDL.h>
#include <string>
#include "rectangle_text.h"
#include "sdl_timer.h"

class FlashNotice {
  private:
    View::RectangleText notice;
    int screenWidth;
    int screenHeight;
    Timer timer;

  public:
    FlashNotice(void);
    FlashNotice(int sw, int sh, int percentWindow = 4);
    ~FlashNotice();
    void render(SDL_Renderer *);
    void showFlashNotice(SDL_Renderer *, const std::string &);
    void showFlashError(SDL_Renderer *, const std::string &);
    void setScreenWidth(int);
    void setScreenHeight(int);
};

#endif
```

```
#include "flying.h"
#define MAX_FLYING_TEXTURES 3

View::Flying::Flying(View::Worm * worm, SDL_Renderer * r) {
  this->state = WS_FLYING;
  this->context = worm;

  const char * sounds[] = {
    gPath.PATH_SOUND_OOFF_1.c_str(),
    gPath.PATH_SOUND_OOFF_2.c_str(),
    gPath.PATH_SOUND_OOFF_3.c_str(),
    gPath.PATH_SOUND_OW_1.c_str(),
    gPath.PATH_SOUND_OW_2.c_str(),
    gPath.PATH_SOUND_OW_3.c_str()
  };
  this->sound.setSound(sounds[rand() % 6]);
  this->soundPlayed = false;

  this->index = 0;
  this->textures[this->index].loadFromFile(gPath.PATH_WORM_FLYING_1, r);
  this->sprites[this->index].setSpriteSheet(&this->textures[this->index]);
  this->sprites[this->index].changeSpriteType(DEPENDENT_ON_GRADES);

  this->index = 1;
  this->textures[this->index].loadFromFile(gPath.PATH_WORM_FLYING_2, r);
  this->sprites[this->index].setSpriteSheet(&this->textures[this->index]);
  this->sprites[this->index].changeSpriteType(DEPENDENT_ON_GRADES);

  this->index = 2;
  this->textures[this->index].loadFromFile(gPath.PATH_WORM_FLYING_3, r);
  this->sprites[this->index].setSpriteSheet(&this->textures[this->index]);
  this->sprites[this->index].changeSpriteType(DEPENDENT_ON_GRADES);

  this->index = 0;
}

View::Flying::~Flying() {

}

void View::Flying::render(SDL_Renderer * r, int camX, int camY, worm_inclination
_t incl, bool mirrored, int angle) {
  if (!(this->index < MAX_FLYING_TEXTURES)) {
    this->index = 0;
  }

  if (!this->soundPlayed) {
    this->sound.playSound(0);
    this->soundPlayed = true;
  }

  View::SpriteAnimation & currentAnimation = this->sprites[this->index];
  View::Texture & current = this->textures[this->index];
  this->index++;

  int angleAdapted = angle;

  if (angle > 180) {
    angleAdapted = 360 - angle;
  }
  SDL_Rect clip = currentAnimation.getNextClip(angleAdapted, 180);
```



```
  if (angle <= 180) {
    current.render(
      r,
      this->context->getX() - current.getWidth() / 2 - camX,
      this->context->getY() - current.getWidth() / 2 - camY,
      &clip,
      0,
      NULL,
      SDL_FLIP_HORIZONTAL
    );
  } else {
    current.render(
      r,
      this->context->getX() - current.getWidth() / 2 - camX,
      this->context->getY() - current.getWidth() / 2 - camY,
      &clip
    );
  }
}

void View::Flying::resetAnimation(void) {
  std::map<size_t, SpriteAnimation>::iterator it = this->sprites.begin();
  for (; it != this->sprites.end() ; it++) {
    it->second.reset();
  }
}
```

jun 25, 18 20:09 **flying.h** Page 1/1

```
#ifndef __FLYING_H__
#define __FLYING_H__

#include <map>
#include "sprite_animation.h"
#include "texture.h"
#include "worm_state.h"
#include "worm.h"
#include "types.h"
#include "sound_effect.h"

namespace View {
  class Worm;

  class Flying: public WormState {
    private:
      std::map<size_t, Texture> textures;
      std::map<size_t, SpriteAnimation> sprites;
      size_t index;
      SoundEffect sound;
      bool soundPlayed;

    public:
      Flying(View::Worm * context, SDL_Renderer * r);
      ~Flying();
      virtual void render(SDL_Renderer *, int, int, worm_inclination_t, bool, in
t);
      virtual void resetAnimation(void);
  };
}


#endif
```

jun 25, 18 20:09 **font.cpp** Page 1/1

```
#include "font.h"

View::Font::Font(std::string fontPath, size_t fontSize) {
  this->font = TTF_OpenFont(fontPath.c_str(), fontSize);
  if (!this->font) {
    throw View::Exception("%s: %s. %s: %s", ERR_MSG_OPEN_FONT, fontPath.c_str(), "S
DL_ttf Error", TTF_GetError());
  }
}

View::Font::~Font() {
  if (this->font) {
    TTF_CloseFont(this->font);
    this->font = NULL;
  }
}

TTF_Font * View::Font::getFont(void) const {
  return this->font;
}
```

```
#ifndef __FONT_H__
#define __FONT_H__

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

namespace View {
  class Font {
    private:
      TTF_Font * font;
    public:
      Font(std::string, size_t);
      ~Font();
      TTF_Font * getFont(void) const;
  };
}

#endif
```



```
#include "girder.h"

View::Girder::~Girder(void) {
}

void View::Girder::rotateClockwise(void) {
  // Si no es -90 grados
  if (this->currentIndexDegrees != DIFF_POS-1) {
    this->currentIndexDegrees++;
  } else {
    // Cambia a +60 grados
    this->currentIndexDegrees = 1;
  }
  this->currentTexture = this->textures[this->getCurrentDegrees()];
}

void View::Girder::rotateCounterClockwise(void) {
  // Si no es +90 grados
  if (this->currentIndexDegrees) {
    this->currentIndexDegrees--;
  } else {
    // Cambia a -60 grados
    this->currentIndexDegrees = DIFF_POS-2;
  }
  this->currentTexture = this->textures[this->getCurrentDegrees()];
}

void View::Girder::render(SDL_Renderer * renderer, int camX, int camY) {
  this->currentTexture.render(
    renderer,
    this->x - (this->currentTexture.getWidth() / 2) - camX,
    this->y - (this->currentTexture.getHeight() / 2) - camY
  );
}

degrees_t View::Girder::getCurrentDegrees(void) {
  degrees_t vec[] = {
    NINETY_DEGREES,
    SIXTY_DEGREES,
    FORTYFIVE_DEGREES,
    THIRTY_DEGREES,
    ZERO_DEGREES,
    NEGATIVE_THIRTY_DEGREES,
    NEGATIVE_FORTYFIVE_DEGREES,
    NEGATIVE_SIXTY_DEGREES,
    NEGATIVE_NINETY_DEGREES
  };
  return vec[this->currentIndexDegrees];
}

int View::Girder::getWidth(void) const {
  return this->currentTexture.getWidth();
}

int View::Girder::getHeight(void) const {
  return this->currentTexture.getHeight();
}

int View::Girder::getX(void) const {
  return this->x;
}
```

```
int View::Girder::getY(void) const {
  return this->y;
}

void View::Girder::setX(int x) {
  this->x = x;
}

void View::Girder::setY(int y) {;
  this->y = y;
}
```



```
#ifndef __GIRDER_H__
#define __GIRDER_H__

#include <iostream>
#include <map>
#include <SDL2/SDL.h>
#include "texture.h"

#define DIFF_POS 9

typedef enum {
  NINETY_DEGREES = 90,
  SIXTY_DEGREES = 60,
  FORTYFIVE_DEGREES = 45,
  THIRTY_DEGREES = 30,
  ZERO_DEGREES = 0,
  NEGATIVE_THIRTY_DEGREES = -30,
  NEGATIVE_FORTYFIVE_DEGREES = -45,
  NEGATIVE_SIXTY_DEGREES = -60,
  NEGATIVE_NINETY_DEGREES = -90,
} degrees_t;

namespace View {
  class Girder: public Drawable {
    protected:
      std::map<int, Texture> textures;
      std::size_t currentIndexDegrees;
      Texture currentTexture;

    public:
      virtual ~Girder(void);
      void rotateClockwise(void);
      void rotateCounterClockwise(void);
      degrees_t getCurrentDegrees(void);
      virtual int getWidth() const;
      virtual int getHeight() const;
      virtual int getX(void) const;
      virtual int getY(void) const;
      virtual void setX(int);
      virtual void setY(int);
      virtual void render(SDL_Renderer *, int, int);
  };
}

#endif
```

```cpp
#include "girder_long.h"

View::GirderLong::GirderLong(SDL_Renderer * renderer, int degrees) {
  this->textures[NINETY_DEGREES].loadFromFile(gPath.PATH_GIRDER_LONG_90, rendere
r);
  this->textures[SIXTY_DEGREES].loadFromFile(gPath.PATH_GIRDER_LONG_60, renderer
);
  this->textures[FORTYFIVE_DEGREES].loadFromFile(gPath.PATH_GIRDER_LONG_45, rend
erer);
  this->textures[THIRTY_DEGREES].loadFromFile(gPath.PATH_GIRDER_LONG_30, rendere
r);
  this->textures[ZERO_DEGREES].loadFromFile(gPath.PATH_GIRDER_LONG_0, renderer);
  this->textures[NEGATIVE_THIRTY_DEGREES].loadFromFile(gPath.PATH_GIRDER_LONG_NE
GATIVE_30, renderer);
  this->textures[NEGATIVE_FORTYFIVE_DEGREES].loadFromFile(gPath.PATH_GIRDER_LONG
_NEGATIVE_45, renderer);
  this->textures[NEGATIVE_SIXTY_DEGREES].loadFromFile(gPath.PATH_GIRDER_LONG_NEG
ATIVE_60, renderer);
  this->textures[NEGATIVE_NINETY_DEGREES].loadFromFile(gPath.PATH_GIRDER_LONG_NE
GATIVE_90, renderer);

  std::map<int, size_t> index;
  index[NINETY_DEGREES] = 0;
  index[SIXTY_DEGREES] = 1;
  index[FORTYFIVE_DEGREES] = 2;
  index[THIRTY_DEGREES] = 3;
  index[ZERO_DEGREES] = 4;
  index[NEGATIVE_THIRTY_DEGREES] = 5;
  index[NEGATIVE_FORTYFIVE_DEGREES] = 6;
  index[NEGATIVE_SIXTY_DEGREES] = 7;
  index[NEGATIVE_NINETY_DEGREES] = 8;

  if (degrees != ZERO_DEGREES) {
    this->currentIndexDegrees = index[degrees];
    this->currentTexture = this->textures[degrees];
  } else {
    this->currentIndexDegrees = 4; // Hardcoded
    this->currentTexture = this->textures[this->getCurrentDegrees()];
  }

  this->x = 0;
  this->y = 0;
}

View::GirderLong::~GirderLong() {}
```



```cpp
#ifndef __GIRDER_LONG_H__
#define __GIRDER_LONG_H__

#include <SDL2/SDL.h>
#include "girder.h"
#include "paths.h"
#include "paths.h"

namespace View {
  class GirderLong: public Girder {
    private:
    public:
      GirderLong(SDL_Renderer *, int d = ZERO_DEGREES);
      ~GirderLong();
  };
}

#endif
```

```cpp
#include "girder_short.h"

View::GirderShort::GirderShort(SDL_Renderer * renderer, int degrees) {
  this->textures[NINETY_DEGREES].loadFromFile(gPath.PATH_GIRDER_SHORT_90, render
er);
  this->textures[SIXTY_DEGREES].loadFromFile(gPath.PATH_GIRDER_SHORT_60, rendere
r);
  this->textures[FORTYFIVE_DEGREES].loadFromFile(gPath.PATH_GIRDER_SHORT_45, ren
derer);
  this->textures[THIRTY_DEGREES].loadFromFile(gPath.PATH_GIRDER_SHORT_30, render
er);
  this->textures[ZERO_DEGREES].loadFromFile(gPath.PATH_GIRDER_SHORT_0, renderer)
;
  this->textures[NEGATIVE_THIRTY_DEGREES].loadFromFile(gPath.PATH_GIRDER_SHORT_N
EGATIVE_30, renderer);
  this->textures[NEGATIVE_FORTYFIVE_DEGREES].loadFromFile(gPath.PATH_GIRDER_SHOR
T_NEGATIVE_45, renderer);
  this->textures[NEGATIVE_SIXTY_DEGREES].loadFromFile(gPath.PATH_GIRDER_SHORT_NE
GATIVE_60, renderer);
  this->textures[NEGATIVE_NINETY_DEGREES].loadFromFile(gPath.PATH_GIRDER_SHORT_N
EGATIVE_90, renderer);

  std::map<int, size_t> index;
  index[NINETY_DEGREES] = 0;
  index[SIXTY_DEGREES] = 1;
  index[FORTYFIVE_DEGREES] = 2;
  index[THIRTY_DEGREES] = 3;
  index[ZERO_DEGREES] = 4;
  index[NEGATIVE_THIRTY_DEGREES] = 5;
  index[NEGATIVE_FORTYFIVE_DEGREES] = 6;
  index[NEGATIVE_SIXTY_DEGREES] = 7;
  index[NEGATIVE_NINETY_DEGREES] = 8;

  if (degrees != ZERO_DEGREES) {
    this->currentIndexDegrees = index[degrees];
    this->currentTexture = this->textures[degrees];
  } else {
    this->currentIndexDegrees = 4; // Harcoded
    this->currentTexture = this->textures[this->getCurrentDegrees()];
  }

  this->x = 0;
  this->y = 0;
}

View::GirderShort::~GirderShort() {}
```



```cpp
#ifndef __GIRDER_SHORT_H__
#define __GIRDER_SHORT_H__

#include "girder.h"
#include "paths.h"

namespace View {
  class GirderShort: public Girder {
    private:
    public:
      GirderShort(SDL_Renderer *, int d = ZERO_DEGREES);
      ~GirderShort();
  };
}

#endif
```

```cpp
#include "green_grenade.h"

#define GREEN_GRENADE_FPC 3

View::GreenGrenade::GreenGrenade(SDL_Renderer * r, int countdown, int ratioExplo
sion) :
        sprite(GREEN_GRENADE_FPC, INFINITE_GOING),
        explosion(r, ratioExplosion, "Grenade"),
        countdownText(COUNTDOWN_TEXT_SIZE) {
  this->texture.loadFromFile(gPath.PATH_GREEN_GRENADE, r);
  this->sprite.setSpriteSheet(&this->texture);

        this->exploded = false;
        this->finished = false;
        this->playedAboutToExplode = false;

        this->sound.setSound(gPath.PATH_SOUND_THROW_PROJECTIL);
  this->sound.playSound(0);
        this->countdown = countdown;
        this->ratioExplosion = ratioExplosion;
}

View::GreenGrenade::~GreenGrenade() {

}

void View::GreenGrenade::render(SDL_Renderer * r, int camX, int camY) {
        if (!this->exploded) {
                // Render GreenGrenade animation
                SDL_Rect clip = this->sprite.getNextClip();
                this->texture.render(r, this->x - camX, this->y - camY, &clip);

                // Render countdown text
                this->countdownText.setText(r, std::to_string(this->countdown));
                this->countdownText.setX(this->x + this->texture.getWidth() +  t
his->countdownText.getWidth() / 2);
                this->countdownText.setY(this->y - this->countdownText.getHeight
() / 2);
                this->countdownText.render(r, camX, camY);
        } else {
                if (!this->explosion.finishedExplosion()) {
                        this->explosion.setX(this->x + this->getWidth() / 2);
                        this->explosion.setY(this->y + this->getHeight() / 2);
                        this->explosion.render(r, camX, camY);
                } else {
                        this->finished = true;
                }
        }
}
```



```cpp
#ifndef __GREEN_GRENADE_H__
#define __GREEN_GRENADE_H__

#include <SDL2/SDL.h>
#include "explosion.h"
#include "paths.h"
#include "projectil.h"
#include "rectangle_text.h"


namespace View {
  class GreenGrenade: public Projectil {
    private:
      SpriteAnimation sprite;
      Explosion explosion;
      RectangleText countdownText;

    public:
      GreenGrenade(SDL_Renderer * r, int countdown, int ratioExplosion = 100);
      ~GreenGrenade();

      virtual void render(SDL_Renderer * r, int, int);
  };
}

#endif
```

jun 25, 18 20:09 **holy_grenade.cpp** Page 1/1

```cpp
#include "holy_grenade.h"

#define HOLY_GRENADE_FPC 3
#define COUNTDOWN_HOLY_SOUND 2

View::HolyGrenade::HolyGrenade(SDL_Renderer * r, int countdown, int ratioExplosi
on) :
        sprite(HOLY_GRENADE_FPC, INFINITE_GOING),
        explosion(r, ratioExplosion, "Holy Grenade"),
        countdownText(COUNTDOWN_TEXT_SIZE) {
  this->texture.loadFromFile(gPath.PATH_HOLY_GRENADE, r);
  this->sprite.setSpriteSheet(&this->texture);

        this->exploded = false;
        this->finished = false;

        this->sound.setSound(gPath.PATH_SOUND_THROW_PROJECTIL);
  this->sound.playSound(0);
        this->countdown = countdown;
        this->ratioExplosion = ratioExplosion;

  this->holySound.setSound(gPath.PATH_SOUND_HOLY);
  this->holySoundPlayed = false;
        this->playedAboutToExplode = false;
}

View::HolyGrenade::~HolyGrenade() {

}

void View::HolyGrenade::render(SDL_Renderer * r, int camX, int camY) {
        if (!this->exploded) {
                // Render HolyGrenade animation
                SDL_Rect clip = this->sprite.getNextClip();
                this->texture.render(r, this->x - camX, this->y - camY, &clip);

    if (!this->holySoundPlayed && this->countdown <= COUNTDOWN_HOLY_SOUND) {
      this->holySound.playSound(0);
      this->holySoundPlayed = true;
    }

                // Render countdown text
                this->countdownText.setText(r, std::to_string(this->countdown));
                this->countdownText.setX(this->x + this->texture.getWidth() +  t
his->countdownText.getWidth() / 2);
                this->countdownText.setY(this->y - this->countdownText.getHeight
() / 2);
                this->countdownText.render(r, camX, camY);
        } else {
                if (!this->explosion.finishedExplosion()) {
                        this->explosion.setX(this->x + this->getWidth() / 2);
                        this->explosion.setY(this->y + this->getHeight() / 2);
                        this->explosion.render(r, camX, camY);
                } else {
                        this->finished = true;
                }
        }
}
```

jun 25, 18 20:09 **holy_grenade.h** Page 1/1

```cpp
#ifndef __HOLY_GRENADE_H__
#define __HOLY_GRENADE_H__

#include <SDL2/SDL.h>
#include "explosion.h"
#include "paths.h"
#include "projectil.h"
#include "sound_effect.h"
#include "rectangle_text.h"


namespace View {
  class HolyGrenade: public Projectil {
    private:
      SpriteAnimation sprite;
      Explosion explosion;
      RectangleText countdownText;
      SoundEffect holySound;
      bool holySoundPlayed;

    public:
      HolyGrenade(SDL_Renderer * r, int countdown, int ratioExplosion = 100);
      ~HolyGrenade();

      virtual void render(SDL_Renderer * r, int, int);
  };
}

#endif
```

```cpp
#include "inventory.h"

View::Inventory::~Inventory(void) {

}

void View::Inventory::toggleOpen(void) {
  this->open = !this->open;
}

bool View::Inventory::isOpen(void) const {
  return this->open;
}
```



```cpp
#include "inventory_editor.h"

View::EditorInventory::EditorInventory(SDL_Renderer * r, size_t amountTeams, int
 healthConfig) :
  amountTeams(amountTeams), font(gPath.PATH_FONT_ARIAL_BOLD, TEXT_SUPPLIES_SIZE)
,
  worm(r, "Worm", 0, healthConfig),
  girderShort(r, 0),
  girderLong(r,0) {

  // Short girder
  ItemIcon * icon = new ItemIcon;
  icon->texture.loadFromFile(gPath.PATH_ICON_SHORT_GIRDER, r);
  icon->selected = true;
  icon->supplies = INFINITY_SUPPLIES;
  icon->itemName = WEAPON_NAME_SHORT_GIRDER;
  this->items.push_back(icon);

  icon = new ItemIcon;
  icon->texture.loadFromFile(gPath.PATH_ICON_LONG_GIRDER, r);
  icon->selected = false;
  icon->supplies = INFINITY_SUPPLIES;
  icon->itemName = WEAPON_NAME_LONG_GIRDER;
  this->items.push_back(icon);

  // Worms teams
  for (size_t i = 1 ; i <= amountTeams ; i++) {
    icon = new ItemIcon;
    icon->texture.loadFromFile(gPath.PATH_PLAIN_WORM, r);
    icon->selected = false;
    icon->supplies = AMOUNT_WORMS_PER_TEAM;
    icon->itemName = std::to_string(i); // Team ID
    this->items.push_back(icon);
  }

  this->open = false;
  this->girdersDegrees = ZERO_DEGREES;
  this->wormsHealth = healthConfig;

  this->iconWidth = this->items.back()->texture.getWidth();
  this->iconHeight = this->items.back()->texture.getHeight();

  this->girderClick.setSound(gPath.PATH_SOUND_GIRDER);
  this->wormClick.setSound(gPath.PATH_SOUND_TELEPORT);
}

View::EditorInventory::~EditorInventory() {
  for (size_t i = 0 ; i < this->items.size() ; i++) {
    delete this->items[i];
  }
}

void View::EditorInventory::render(SDL_Renderer * r) {
  if (this->open) {
    SDL_Color colors[] = {
      {0, 0, 0, 0},
      {255, 0, 0, 0},
      {0, 255, 0, 0},
      {0, 0, 255, 0}
    };

    std::vector<ItemIcon *>::iterator it = this->items.begin();
```

```
    // Render short girder
    (*it)->texture.render(r, this->xOffset, this->yOffset, this->iconWidth, this
->iconHeight);
    if ((*it)->selected) {
      this->renderItemSelected(r, this->xOffset, this->yOffset, *it);
    }
    it++;

    (*it)->texture.render(r, this->xOffset, this->yOffset + this->iconHeight, th
is->iconWidth, this->iconHeight);
    if ((*it)->selected) {
      this->renderItemSelected(r, this->xOffset, this->yOffset + this->iconHeigh
t, *it);
    }
    it++;

    for (int i = POS_FIRST_WORMS_TEAM ; it != this->items.end() ; it++, i++) {
      size_t teamId = std::stoi((*it)->itemName);

      // Black rect
      SDL_Rect blackRect = {
        this->xOffset,
        this->yOffset + i * this->iconHeight,
        this->iconWidth,
        this->iconHeight
      };
      SDL_SetRenderDrawColor(r, 0x00, 0x00, 0x00, 0xFF);
      SDL_RenderFillRect(r, &blackRect);

      // Color rect
      SDL_Rect colorRect = {
        this->xOffset + PADDING,
        this->yOffset + i * this->iconHeight + PADDING,
        this->iconWidth - 2 * PADDING,
        this->iconHeight - 2 * PADDING
      };
      SDL_SetRenderDrawColor(r, colors[teamId].r, colors[teamId].g, colors[teamI
d].b, 0xFF);
      SDL_RenderFillRect(r, &colorRect);

      // Worm icon
      (*it)->texture.render(r, this->xOffset, this->yOffset + i * this->iconHeig
ht);

      if ((*it)->selected) {
        this->renderItemSelected(r, this->xOffset, this->yOffset + i * this->ico
nHeight, *it);
      }
    }
  }
}

void View::EditorInventory::renderItemSelected(SDL_Renderer * renderer, int x, i
nt y, ItemIcon * item) {
  SDL_Rect outlineRect = {
    x,
    y,
    this->iconWidth,
    this->iconHeight
  };
```



```
  // Color blanco
  SDL_SetRenderDrawColor(renderer, 0xFF, 0xFF, 0xFF, 0xFF);
  // Dibujamos rectangulo blanco en item seleccionado
  SDL_RenderDrawRect(renderer, &outlineRect);

  // Render text supplies
  SDL_Color white = {255, 255, 255, 0};

  std::string supplies;
  if (item->supplies != INFINITY_SUPPLIES) {
    supplies = std::to_string(item->supplies);
  } else {
    supplies = "oo";
  }
  this->suppliesTexture.loadFromRenderedText(renderer, this->font, "Supplies " + s
upplies, white);

  SDL_Rect rectSupplies = {
    x + this->iconWidth + PADDING,
    y + this->iconHeight / 2 - (this->suppliesTexture.getHeight() + PADDING * 2)
 / 2,
    this->suppliesTexture.getWidth() + PADDING * 2,
    this->suppliesTexture.getHeight() + PADDING * 2,
  };

  // Color negro
  SDL_SetRenderDrawColor(renderer, 0x00, 0x00, 0x00, 0xFF);
  // Dibujamos rectangulo negro en item seleccionado
  SDL_RenderFillRect(renderer, &rectSupplies);

  this->suppliesTexture.render(renderer, x + this->iconWidth + PADDING * 2, y +
this->iconHeight / 2 - (this->suppliesTexture.getHeight() + PADDING * 2) / 2 + P
ADDING);
}

void View::EditorInventory::renderSelectedInMouse(SDL_Renderer * r) {
  int mouseX, mouseY;
  SDL_GetMouseState(&mouseX, &mouseY);

  for (size_t i = 0; i < this->items.size() ; i++) {
    if (this->items.at(i)->selected) {
      // Short girder
      if (i == POS_GIRDER_SHORT) {

        // View::GirderShort g(r, this->girdersDegrees);
        // g.setX(0);
        // g.setY(0);
        //g.render(r, -mouseX, -mouseY);
        this->girderShort.setX(0);
        this->girderShort.setY(0);
        this->girderShort.render(r, -mouseX, -mouseY);
      } else if (i == POS_GIRDER_LONG) {
        // View::GirderLong g(r, this->girdersDegrees);
        // g.setX(0);
        // g.setY(0);
        // g.render(r, -mouseX, -mouseY);
        this->girderLong.setX(0);
        this->girderLong.setY(0);
        this->girderLong.render(r, -mouseX, -mouseY);
      } else {
        if (this->items.at(i)->supplies) {
          //std::string name("Worm " + std::to_string(AMOUNT_WORMS_PER_TEAM - th
```

```
is->items.at(i)->supplies + 1));
          this->worm.setX(0);
          this->worm.setY(0);
          this->worm.setDataConfiguration(NO_DATA);
          //this->worm.setTeamId(std::stoi(this->items.at(i)->itemName));
          //View::Worm w(r, name, std::stoi(this->items.at(i)->itemName), this->
wormsHealth);
          this->worm.render(r, -mouseX, -mouseY);
          //w.setX(0);
          //w.setY(0);
          //w.render(r, -mouseX, -mouseY);
        }
      }
    }
  }
}

// Evite mirar y entender este metodo
// puede causar migraÃ±a
void View::EditorInventory::handleEvent(
  SDL_Renderer * r,
  SDL_Event & e,
  View::MapGame & map,
  int camX,
  int camY
  ) {

  if (e.type == SDL_KEYDOWN) {
    // Si es Q y el inventario esta abierto
    // elige el arma siguiente
    if (e.key.keysym.sym == SDLK_q) {
      if (this->isOpen()) {
        this->pickNextItem();
      }
    }
    // Si es R y el inventario esta abierto
    // rotamos las vigas (para usuarios sin ruedita)
    if (e.key.keysym.sym == SDLK_r) {
      if (this->isOpen()) {
        // View::GirderShort g(r, this->girdersDegrees);
        // g.rotateClockwise();
        // this->girdersDegrees = g.getCurrentDegrees();
        this->girderShort.rotateClockwise();
        this->girderLong.rotateClockwise();
        this->girdersDegrees = this->girderShort.getCurrentDegrees();
      }
    }
  }

  // Click derecho abre o cierra el inventario
  if (e.type == SDL_MOUSEBUTTONDOWN) {
    if (e.button.button == SDL_BUTTON_RIGHT) {
      this->toggleOpen();
    }
  }

  if (e.type == SDL_MOUSEBUTTONDOWN) {
    int mouseX, mouseY;
    SDL_GetMouseState(&mouseX, &mouseY);
    if (
      e.button.button == SDL_BUTTON_LEFT &&
      this->isMouseOnInventoryRanges(mouseX, mouseY) &&
```



```
      this->isOpen()
    ) {
      this->handleClick();
      return;
    }
  }

  // Rotamos el dibujo de las vigas
  if (e.type == SDL_MOUSEWHEEL) {
    if (e.wheel.y > 0) {
      //View::GirderShort g(r, this->girdersDegrees);
      this->girderShort.rotateClockwise();
      this->girderLong.rotateClockwise();
      this->girdersDegrees = this->girderShort.getCurrentDegrees();
      //this->girdersDegrees = g.getCurrentDegrees();
    }
    if (e.wheel.y < 0) {
      //View::GirderShort g(r, this->girdersDegrees);
      //g.rotateCounterClockwise();
      //this->girdersDegrees = g.getCurrentDegrees();
      this->girderShort.rotateCounterClockwise();
      this->girderLong.rotateCounterClockwise();
      this->girdersDegrees = this->girderShort.getCurrentDegrees();
    }
  }

  // Click izquierdo actualiza la coleccion de objetos estaticos
  if (e.type == SDL_MOUSEBUTTONDOWN) {
    if (e.button.button == SDL_BUTTON_LEFT) {
      int mouseX, mouseY;
      SDL_GetMouseState(&mouseX, &mouseY);

      size_t index = this->getIndexSelected();

      if (index == POS_GIRDER_SHORT) {
        map.addShortGirder(this->girdersDegrees, camX + mouseX, camY + mouseY);
        this->girderClick.playSound(0);
      } else if (index == POS_GIRDER_LONG) {
        map.addLongGirder(this->girdersDegrees, camX + mouseX, camY + mouseY);
        this->girderClick.playSound(0);
      } else {
        if (this->items.at(index)->supplies) {
          this->wormClick.playSound(0);
          int teamId = std::stoi(this->items.at(index)->itemName);
          std::string name("Worm " + std::to_string(map.amountWormsTeam(teamId)
+ 1));
          map.addWormInTeam(teamId, name, this->wormsHealth, camX + mouseX, camY
 + mouseY);
          this->items.at(index)->supplies--;
        }

      }
    }
  }
}

void View::EditorInventory::handleClick(void) {
  int mouseX, mouseY;
  SDL_GetMouseState(&mouseX, &mouseY);

  if (this->isMouseOnInventoryRanges(mouseX, mouseY)) {
    for (size_t i = 0 ; i < this->items.size() ; i++) {
```

```
      ItemIcon * current = this->items[i];
      int lowLimit = this->yOffset + i * this->iconHeight;
      int upLimit = lowLimit + this->iconHeight;
      if (current->selected && mouseY > lowLimit && mouseY < upLimit) {
        // Si clickeo el que ya estaba seleccionado no hacemos nada
        break;
      }

      // El seleccionado viejo hay que desseleccionarlo
      if (current->selected) {
        current->selected = false;
      }

      // Y el clickeado hay que seleccionarlo
      if (mouseY > lowLimit && mouseY < upLimit) {
        current->selected = true;
      }
    }
  }
}

bool View::EditorInventory::isMouseOnInventoryRanges(int x, int y) {
  return (
    (this->xOffset < x) &&
    (x < this->xOffset + this->iconWidth) &&
    (y > this->yOffset) &&
    (y < this->yOffset + (int)this->items.size() * this->iconHeight)
  );
}

int View::EditorInventory::getIndexSelected(void) {
  for (size_t i = 0 ; i < this->items.size() ; i++) {
    if (this->items.at(i)->selected) {
      return i;
    }
  }
  return -1;
}

void View::EditorInventory::updateWormsTeamSupplies(const std::map<size_t, std::
vector<View::Worm*>> & worms) {
  std::map<std::size_t, std::vector<View::Worm*>>::const_iterator team_it;
  for (team_it = worms.begin(); team_it != worms.end(); ++team_it) {
    std::vector<View::Worm*>::iterator worm_it;
    ItemIcon * wormTeam = this->items[team_it->first -1 + POS_FIRST_WORMS_TEAM];
    wormTeam->supplies = AMOUNT_WORMS_PER_TEAM - team_it->second.size();
  }

  for (size_t i = 1; i <= amountTeams; ++i) {
    if (worms.find(i) == worms.end()) {
      this->items[i -1 + POS_FIRST_WORMS_TEAM]->supplies = AMOUNT_WORMS_PER_TEAM
;
    }
  }
}

void View::EditorInventory::pickNextItem(void) {
  for (size_t i = 0; i < this->items.size() ; i++) {
    if (this->items.at(i)->selected == true) {
      if (i == this->items.size() - 1) {
        this->items.back()->selected = false;
        this->items.front()->selected = true;
```



```
      } else {
        this->items.at(i)->selected = false;
        this->items.at(i+1)->selected = true;
      }
      break;
    }
  }
}
```

```
#ifndef __INVENTORY_EDITOR_H__
#define __INVENTORY_EDITOR_H__

#include <iostream>
#include <string>
#include <vector>
#include <SDL2/SDL.h>
#include "font.h"
#include "inventory.h"
#include "map_game.h"
#include "paths.h"
#include "sound_effect.h"
#include "texture.h"
#include "yaml.h"

#define WEAPON_NAME_SHORT_GIRDER "Short girder"
#define WEAPON_NAME_LONG_GIRDER "Long girder"

#define AMOUNT_WORMS_PER_TEAM 99
#define PADDING 5

#define POS_GIRDER_SHORT 0
#define POS_GIRDER_LONG 1
#define POS_FIRST_WORMS_TEAM 2

#define TEXT_SUPPLIES_SIZE 15

namespace View {
  class MapGame;

  class EditorInventory: public Inventory {
    private:
      size_t amountTeams;
      degrees_t girdersDegrees;
      int wormsHealth;

      Font font;
      Texture suppliesTexture;

      SoundEffect girderClick;
      SoundEffect wormClick;

      View::GirderShort girderShort;
      View::GirderLong girderLong;
      View::Worm worm;

      // Devuelve el indice del item seleccionado
      int getIndexSelected(void);

      // Dibuja rect blanco en item seleccionado y el texto de supplies
      void renderItemSelected(SDL_Renderer *, int, int, ItemIcon *);

      // Handlea el click izquierdo del mouse
      virtual void handleClick(void);

      // Checkea si el mouse esta en las
      // dimensiones del dibujo del inventario
      virtual bool isMouseOnInventoryRanges(int, int);

      bool newGirderDeegres;

    public:
```



```
      // Constructor por default con todas las armas
      EditorInventory(SDL_Renderer *, size_t, int);

      // Destructor
      ~EditorInventory();

      virtual void render(SDL_Renderer *);

      // Dibuja el item elegido en la posicion del mouse
      void renderSelectedInMouse(SDL_Renderer *);

      // Se le agrega al inventario del editor
      // que actualice las colecciones de objetos estaticos
      void handleEvent(
        SDL_Renderer *,
        SDL_Event &,
        View::MapGame &,
        int,
        int
      );

      // Actualiza la cantidad de worms que se pueden poner
      // Este metodo lo utiliza el map game dependiendo de cuantos
      // gusanos hay en el mapa en el estado actual
      void updateWormsTeamSupplies(const std::map<size_t, std::vector<View::Worm
*>> & worms);

      virtual void pickNextItem(void);
  };
}

#endif
```

```
#ifndef __INVENTORY_H__
#define __INVENTORY_H__

#include <iostream>
#include <string>
#include <vector>
#include <SDL2/SDL.h>
#include "texture.h"
#include "paths.h"
#include "worm.h"
#include "girder_short.h"
#include "girder_long.h"

#define INFINITY_SUPPLIES -1

struct ItemIcon {
  View::Texture texture;
  std::string itemName;
  int supplies;
  bool selected;
};

namespace View {
  class Inventory {
    protected:
      std::vector<ItemIcon *> items;
      bool open;
      const int xOffset = 10;
      const int yOffset = 10;
      int iconWidth;
      int iconHeight;

      virtual void handleClick(void) = 0;
      virtual bool isMouseOnInventoryRanges(int, int) = 0;

    public:
      virtual ~Inventory();

      // Checkea si el inventario esta abierto
      bool isOpen(void) const;

      // Invierte el estado de 'open'
      void toggleOpen(void);

      // Marca como seleccionada el item siguiente
      virtual void pickNextItem(void) = 0;

      virtual void render(SDL_Renderer *) = 0;
  };
}

#endif
```



```
#include "inventory.h"
#include "inventory_weapons.h"
#include "types.h"
#include <iostream>

View::WeaponsInventory::WeaponsInventory(SDL_Renderer * r, const YAML::Node & in
itInv) :
  font(gPath.PATH_FONT_ARIAL_BOLD, TEXT_SUPPLIES_SIZE) {

  this->iconPaths[w_bazooka] = gPath.PATH_ICON_BAZOOKA;
  this->iconPaths[w_mortar] = gPath.PATH_ICON_MORTAR;
  this->iconPaths[w_cluster] = gPath.PATH_ICON_RED_GRENADE;
  this->iconPaths[w_green_grenade] = gPath.PATH_ICON_GREEN_GRENADE;
  this->iconPaths[w_banana] = gPath.PATH_ICON_BANANA;
  this->iconPaths[w_holy_grenade] = gPath.PATH_ICON_HOLY_GRENADE;
  this->iconPaths[w_air_strike] = gPath.PATH_ICON_AIR_STRIKE;
  this->iconPaths[w_dynamite] = gPath.PATH_ICON_DYNAMITE;
  this->iconPaths[w_bat] = gPath.PATH_ICON_BASEBALL;
  this->iconPaths[w_teleport] = gPath.PATH_ICON_TELEPORT;

  YAML::const_iterator invIt = initInv.begin();
  bool isFirstItem = true;
  for (; invIt != initInv.end() ; invIt++) {
    ItemIcon * icon = new ItemIcon;
    weapon_t idItem = (weapon_t)invIt->first.as<int>();
    icon->texture.loadFromFile(this->iconPaths[idItem], r);
    icon->supplies = invIt->second["supplies"].as<int>();
    std::string itName(invIt->second["item_name"].as<std::string>());
    icon->itemName = itName;

    if (isFirstItem) {
      icon->selected = true;
      isFirstItem = false;
    } else {
      icon->selected = false;
    }

    this->itemsMap[idItem] = icon;
  }

  this->open = false;

  // Tamanio grande hardcodeado
  this->iconWidth = 60;
  this->iconHeight = 60;
}

View::WeaponsInventory::~WeaponsInventory() {
  for (size_t i = 0 ; i < this->items.size() ; i++) {
    delete this->items[i];
  }

  std::map<weapon_t, ItemIcon *>::iterator it = this->itemsMap.begin();
  for (; it != this->itemsMap.end() ; it++) {
    delete it->second;
  }
}

void View::WeaponsInventory::setIconSide(int size) {
  this->iconWidth = size;
  this->iconHeight = size;
}
```

```
void View::WeaponsInventory::render(SDL_Renderer * renderer) {
  if (this->open) {
    std::map<weapon_t, ItemIcon *>::iterator itMap = this->itemsMap.begin();
    size_t i = 0;
    for (; itMap != this->itemsMap.end() ; itMap++) {
      itMap->second->texture.render(
        renderer,
        this->xOffset,
        this->yOffset + i * this->iconHeight,
        this->iconWidth,
        this->iconHeight
      );

      if (itMap->second->selected) {
        this->renderItemSelected(
          renderer,
          this->xOffset,
          this->yOffset + i * this->iconHeight,
          itMap->second
        );
      }

      i++;
    }
  }
}

void View::WeaponsInventory::renderItemSelected(SDL_Renderer * renderer, int x,
int y, ItemIcon * item) {
  SDL_Rect outlineRect = {
    x,
    y,
    this->iconWidth,
    this->iconHeight
  };
  // Color verde
  SDL_SetRenderDrawColor(renderer, 0x00, 0xFF, 0x00, 0xFF);
  // Dibujamos rectangulo verde en arma seleccionada
  SDL_RenderDrawRect(renderer, &outlineRect);

  // Render text supplies
  SDL_Color white = {255, 255, 255, 0};

  std::string supplies;
  if (item->supplies != INFINITY_SUPPLIES) {
    supplies = std::to_string(item->supplies);
  } else {
    supplies = "oo";
  }
  this->suppliesTexture.loadFromRenderedText(renderer, this->font, "Supplies " + s
upplies, white);

  SDL_Rect rectSupplies = {
    x + this->iconWidth + PADDING,
    y + this->iconHeight / 2 - (this->suppliesTexture.getHeight() + PADDING * 2)
 / 2,
    this->suppliesTexture.getWidth() + PADDING * 2,
    this->suppliesTexture.getHeight() + PADDING * 2,
  };

  // Color negro
```



```
  SDL_SetRenderDrawColor(renderer, 0x00, 0x00, 0x00, 0xFF);
  // Dibujamos rectangulo negro en item seleccionado
  SDL_RenderFillRect(renderer, &rectSupplies);

  this->suppliesTexture.render(
    renderer,
    x + this->iconWidth + PADDING * 2,
    y + this->iconHeight / 2 - (this->suppliesTexture.getHeight() + PADDING * 2)
 / 2 + PADDING
  );
}

void View::WeaponsInventory::handleEvent(SDL_Event & e) {
  if (e.type == SDL_KEYDOWN) {
    // Si es Q y el inventario esta abierto
    // elige el arma siguiente
    if (e.key.keysym.sym == SDLK_q) {
      if (this->isOpen()) {
        this->pickNextItem();
      } else {
        this->toggleOpen();
      }
    }

    if (e.key.keysym.sym == SDLK_e) {
      if (this->isOpen()) {
        this->toggleOpen();
      }
    }
  }

  // Click derecho abre o cierra el inventario
  if (e.type == SDL_MOUSEBUTTONDOWN) {
    if (e.button.button == SDL_BUTTON_RIGHT) {
      this->toggleOpen();
    }
  }

  if (e.type == SDL_MOUSEBUTTONDOWN) {
    int mouseX, mouseY;
    SDL_GetMouseState(&mouseX, &mouseY);

    if (
      e.button.button == SDL_BUTTON_LEFT &&
      this->isMouseOnInventoryRanges(mouseX, mouseY) &&
      this->isOpen()
    ) {
      this->handleClick();
      return;
    }
  }
}

void View::WeaponsInventory::handleClick(void) {
  int mouseX, mouseY;
  SDL_GetMouseState(&mouseX, &mouseY);

  if (this->isMouseOnInventoryRanges(mouseX, mouseY)) {
    std::map<weapon_t, ItemIcon *>::iterator it = this->itemsMap.begin();
    size_t i = 0;
    for (; it != this->itemsMap.end() ; it++) {
      ItemIcon * current = it->second;
```

```
      int lowLimit = this->yOffset + i * this->iconHeight;
      int upLimit = lowLimit + this->iconHeight;
      if (current->selected && mouseY > lowLimit && mouseY < upLimit) {
        // Si clickeo el que ya estaba seleccionado no hacemos nada
        break;
      }

      // El seleccionado viejo hay que desseleccionarlo
      if (current->selected) {
        current->selected = false;
      }

      // Y el clickeado hay que seleccionarlo
      if (mouseY > lowLimit && mouseY < upLimit) {
        current->selected = true;
      }

      i++;
    }
  }
}

bool View::WeaponsInventory::isMouseOnInventoryRanges(int x, int y) {
  return (
    (this->xOffset < x) &&
    (x < this->xOffset + this->iconWidth) &&
    (y > this->yOffset) &&
    (y < this->yOffset + (int)this->itemsMap.size() * this->iconHeight)
  );
}

weapon_t View::WeaponsInventory::getSelectedWeapon(void) {
  std::map<weapon_t, ItemIcon *>::iterator it = this->itemsMap.begin();
  weapon_t weapon;

  for (; it != this->itemsMap.end() ; it++) {
    if (it->second->selected) {
      weapon = it->first;
      break;
    }
  }

  return weapon;
}

void View::WeaponsInventory::update(const YAML::Node & node) {
  YAML::const_iterator it = node.begin();
  for (; it != node.end() ; it++) {
    weapon_t weaponId = (weapon_t)it->first.as<int>();
    int supplies = it->second["supplies"].as<int>();
    if (supplies) {
      if (this->itemsMap.find(weaponId) != this->itemsMap.end()) {
        this->itemsMap[weaponId]->supplies = supplies;
      }
    } else {
      if (this->itemsMap.find(weaponId) != this->itemsMap.end()) {
        std::map<weapon_t, ItemIcon *>::iterator itMap = this->itemsMap.begin();

        for (; itMap != this->itemsMap.end() ; itMap++) {
          if (itMap->first == weaponId && itMap->second->selected) {
            itMap->second->selected = false;
            itMap++;
```



```
            if (itMap != this->itemsMap.end()) {

            } else {
              itMap = this->itemsMap.begin();
            }
            itMap->second->selected = true;
          }
        }

        delete this->itemsMap[weaponId];
        this->itemsMap.erase(weaponId);
      }
    }
  }
}

void View::WeaponsInventory::pickNextItem(void) {
  std::map<weapon_t, ItemIcon *>::iterator it = this->itemsMap.begin();
  for (; it != this->itemsMap.end() ; it++) {
    if (it->second->selected) {
      it->second->selected = false;
      it++;

      if (it == this->itemsMap.end()) {
        it = this->itemsMap.begin();
      }

      it->second->selected = true;
      break;
    }
  }
}
```

```
#ifndef __INVENTORY_WEAPONS_H__
#define __INVENTORY_WEAPONS_H__

#include <SDL2/SDL.h>
#include "font.h"
#include "inventory.h"
#include "paths.h"
#include "texture.h"
#include "types.h"
#include "yaml.h"

#define WEAPON_NAME_BAZOOKA "Bazooka"
#define WEAPON_NAME_MORTAR "Mortar"
#define WEAPON_NAME_GREEN_GRENADE "Grenade"
#define WEAPON_NAME_RED_GRENADE "Cluster"
#define WEAPON_NAME_BANANA "Banana"
#define WEAPON_NAME_HOLY_GRENADE "Holy grenade"
#define WEAPON_NAME_DYNAMITE "Dynamite"
#define WEAPON_NAME_BASEBALL "Baseball bat"
#define WEAPON_NAME_AIR_STRIKE "Air strike"
#define WEAPON_NAME_TELEPORT "Teleport"

#define TEXT_SUPPLIES_SIZE 15
#define PADDING 5

// El click esta programado para funcionar
// con un inventario de columna unica
#define MAX_COLS 1

namespace View {
  class WeaponsInventory: public Inventory {
    private:
      Font font;
      Texture suppliesTexture;
      std::map<weapon_t, std::string> iconPaths;
      std::map<weapon_t, ItemIcon *> itemsMap;

      // Handlea el click izquierdo del mouse
      virtual void handleClick(void);

      // Checkea si el mouse esta en las
      // dimensiones del dibujo del inventario
      virtual bool isMouseOnInventoryRanges(int, int);

      // Dibuja rect blanco en item seleccionado y el texto de supplies
      void renderItemSelected(SDL_Renderer *, int, int, ItemIcon *);

    public:
      // Constructor por default con todas las armas
      WeaponsInventory(SDL_Renderer *, const YAML::Node &);

      // Destructor
      ~WeaponsInventory();

      virtual void render(SDL_Renderer *);

      // Handlea un evento
      void handleEvent(SDL_Event &);

      // Setea el tamanio de las vistas de los iconos
      void setIconSide(int);
```



```
      // Retorna el weapon_t seleccionado
      weapon_t getSelectedWeapon(void);

      // Updatea el inventario con el nodo que envia el server
      void update(const YAML::Node &);

      // Pickea la siguiente arma
      virtual void pickNextItem(void);
  };
}

#endif
```

jun 25, 18 20:09 **main.cpp** Page 1/1

```cpp
#include "client_lobby.h"
#include "socket_error.h"
#include "paths.h"
#include "client_settings.h"
#include <QApplication>

#define MAX_QUEUE_MODELS 256

// Variable global
Paths gPath;
ClientSettings gClientSettings;

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

    return a.exec();
} catch(const SocketError & e) {
        std::cout << e.what() << std::endl;
} catch(const std::exception & e) {
                std::cout << e.what() << std::endl;
}
```

jun 26, 18 10:43 **model_receiver.cpp** Page 1/1

```cpp
#include <iostream>
#include <sstream>
#include <unistd.h>
#include "model_receiver.h"
#include "protected_dynamics.h"

ModelReceiver::ModelReceiver(Protocol * prot, ProtectedDynamics & dyn) :
protocol(prot),
dynamics(dyn) {
    keep_runing = true;
}

ModelReceiver::~ModelReceiver(void) {

}

bool ModelReceiver::isRunning(void) const {
    return true;
}

size_t ModelReceiver::getId(void) const {
    return 0;
}

void ModelReceiver::run(void) {
    while (this->keep_runing) {
        YAML::Node newDynamics;
        this->protocol->rcvModel(newDynamics);
        if (newDynamics["game_status"]) {
            if (newDynamics["game_status"]["finished"].as<int>() == 1) {
                std::cout << "La partida termino." << std::endl;
                this->keep_runing = false;
            }
        }
        this->dynamics.addModel(newDynamics);
    }
}

void ModelReceiver::stop(void) {
    this->keep_runing = false;
}
```

jun 25, 18 20:09 **model_receiver.h** Page 1/1

```
#ifndef _MODEL_RECEIVER_H__
#define _MODEL_RECEIVER_H__

#include "thread.h"
#include "protocol.h"
#include "blocking_queue.h"
#include "protected_dynamics.h"

class ModelReceiver : public Thread {
    private:
        Protocol * protocol;
        ProtectedDynamics & dynamics;
        bool keep_runing;

        virtual bool isRunning(void) const;
        virtual size_t getId(void) const;
    public:
        ModelReceiver(Protocol *, ProtectedDynamics &);
        ~ModelReceiver(void);
        virtual void run(void);
        void stop(void);
};

#endif
```

jun 25, 18 20:09 **pick_weapon.cpp** Page 1/1

```
#include "pick_weapon.h"

View::PickWeapon::PickWeapon(View::Worm * worm, SDL_Renderer * r) {
  this->state = WS_PICK_WEAPON;
  this->context = worm;



  this->textures[NONE].loadFromFile(gPath.PATH_WORM_BREATH_1, r);
  this->textures[UP].loadFromFile(gPath.PATH_WORM_BREATH_1_UP, r);
  this->textures[DOWN].loadFromFile(gPath.PATH_WORM_BREATH_1_DOWN, r);
  this->sprites[NONE].setSpriteSheet(&this->textures[NONE]);
  this->sprites[UP].setSpriteSheet(&this->textures[UP]);
  this->sprites[DOWN].setSpriteSheet(&this->textures[DOWN]);
}

View::PickWeapon::~PickWeapon() {

}

void View::PickWeapon::render(SDL_Renderer * r, int camX, int camY, worm_inclina
tion_t incl, bool mirrored, int angle) {
  SDL_Rect clip = this->sprites[incl].getNextClip();
  View::Texture & current = this->textures[incl];
  if (mirrored) {
    current.render(
      r,
      this->context->getX() - current.getWidth() / 2 - camX,
      this->context->getY() - current.getWidth() / 2 - camY,
      &clip,
      0,
      NULL,
      SDL_FLIP_HORIZONTAL
    );
  } else {
    current.render(
      r,
      this->context->getX() - current.getWidth() / 2 - camX,
      this->context->getY() - current.getWidth() / 2 - camY,
      &clip
    );
  }

}
```

```
#ifndef __PICK_WEAPON_H__
#define __PICK_WEAPON_H__

#include <map>
#include "sprite_animation.h"
#include "texture.h"
#include "worm_state.h"
#include "worm.h"
#include "types.h"
#include "sound_effect.h"

namespace View {
  class Worm;

  class PickWeapon: public WormState {
    private:
      std::map<weapon_t, std::map<worm_inclination_t, Texture>> textures;
      std::map<weapon_t, std::map<worm_inclination_t, SpriteAnimation>> sprites;

    public:
      PickWeapon(View::Worm * context, SDL_Renderer * r);
      ~PickWeapon();
      virtual void render(SDL_Renderer *, int, int, worm_inclination_t, bool, in
t);
      virtual void resetAnimation(void);
  };
}


#endif
```



```
#include "projectil.h"

#define ABOUT_TO_EXPLODE 1
#define NONE_COUNTDOWN -1

View::Projectil::~Projectil(void) {
}

int View::Projectil::getWidth(void) const {
  return this->texture.getWidth();
}

int View::Projectil::getHeight(void) const {
  return this->texture.getWidth();
}

int View::Projectil::getX(void) const {
  return this->x;
}

int View::Projectil::getY(void) const {
  return this->y;
}

void View::Projectil::setX(int x) {
  this->x = x - (this->texture.getWidth() / 2);
}

void View::Projectil::setY(int y) {
  this->y = y - (this->texture.getWidth() / 2);
}

bool View::Projectil::hasExploded(void) const {
  return this->exploded;
}

bool View::Projectil::hasFinished(void) const {
  return this->finished;
}

void View::Projectil::setExplode(bool exploded) {
  this->exploded = exploded;
}

void View::Projectil::setCountdown(int newCount) {
  this->countdown = newCount;

  if (!this->playedAboutToExplode && this->countdown <= ABOUT_TO_EXPLODE && this
->countdown != NONE_COUNTDOWN) {
    const char * ab2exp[] = {
      gPath.PATH_SOUND_WHAT_THE.c_str(),
      gPath.PATH_SOUND_UH_OH.c_str(),
      gPath.PATH_SOUND_TAKE_COVER.c_str(),
      gPath.PATH_SOUND_RUN_AWAY.c_str()
    };
    this->aboutToExplode.setSound(ab2exp[rand() % 4]);
    this->aboutToExplode.playSound(0);
    this->playedAboutToExplode = true;
  }
}

void View::Projectil::setWeaponType(weapon_t type) {
```

```
  this->type = type;
}

void View::Projectil::setAngleDirection(int angle) {
  this->angleDirection = angle;
}
```



```
#include "projectiles.h"

View::Projectiles::Projectiles(void) {

}

View::Projectiles::~Projectiles(void) {
  std::map<size_t, Projectil *>::iterator it = this->projectiles.begin();
  for (; it != this->projectiles.end() ; it++) {
    delete it->second;
  }
}

void View::Projectiles::render(SDL_Renderer * r, Camera & cam) {
  std::map<size_t, Projectil *>::iterator it = this->projectiles.begin();
  for (; it != this->projectiles.end() ; it++) {
    Projectil * current = it->second;
    if (!current->hasFinished()) {
      current->render(r, cam.getX(), cam.getY());
    }
  }
}

void View::Projectiles::update(SDL_Renderer * r, const YAML::Node & projNode) {
  YAML::const_iterator it = projNode.begin();
  for (; it != projNode.end() ; it++) {
    const YAML::Node & proj = *it;
    int projId = proj["id"].as<int>();

    if (this->projectiles.find(projId) == this->projectiles.end()) {
      this->createProjectil(
        r,
        projId,
        proj
      );
    }
    Projectil * viewProjectile = this->projectiles[projId];
    viewProjectile->setX(proj["x"].as<int>());
    viewProjectile->setY(proj["y"].as<int>());
    viewProjectile->setCountdown(proj["countdown"].as<int>());
    viewProjectile->setExplode((bool)proj["exploded"].as<int>());
    if (proj["angle_direction"]) {
      viewProjectile->setAngleDirection(proj["angle_direction"].as<int>());
    }
  }

  this->cleanProjectilesFinished();
}

void View::Projectiles::createProjectil(SDL_Renderer * r, int projId, const YAML
::Node & proj) {
  int count = proj["countdown"].as<int>();
  weapon_t type = (weapon_t)proj["type"].as<int>();
  int ratio = proj["blast_radius"].as<int>();

  switch (type) {
    case w_dynamite:
      this->projectiles[projId] = new View::Dynamite(r, count, ratio);
      break;

    case w_green_grenade:
      this->projectiles[projId] = new View::GreenGrenade(r, count, ratio);
```

```
      break;

    case w_holy_grenade:
      this->projectiles[projId] = new View::HolyGrenade(r, count, ratio);
      break;

    case w_banana:
      this->projectiles[projId] = new View::Banana(r, count, ratio);
      break;

    case w_bazooka:
      this->projectiles[projId] = new View::Bazooka(r, ratio);
      break;

    case w_mortar:
      this->projectiles[projId] = new View::Bazooka(r, ratio, w_mortar);
      break;

    case w_air_strike:
      this->projectiles[projId] = new View::AirStrike(r, ratio);
      break;

    case w_cluster:
      this->projectiles[projId] = new View::Cluster(r, count, ratio);
      break;

    default:
      throw View::Exception("%s: %i", ERR_MSG_UNKNOWN_PROJECTIL_TYPE, w_dynamite)
;
  }
}

void View::Projectiles::cleanProjectilesFinished(void) {
  std::map<size_t, Projectil *>::iterator it = this->projectiles.begin();
  for (; it != this->projectiles.end() ;) {
    Projectil * current = it->second;
    if (current->hasFinished()) {
      delete it->second;
      it = this->projectiles.erase(it);
    } else {
      it++;
    }
  }
}

const View::Projectil * View::Projectiles::getProjectilProtagonic(void) {
  if (this->projectiles.size()) {
    std::map<size_t, View::Projectil*>::const_iterator it = this->projectiles.be
gin();
    return it->second;
  } else {
    return nullptr;
  }
}
```



```
#ifndef __PROJECTILES_H__
#define __PROJECTILES_H__

#include <SDL2/SDL.h>
#include <map>
#include <string>
#include "camera.h"
#include "green_grenade.h"
#include "dynamite.h"
#include "holy_grenade.h"
#include "banana.h"
#include "bazooka.h"
#include "projectil.h"
#include "air_strike.h"
#include "cluster.h"
#include "types.h"
#include "yaml.h"
#include "view_exceptions.h"
#include "inventory_weapons.h"

namespace View {
  class Projectiles {
    private:
      std::map<size_t, Projectil *> projectiles;

      // Crea un proyectil nuevo dependiendo el tipo
      void createProjectil(SDL_Renderer *, int, const YAML::Node &);

      // Libera la memoria de los proyectiles que ya terminaron de usarse
      void cleanProjectilesFinished(void);

    public:
      Projectiles();
      ~Projectiles();
      void render(SDL_Renderer *, Camera &);
      void update(SDL_Renderer *, const YAML::Node &);

      // Devuelve la vista del proyectil protagonico
      // en caso de no haber proyectiles devuelve NULL
      const View::Projectil * getProjectilProtagonic(void);
  };
}

#endif
```

jun 25, 18 20:09 **projectil.h** Page 1/1

```
#ifndef __PROJECTIL_H__
#define __PROJECTIL_H__

#include <SDL2/SDL.h>
#include "drawable.h"
#include "paths.h"
#include "sound_effect.h"
#include "sprite_animation.h"
#include "texture.h"
#include "types.h"

#define COUNTDOWN_TEXT_SIZE 18

namespace View {
  class Projectil: public Drawable {
    protected:
      Texture texture;
      SoundEffect sound;
      SoundEffect aboutToExplode;
      bool playedAboutToExplode;

      bool exploded;
      bool finished;
      int countdown;
      int ratioExplosion;
      int angleDirection;
      weapon_t type;

    public:
      virtual ~Projectil();
      virtual int getWidth(void) const;
      virtual int getHeight(void) const;
      virtual int getX(void) const;
      virtual int getY(void) const;
      virtual void setX(int);
      virtual void setY(int);
      virtual void render(SDL_Renderer *, int, int) = 0;

      bool hasExploded(void) const;
      bool hasFinished(void) const;
      void setCountdown(int);
      void setExplode(bool);
      void setWeaponType(weapon_t);
      void setAngleDirection(int);
  };
}

#endif
```

jun 26, 18 12:23 **protected_dynamics.cpp** Page 1/2

```
#include <sstream>
#include <iostream>
#include "protected_dynamics.h"

#define TIE_GAME_CODE 0

ProtectedDynamics::ProtectedDynamics(YAML::Node & dyn) {
    this->dynamics = dyn;
}

void ProtectedDynamics::addModel(YAML::Node & new_dyn) {
    this->models.push(new_dyn);
}

bool ProtectedDynamics::popModel(void) {
    bool thereIsModel = this->models.size() != 0;
    if (this->models.size()) {
        this->dynamics.reset();
        this->dynamics = this->models.front();
        this->models.pop();
    }
    return thereIsModel;
}

bool ProtectedDynamics::finishedMatch(void) {
    bool end_match = false;
    if (this->dynamics["game_status"]) {
        end_match = this->dynamics["game_status"]["finished"].as<int>();
    }
    return end_match;
}

YAML::Node ProtectedDynamics::getWorms(void) {
    const YAML::Node & teams = this->dynamics["worms_teams"];
    return teams;
}

YAML::Node ProtectedDynamics::getProjectiles(void) {
    const YAML::Node & projectiles = this->dynamics["projectiles"];
    return projectiles;
}

YAML::Node ProtectedDynamics::getGameStatus(void) {
    const YAML::Node & game_status = this->dynamics["game_status"];
    return game_status;
}

YAML::Node ProtectedDynamics::getTeamInventory(size_t teamId) {
    return this->dynamics["worms_teams"][std::to_string(teamId)]["inventory"];
}

int ProtectedDynamics::getTurnTimeLeft(void) {
    if (this->dynamics["game_status"]) {
        return this->dynamics["game_status"]["turn_timeleft"].as<int>();
    }
    return -1;
}

size_t ProtectedDynamics::getWormProtagonicId(void) {
    if (this->dynamics["game_status"]) {
        return this->dynamics["game_status"]["protagonic_worm"].as<size_t>();
    }
```

```
    return 1;
}

bool ProtectedDynamics::hasGameStatus(void) {
    if (this->dynamics["game_status"]) {
        return true;
    }
    return false;
}

 bool ProtectedDynamics::teamDefeated(size_t team_id) {
     if (this->dynamics["game_status"]) {
         int team_health = this->dynamics["game_status"]["teams_health"][team_id].as<
int>();
         if (team_health <= 0) {
             return true;
         } else {
             return false;
         }
     }
     return false;
 }

 size_t ProtectedDynamics::getWinnerTeam(void) {
    std::map<size_t, int> teams_health = this->dynamics["game_status"]["teams_health"
].as<std::map<size_t, int>>();
    std::map<size_t, int>::iterator it;
    for (it = teams_health.begin(); it != teams_health.end(); it++) {
        if (it->second > 0) {
            return it->first;
        }
    }
    return TIE_GAME_CODE;
 }
size_t ProtectedDynamics::getTeamTurn(void) {
    if (this->dynamics["game_status"]) {
        return this->dynamics["game_status"]["team_turn"].as<size_t>();
    }
    return 0;
}
```



```
#ifndef __PROTECTED_DYNAMIC_MAP_H__
#define __PROTECTED_DYNAMIC_MAP_H__

#include "yaml.h"
#include <queue>

class ProtectedDynamics {
    private:
        std::queue<YAML::Node> models;
        YAML::Node dynamics;
    public:
        ProtectedDynamics(YAML::Node &);
        void addModel(YAML::Node &);
        YAML::Node getWorms(void);
        YAML::Node getProjectiles(void);
        YAML::Node getGameStatus(void);
        YAML::Node getTeamInventory(size_t);
        int getTurnTimeLeft(void);
        size_t getWormProtagonicId(void);
        bool popModel(void);
        bool finishedMatch(void);
        bool hasGameStatus(void);
        bool teamDefeated(size_t);
        size_t getWinnerTeam(void);
        size_t getTeamTurn(void);
};

#endif
```

```
#include "rectangle_text.h"

View::RectangleText::RectangleText(int height, int padding, std::string fontPath
) :
  font(fontPath, height - padding * 2) {

  this->x = 0;
  this->y = 0;
  this->width = 0; // El ancho dependera del texto
  this->height = height; // El alto del rectangulo es configurable

  // Text color default blanco
  this->textColor = {255, 255, 255, 0};

  // Background color default azul
  this->backgroundColor = {0, 0, 0, 0};

  this->hide = false;
  this->padding = padding;
}

View::RectangleText::~RectangleText() {

}

int View::RectangleText::getWidth(void) const {
        return this->width;
}

int View::RectangleText::getHeight(void) const {
        return this->height;
}

int View::RectangleText::getX(void) const {
        return this->x;
}

int View::RectangleText::getY(void) const {
        return this->y;
}

void View::RectangleText::setX(int x) {
        this->x = x - this->width / 2;
}

void View::RectangleText::setY(int y) {
        this->y = y - this->height / 2;
}

void View::RectangleText::setText(SDL_Renderer * r, const std::string & text) {
  this->text.loadFromRenderedText(r, this->font, text, this->textColor);
  this->width = this->text.getWidth() + this->padding * 2;
}

void View::RectangleText::setTextColor(SDL_Color & c) {
  this->textColor = c;
}

void View::RectangleText::setBackgroundColor(SDL_Color & c) {
  this->backgroundColor = c;
}
```



```
void View::RectangleText::toggleHide(bool h) {
  this->hide = h;
}

void View::RectangleText::render(SDL_Renderer * r, int camX, int camY) {
  if (!this->hide) {
    // Background rect
    SDL_Rect bgRect = {
      this->x - camX,
      this->y - camY,
      this->width,
      this->height,
    };
    SDL_SetRenderDrawColor(
      r,
      this->backgroundColor.r,
      this->backgroundColor.g,
      this->backgroundColor.b,
      this->backgroundColor.a
    );

    SDL_RenderFillRect(r, &bgRect);

    // Render text
    this->text.render(
      r,
      this->x /*- this->width / 2*/ + this->padding - camX,
      this->y /*- this->height / 2 */+ this->padding - camY
    );
  }
}
```

```
#ifndef __RECTANGLE_TEXT_H__
#define __RECTANGLE_TEXT_H__

#include <SDL2/SDL.h>
#include <string>
#include "drawable.h"
#include "font.h"
#include "paths.h"
#include "texture.h"

#define DEFAULT_PADDING_RECTANGLE_TEXT 2

namespace View {
  class RectangleText: public Drawable {
    private:
      Texture text;
      Font font;
      SDL_Color textColor;
      SDL_Color backgroundColor;

      int padding;
      bool hide;
    public:
      RectangleText(
        int h,
        int padding = DEFAULT_PADDING_RECTANGLE_TEXT,
        std::string path = gPath.PATH_FONT_ARIAL_BOLD
      );

      ~RectangleText();

      void setText(SDL_Renderer *, const std::string &);
      void setTextColor(SDL_Color &);
      void setBackgroundColor(SDL_Color &);
      void toggleHide(bool);

      virtual int getWidth(void) const;
      virtual int getHeight(void) const;
      virtual int getX(void) const;
      virtual int getY(void) const;
      virtual void setX(int);
      virtual void setY(int);
      virtual void render(SDL_Renderer *, int, int);
  };
}

#endif
```



```
#include "shoot_power.h"

#define WHITE_TICKNESS_PERCENT 10



View::ShootPower::ShootPower(int w, int h, int maxTime) {
  this->width = w;
  this->height = h;
  this->maxTimeShoot = maxTime;
  this->x = 0;
  this->y = 0;
  this->whiteTickness = this->height / (100 / WHITE_TICKNESS_PERCENT);
}

View::ShootPower::~ShootPower() {

}

int View::ShootPower::getWidth(void) const {
  return this->width;
}

int View::ShootPower::getHeight(void) const {
  return this->height;
}

int View::ShootPower::getX(void) const {
  return this->x;
}

int View::ShootPower::getY(void) const {
  return this->y;
}

void View::ShootPower::setX(int x) {
  this->x = x - this->width / 2;
}

void View::ShootPower::setY(int y) {
  this->y = y - this->height / 2;
}

void View::ShootPower::render(SDL_Renderer * r, int camX, int camY) {
  return;
}

void View::ShootPower::render(SDL_Renderer * r, int timeShooting) {
  // White rect
  SDL_Rect whiteRect = {
    this->x,
    this->y,
    this->width,
    this->height,
  };

  SDL_SetRenderDrawColor(r, 0xFF, 0xFF, 0xFF, 0xFF);
  SDL_RenderFillRect(r, &whiteRect);

  int totalWidth = this->width - 2 * this->whiteTickness;
  // Black rect
  SDL_Rect blackRect = {
```

```
    this->x + this->whiteTickness,
    this->y + this->whiteTickness,
    totalWidth,
    this->height - 2 * this->whiteTickness,
  };
  SDL_SetRenderDrawColor(r, 0x00, 0x00, 0x00, 0xFF);
  SDL_RenderFillRect(r, &blackRect);

  float factor = ((float)timeShooting / (float)this->maxTimeShoot);
  int widthRedRect = (int)(factor * totalWidth);
  // Red rect
  SDL_Rect redRect = {
    this->x + this->whiteTickness,
    this->y + this->whiteTickness,
    widthRedRect,
    this->height - 2 * this->whiteTickness,
  };

  SDL_SetRenderDrawColor(r, 0xFF, 0x00, 0x00, 0xFF);
  SDL_RenderFillRect(r, &redRect);
}
```



```
#ifndef __SHOOT_POWER_H__
#define __SHOOT_POWER_H__

#include <iostream>
#include <SDL2/SDL.h>
#include "drawable.h"

namespace View {
  class ShootPower: public Drawable {
    private:
      int maxTimeShoot;
      int whiteTickness;

    public:
      ShootPower(int w, int h, int time);
      ~ShootPower();
      virtual int getWidth(void) const;
      virtual int getHeight(void) const;
      virtual int getX(void) const;
      virtual int getY(void) const;
      virtual void setX(int);
      virtual void setY(int);
      virtual void render(SDL_Renderer *, int, int);
      void render(SDL_Renderer *, int timeShooting);
  };
}

#endif
```

```
#include "sight.h"

#define SIGHT_FPC 3

View::Sight::Sight(SDL_Renderer * r, int ratio, int angle) :
  sprite(SIGHT_FPC, INFINITE_GOING_AND_BACK) {
  this->texture.loadFromFile(gPath.PATH_DEFAULT_SIGHT, r);
  this->ratio = ratio;
  this->angle = angle;

  this->sprite.setSpriteSheet(&this->texture);
}

View::Sight::~Sight() {

}

int View::Sight::getWidth(void) const {
        return this->width;
}

int View::Sight::getHeight(void) const {
        return this->height;
}

int View::Sight::getX(void) const {
        return this->x;
}

int View::Sight::getY(void) const {
        return this->y;
}

void View::Sight::setX(int x) {
        this->x = x - this->texture.getWidth() / 2;
}

void View::Sight::setY(int y) {
        this->y = y - this->texture.getWidth() / 2;
}

void View::Sight::setXYcenter(int xc, int yc) {
  this->xCenter = xc;
  this->yCenter = yc;
}

void View::Sight::setAngle(int a) {
  this->angle = a * DEGTORAD;
}

void View::Sight::setRatio(int r) {
  this->ratio = r;
}

void View::Sight::setMirrored(bool m) {
  this->mirrored = m;
}

void View::Sight::render(SDL_Renderer * r, int camX, int camY) {
  int fct;
  this->mirrored ? fct = 1 : fct = -1;
```



```
  SDL_Rect clip = this->sprite.getNextClip();
  int x = xCenter + (int)(this->ratio * cos(this->angle) * fct);
  int y = yCenter - (int)(this->ratio * sin(this->angle));
  this->texture.render(r, x - this->texture.getWidth() / 2 - camX, y - this->tex
ture.getWidth() / 2 - camY, &clip);
}
```

jun 25, 18 20:09 **sight.h** Page 1/1

```
#ifndef __SIGHT_H__
#define __SIGHT_H__

#include "drawable.h"
#include "texture.h"
#include "paths.h"
#include "sprite_animation.h"

#define DEGTORAD 0.0174533

namespace View {
  class Sight: public Drawable {
    private:
      Texture texture;
      SpriteAnimation sprite;
      int ratio;
      float angle;

      int xCenter;
      int yCenter;

      bool mirrored;

    public:
      Sight(SDL_Renderer * r, int ratio = 100, int angle = 0);
      ~Sight();

      void setXYcenter(int, int);
      void setAngle(int);
      void setRatio(int);
      void setMirrored(bool);

      virtual int getWidth(void) const;
      virtual int getHeight(void) const;
      virtual int getX(void) const;
      virtual int getY(void) const;
      virtual void setX(int);
      virtual void setY(int);
      virtual void render(SDL_Renderer *, int, int);
  };
}

#endif
```

jun 25, 18 20:09 **sprite_animation.cpp** Page 1/3

```
#include "sprite_animation.h"

View::SpriteAnimation::SpriteAnimation(size_t fpc, sprite_type_t type) :
  fpc(fpc) {
  this->type = type;
  this->reverse = false;
  this->counter = 0;
  this->finish = false;
}

View::SpriteAnimation::~SpriteAnimation() {}

// Funciona y no le deseo a nadie que tenga
// que intentar de entender como es que funciona
// y mucho menos tener que debuggear este metodo.
SDL_Rect View::SpriteAnimation::getNextClip(int grades, int maxGrades) {
  switch (this->type) {
    case INFINITE_GOING_AND_BACK:
      return this->getNextClipInfiniteRoundTrip();
      break;
    case ONLY_GOING:
      return this->getNextClipOnlyGoing();
      break;
    case INFINITE_GOING:
      return this->getNextClipInfiniteGoing();
      break;
    case DEPENDENT_ON_GRADES:
      return this->getNextClipDependentOnGrades(grades, maxGrades);
      break;
    default:
      return {0, 0, 0, 0};
  }
}

void View::SpriteAnimation::setSpriteSheet(Texture * newTexture) {
  this->currentSpriteSheet = newTexture;
  this->clipWidth = this->currentSpriteSheet->getWidth();
  this->clipHeight = clipWidth;
  this->numClips = this->currentSpriteSheet->getHeight() / clipWidth;
  this->reverse = false;
  this->finish = false;
  this->counter = 0;
}

SDL_Rect View::SpriteAnimation::getNextClipInfiniteRoundTrip(void) {
  if (this->reverse == false) {
    if (this->counter < this->numClips * this->fpc) {
      SDL_Rect currentClip = {
          0,
          0 + (this->counter / this->fpc ) * this->clipHeight,
          this->clipWidth,
          this->clipHeight
      };
      this->counter++;
      return currentClip;
    }

    this->reverse = true;
    this->counter = ((this->numClips - 1) * fpc) - 1;
  }

  if (this->reverse == true) {
```

```
    if (this->counter >= this->fpc) {
      SDL_Rect currentClip = {
          0,
          0 + (this->counter / this->fpc ) * this->clipHeight,
          this->clipWidth,
          this->clipHeight
      };
      this->counter--;
      return currentClip;
    }

    this->counter = 0;
    this->reverse = false;
  }

  SDL_Rect secondClip = {
    0,
    0 + (this->counter / this->fpc ) * this->clipHeight,
    this->clipWidth,
    this->clipHeight
  };

  this->counter++;
  return secondClip;
}

SDL_Rect View::SpriteAnimation::getNextClipOnlyGoing(void) {
  SDL_Rect currentClip = {
    0,
    0 + (this->counter / this->fpc ) * this->clipHeight,
    this->clipWidth,
    this->clipHeight
  };

  if (this->counter < this->numClips * this->fpc) {
    this->counter++;
  } else {
    this->finish = true;
  }

  return currentClip;
}

SDL_Rect View::SpriteAnimation::getNextClipInfiniteGoing(void) {
  SDL_Rect currentClip = {
    0,
    0 + (this->counter / this->fpc ) * this->clipHeight,
    this->clipWidth,
    this->clipHeight
  };

  if (this->counter < this->numClips * this->fpc) {
    this->counter++;
  } else {
    this->counter = 0;
    currentClip = {
      0,
      0 + (this->counter / this->fpc ) * this->clipHeight,
      this->clipWidth,
      this->clipHeight
    };
  }
```



```
  return currentClip;
}

SDL_Rect View::SpriteAnimation::getNextClipDependentOnGrades(int grades, int max
Grades) {
  int numClip = (grades * this->numClips) / maxGrades;

  if (numClip == this->numClips) {
    numClip--;
  }

  SDL_Rect currentClip = {
    0,
    0 + numClip * this->clipHeight,
    this->clipWidth,
    this->clipHeight
  };

  return currentClip;
}

bool View::SpriteAnimation::finished(void) {
  return this->finish;
}

void View::SpriteAnimation::changeSpriteType(sprite_type_t newType) {
  this->type = newType;
}

void View::SpriteAnimation::reset(void) {
  this->finish = false;
  this->reverse = false;
  this->counter = 0;
}
```

jun 25, 18 20:09 **sprite_animation.h** Page 1/1

```
#ifndef __SPRITE_ANIMATION_H__
#define __SPRITE_ANIMATION_H__

#include "texture.h"

#define DEFAULT_FPC 3

typedef enum {
  INFINITE_GOING_AND_BACK = 0,
  ONLY_GOING = 1,
  INFINITE_GOING = 2,
  DEPENDENT_ON_GRADES = 3
} sprite_type_t;

namespace View {
  class SpriteAnimation {
    private:
      Texture * currentSpriteSheet;

      int fpc; // Frames per clip
      int counter;
      int clipWidth;
      int clipHeight;
      int numClips;
      bool reverse;
      bool finish;

      sprite_type_t type;

      SDL_Rect getNextClipInfiniteRoundTrip(void);
      SDL_Rect getNextClipOnlyGoing(void);
      SDL_Rect getNextClipInfiniteGoing(void);
      SDL_Rect getNextClipDependentOnGrades(int grades, int maxGrades = 360);

    public:
      SpriteAnimation(size_t fpc = DEFAULT_FPC, sprite_type_t type = INFINITE_GO
ING_AND_BACK);
      ~SpriteAnimation();
      void setSpriteSheet(Texture *);

      SDL_Rect getNextClip(int grades = -1, int maxGrades = 360);

      // Verifica si el recorrido del spritesheet
      // finalizo (para ONLY_GOING sprites)
      bool finished(void);

      // Cambia el tipo de sprite sheet
      void changeSpriteType(sprite_type_t);

      // Resetea la animacion y empieza desde el principio
      void reset(void);
  };
}

#endif
```

jun 25, 18 20:09 **teams_health.cpp** Page 1/3

```
#include "teams_health.h"

#define PADDING_PERCENT 5
#define WHITE_TICKNESS_PERCENT 10
#define SIZE_TEXT_PERCENT 25
#define BARS_PERCENT 75

View::TeamsHealth::TeamsHealth(SDL_Renderer * r, int width, int height, int team
sAmount, int wormsHealth, int maxWormsTeams) :
  text(height / (100 / SIZE_TEXT_PERCENT), 2, gPath.PATH_FONT_ARIAL_BOLD) {
  this->width = width;
  this->height = height;
  this->teamsAmount = teamsAmount;
  this->wormsHealth = wormsHealth;
  this->padding = this->height / (100 / PADDING_PERCENT);
  this->heightHealthRect = (height / (100 / BARS_PERCENT) / teamsAmount) - (this
->padding * 2);
  this->whiteTickness = this->heightHealthRect / (100 / WHITE_TICKNESS_PERCENT);
  this->maxTeamHealth = maxWormsTeams * this->wormsHealth;
  this->hide = false;

  SDL_Color black = {0,0,0,0};
  this->text.setBackgroundColor(black);

  SDL_Color white = {255, 255, 255, 255};
  this->text.setTextColor(white);

  this->text.setText(r, "Teams health");

  for (int i = 0 ; i < this->teamsAmount ; i++) {
    this->teamsHealth[i+1] = this->maxTeamHealth;
  }
}

View::TeamsHealth::~TeamsHealth() {

}

int View::TeamsHealth::getWidth(void) const {
        return this->width;
}

int View::TeamsHealth::getHeight(void) const {
        return this->height;
}

int View::TeamsHealth::getX(void) const {
        return this->x;
}

int View::TeamsHealth::getY(void) const {
        return this->y;
}

void View::TeamsHealth::setX(int x) {
        this->x = x - this->width / 2;
  this->text.setX(x);
}

void View::TeamsHealth::setY(int y) {
        this->y = y - this->height / 2;
  this->text.setY(y - this->height / 2 + this->text.getHeight() / 2);
```

```
}

void View::TeamsHealth::render(SDL_Renderer * r, int camX, int camY) {
  if (this->hide) {
    return;
  }

  const SDL_Color colors[] = {
    {0, 0, 0, 0},
    {255, 0, 0, 0},
    {0, 255, 0, 0},
    {0, 0, 255, 0}
  };

  // Render text
  this->text.render(r, camX, camY);

  for (int i = 0 ; i < this->teamsAmount ; i++) {
    // White rects
    SDL_Rect whiteRect = {
      this->x + this->padding,
      this->y + i * (this->padding + this->heightHealthRect) + this->text.getHei
ght(),
      this->width - this->padding * 2,
      this->heightHealthRect
    };

    SDL_SetRenderDrawColor(r, 0xFF, 0xFF, 0xFF, 0xFF);
    SDL_RenderFillRect(r, &whiteRect);

    // Black rects
    SDL_Rect blackRect = {
      this->x + this->padding + this->whiteTickness,
      this->y + i * (this->padding + this->heightHealthRect) + this->whiteTickne
ss + this->text.getHeight(),
      this->width - this->padding * 2 - this->whiteTickness * 2,
      this->heightHealthRect - this->whiteTickness * 2
    };

    SDL_SetRenderDrawColor(r, 0x00, 0x00, 0x00, 0xFF);
    SDL_RenderFillRect(r, &blackRect);

    int currentHealth = this->teamsHealth[i + 1];
    int maxRectHealthWidth = this->width - this->padding * 2 - this->whiteTickne
ss * 2;
    int rectHealthWidth = (currentHealth * maxRectHealthWidth) / this->maxTeamHe
alth;

    // Team health rect
    SDL_Rect teamRect = {
      this->x + this->padding + this->whiteTickness,
      this->y + i * (this->padding + this->heightHealthRect) + this->whiteTickne
ss + this->text.getHeight(),
      rectHealthWidth,
      this->heightHealthRect - this->whiteTickness * 2
    };

    SDL_SetRenderDrawColor(r, colors[i+1].r, colors[i+1].g, colors[i+1].b, color
s[i+1].a);
    SDL_RenderFillRect(r, &teamRect);
  }
}
```



```
void View::TeamsHealth::update(const YAML::Node & teamsHealthNode) {
  YAML::const_iterator it = teamsHealthNode.begin();

  for (; it != teamsHealthNode.end() ; it++) {
    int teamId = it->first.as<int>();
    this->teamsHealth[teamId] = it->second.as<int>();
  }
}

void View::TeamsHealth::toggleHide(void) {
  this->hide = !this->hide;
}
```

```
#ifndef __TEAMS_HEALTH_H__
#define __TEAMS_HEALTH_H__

#include <map>
#include "drawable.h"
#include "rectangle_text.h"
#include "paths.h"
#include "texture.h"
#include "yaml.h"


namespace View {
  class TeamsHealth: public Drawable {
    private:
      RectangleText text;
      int padding;
      int whiteTickness;
      int teamsAmount;
      int wormsHealth;
      int heightHealthRect;
      int maxTeamHealth;

      std::map<int, int> teamsHealth;
      bool hide;

    public:
      TeamsHealth(SDL_Renderer *, int, int, int, int, int);
      ~TeamsHealth();
      virtual int getWidth(void) const;
      virtual int getHeight(void) const;
      virtual int getX(void) const;
      virtual int getY(void) const;
      virtual void setX(int);
      virtual void setY(int);
      virtual void render(SDL_Renderer *, int, int);

      void update(const YAML::Node &);
      void toggleHide(void);
  };
}

#endif
```



```
#include "texture.h"

View::Texture::Texture() {
        this->texture = NULL;
        this->width = 0;
        this->height = 0;
        this->x = 0;
        this->y = 0;
}

View::Texture::~Texture() {
        this->free();
}

void View::Texture::loadFromFile(std::string path, SDL_Renderer * renderer) {
        // Liberamos la textura actual
        this->free();

        // La textura final
        SDL_Texture* newTexture = NULL;

        // Cargamos la surface en el path indicado
        SDL_Surface* loadedSurface = IMG_Load(path.c_str());
        if (loadedSurface == NULL) {
    throw View::Exception("%s %s. %s: %s", ERR_MSG_LOAD_IMAGE, path.c_str(), "SDL_I
MG_Load() ", IMG_GetError());
        } else {
                //Color key image
                SDL_SetColorKey(loadedSurface, SDL_TRUE, SDL_MapRGB(loadedSurfac
e->format, 0x80, 0x80, 0xBE));

                //Create texture from surface pixels
    newTexture = SDL_CreateTextureFromSurface(renderer, loadedSurface);
                if (newTexture == NULL) {
      throw View::Exception("%s %s. %s: %s", ERR_MSG_CREATE_TEXTURE, path.c_str(),
 "SDL_CreateTextureFromSurface() ", SDL_GetError());
                } else {
                        // Cargamos las dimensiones de la imagen
                        this->width = loadedSurface->w;
                        this->height = loadedSurface->h;
                }
                // Liberamos la surface
                SDL_FreeSurface( loadedSurface );
        }

        //Return success
        this->texture = newTexture;
}

void View::Texture::loadFromRenderedText(SDL_Renderer * r, Font & font, std::str
ing textureText, SDL_Color textColor) {
        //Get rid of preexisting texture
        free();

        //Render text surface
        SDL_Surface * textSurface = TTF_RenderText_Solid(font.getFont(), texture
Text.c_str(), textColor);
        if (textSurface == NULL) {
                throw View::Exception("%s. %s: %s", ERR_MSG_RENDER_TEXT_SURFACE, "
SDL_ttf Error", TTF_GetError());
        }       else {
                //Create texture from surface pixels
```

```
    this->texture = SDL_CreateTextureFromSurface(r, textSurface);
                if (this->texture == NULL) {
                        throw View::Exception("%s. %s: %s", ERR_MSG_CREATE_TEXTURE
_TEXT, "SDL Error", SDL_GetError());
                }
                else {
                        //Get image dimensions
                        this->width = textSurface->w;
                        this->height = textSurface->h;
                }

                //Get rid of old surface
                SDL_FreeSurface( textSurface );
        }
}

void View::Texture::free() {
        //Free texture if it exists
        if (this->texture != NULL) {
                SDL_DestroyTexture( this->texture );
                this->texture = NULL;
                this->width = 0;
                this->height = 0;
                this->x = 0;
                this->y = 0;
        }
}

void View::Texture::render(SDL_Renderer * renderer, int x, int y) {
        // Seteamos el espacio de dibujado y donde dibujarlo
        SDL_Rect renderQuad = { x, y, this->width, this->height };
        SDL_RenderCopy(renderer, this->texture, NULL, &renderQuad);
}

void View::Texture::render(SDL_Renderer * renderer, int x, int y, int width, int
 height, SDL_Rect * clip) {
        SDL_Rect renderQuad = { x, y, width, height };
        SDL_RenderCopy(renderer, this->texture, clip, &renderQuad);
}

void View::Texture::render(SDL_Renderer * renderer) {
        SDL_RenderCopy(renderer, this->texture, NULL, NULL);
}

void View::Texture::render(
        SDL_Renderer * renderer,
        int x,
        int y,
        SDL_Rect* clip,
        double angle,
        SDL_Point* center,
        SDL_RendererFlip flip) {

        // Seteamos espacio de renderizado
        SDL_Rect renderQuad = { x, y, this->width, this->height };

        // Seteamos las dimensiones del clip
        if (clip) {
                renderQuad.w = clip->w;
                renderQuad.h = clip->h;
        }
```



```
        //Renderizamos al screen
        SDL_RenderCopyEx(renderer, this->texture, clip, &renderQuad, angle, cent
er, flip);
}

int View::Texture::getWidth(void) const {
        return this->width;
}

int View::Texture::getHeight(void) const {
        return this->height;
}

int View::Texture::getX(void) const {
        return this->x;
}

int View::Texture::getY(void) const {
        return this->y;
}

void View::Texture::setX(int x) {
        this->x = x;
}

void View::Texture::setY(int y) {
        this->y = y;
}

void View::Texture::setColor(Uint8 red, Uint8 green, Uint8 blue) {
        //Modulate texture rgb
        SDL_SetTextureColorMod(this->texture, red, green, blue);
}

void View::Texture::setBlendMode(SDL_BlendMode blending) {
        //Set blending function
        SDL_SetTextureBlendMode(this->texture, blending);
}

void View::Texture::setAlpha(Uint8 alpha) {
        //Modulate texture alpha
        SDL_SetTextureAlphaMod(this->texture, alpha);
}
```

```
#ifndef __TEXTURE_H__
#define __TEXTURE_H__

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_ttf.h>
#include <string>
#include "font.h"
#include "view_exceptions.h"
#include "drawable.h"

namespace View {
  class Texture: public Drawable {
    private:
      // La textura actual
      SDL_Texture* texture;

      // Desaloca memoria
      void free();

    public:
      //Initializes variables
      Texture();

      //Deallocates memory
      ~Texture();

      // Carga la imagen desde un archivo
      void loadFromFile(std::string path, SDL_Renderer *);

      // Renderiza la textura en toda la pantalla
      void render(SDL_Renderer *);
      // Render textura a un punto dado
      virtual void render(SDL_Renderer *, int, int);
      // Para poder renderizar texturas espejadas o rotadas
                  void render(SDL_Renderer *, int x, int y, SDL_Rect* clip, doub
le angle = 0.0, SDL_Point* center = NULL, SDL_RendererFlip flip = SDL_FLIP_NONE)
;
      // Para forzar un ancho y un alto de la imagen a renderizar
      void render(SDL_Renderer * renderer, int x, int y, int width, int height,
SDL_Rect * rct = NULL);

      // Dimensiones de la imagen
      virtual int getWidth(void) const;
      virtual int getHeight(void) const;

      // Posiciones de la imagen
      virtual int getX(void) const;
      virtual int getY(void) const;

      // Seteo de posiciones de la imagen
      virtual void setX(int);
      virtual void setY(int);

      //Creates image from font string
      void loadFromRenderedText(SDL_Renderer *, Font &, std::string textureText,
 SDL_Color textColor);

      //Set color modulation
      void setColor(Uint8 red, Uint8 green, Uint8 blue);

      //Set blending
```



```
      void setBlendMode(SDL_BlendMode blending);

      //Set alpha modulation
      void setAlpha(Uint8 alpha);
  };
}

#endif
```

jun 25, 18 20:09 **view_exceptions.cpp** Page 1/1

```cpp
#include "view_exceptions.h"
#include <errno.h>
#include <cstdio>
#include <cstdarg>

View::Exception::Exception(const char* fmt, ...) noexcept {
  this->_errno = errno;

  va_list args;
  va_start(args, fmt);

  int s = vsnprintf(this->msg_error, ERR_BUFF_LEN, fmt, args);
  va_end(args);

  if (this->_errno) {
    strncpy(this->msg_error+s, strerror(_errno), ERR_BUFF_LEN-s);
  }

  this->msg_error[ERR_BUFF_LEN-1] = 0;
}

View::Exception::~Exception() {}

const char * View::Exception::what() const noexcept {
  return this->msg_error;
}
```

jun 25, 18 20:09 **view_exceptions.h** Page 1/1

```cpp
#ifndef __COMMON_EXCEPTIONS_H__
#define __COMMON_EXCEPTIONS_H__

#include <iostream>
#include <typeinfo>
#include <errno.h>
#include <cstring>
#include <cstdio>
#include <cstdarg>
#include <string>
#include "view_exceptions_messages.h"

#define ERR_BUFF_LEN 512

namespace View {
  class Exception : public std::exception {
    private:
      char msg_error[ERR_BUFF_LEN];
      int _errno;

    public:
      explicit Exception(const char* fmt, ...) noexcept;
      virtual const char *what() const noexcept;
      virtual ~Exception() noexcept;
  };
}


#endif
```

jun 25, 18 20:09 **view_exceptions_messages.h** Page 1/1

```
#ifndef __VIEW_EXCEPTION_MESSAGES_H__
#define __VIEW_EXCEPTION_MESSAGES_H__

#define ERR_MSG_LOAD_IMAGE \
"No se pudo cargar la imagen"

#define ERR_MSG_INIT_WINDOW \
"No se pudo inicializar la ventana"

#define ERR_MSG_SDL_CREATE_WINDOW \
"No se pudo crear la ventana"

#define ERR_MSG_SDL_CREATE_RENDERER \
"No se pudo crear el renderer"

#define ERR_MSG_SDL_INIT_VIDEO \
"No se pudo inicializar el video de SDL"

#define ERR_MSG_SDL_IMAGE_INIT \
"No se pudo inicializar la imagen de SDL"

#define ERR_MSG_SDL_MIXER_INIT \
"No se pudo inicializar el sonido de SDL"

#define ERR_MSG_LOADING_SOUND \
"No se pudo cargar el sonido"

#define ERR_MSG_SDL_TTF_INIT \
"No se pudo inicializar TTF de SDL"

#define ERR_MSG_CREATE_TEXTURE \
"No se pudo crear la textura"

#define ERR_MSG_OPEN_FONT \
"No se pudo abrir la fuente especificada"

#define ERR_MSG_RENDER_TEXT_SURFACE \
"No se pudo renderizar superficie de texto"

#define ERR_MSG_CREATE_TEXTURE_TEXT \
"No se pudo crear la textura con el texto"

#define ERR_MSG_UNKNOWN_PROJECTIL_TYPE \
"Tipo de proyectil desconocido"

#endif
```

jun 26, 18 12:23 **waiting_match.cpp** Page 1/2

```
#include "waiting_match.h"
#include "thread.h"
#include "protocol.h"
#include "yaml.h"
#include <qt5/QtWidgets/QMessageBox>
#include "QStackedWidget"
#include <fstream>
#include "client_game.h"

#define PAGE_LOBBY_INDEX 1

WaitingMatch::WaitingMatch(Protocol * p, QStackedWidget * pag) :
protocol(p),
pages(pag) {
    this->keep_running = true;
}

bool WaitingMatch::isRunning(void) const {
    return this->keep_running;
}

void WaitingMatch::run(void) {
    while (keep_running) {
        YAML::Node msg;
        this->protocol->rcvMsg(msg);

        if (msg["code"].as<int>() == 1) {
            if (msg["msg"].as<std::string>() == "exited") {
                this->keep_running = false;
                return;
            }
            if (msg["msg"].as<std::string>() == "started") {
                size_t team_id = msg["team_id"].as<size_t>();
                // Este evento se manda para destrabar el event receiver del ser
vidor y que pueda pasarle el socket
                // al handler de la partida...
                Event new_event(a_goToMatch);
                this->protocol->sendEvent(new_event);
                ClientGame the_game(this->protocol, team_id);
                the_game.startGame();
                this->pages->setCurrentIndex(PAGE_LOBBY_INDEX);
                this->keep_running = false;
                return;
            }
        } else if (msg["code"].as<int>() == 0) {
            if (msg["msg"].as<std::string>() == "aborted") {
                QMessageBox msgBox;
                msgBox.setWindowTitle("Partida cancelada.");
                msgBox.setText("El creador del juego cancelÃ³ la partida.");
                msgBox.exec();
                this->keep_running = false;
                return;
            }
        }
    }
}

void WaitingMatch::stop(void) {
    this->keep_running = false;
}

size_t WaitingMatch::getId(void) const {
```

```
    return 0;
}
```



```
#ifndef __WAITING_MATCH_H__
#define __WAITING_MATCH_H__

#include "thread.h"
#include "protocol.h"
#include "QStackedWidget"

class WaitingMatch : public Thread {
    private:
        Protocol * protocol;
        QStackedWidget * pages;
        bool keep_running;

        size_t getId(void) const;

    public:
        WaitingMatch(Protocol *, QStackedWidget*);
        bool isRunning(void) const;
        virtual void run(void);
        void stop(void);
};

#endif
```

jun 25, 18 20:09 **walking.cpp** Page 1/1

```
#include "walking.h"

View::Walking::Walking(View::Worm * worm, SDL_Renderer * r) {
  this->state = WS_WALKING;
  this->context = worm;
  this->textures[NONE].loadFromFile(gPath.PATH_WORM_WALK, r);
  this->textures[UP].loadFromFile(gPath.PATH_WORM_WALK_UP, r);
  this->textures[DOWN].loadFromFile(gPath.PATH_WORM_WALK_DOWN, r);
  this->sprites[NONE].setSpriteSheet(&this->textures[NONE]);
  this->sprites[UP].setSpriteSheet(&this->textures[UP]);
  this->sprites[DOWN].setSpriteSheet(&this->textures[DOWN]);
  this->walkingSound.setSound(gPath.PATH_SOUND_WORM_WALKING);
  this->walkingExpandSound.setSound(gPath.PATH_SOUND_WORM_WALKING_EXPAND);
  this->playedExpand = true;
}

View::Walking::~Walking() {
  this->walkingSound.stopSound();
}

void View::Walking::render(SDL_Renderer * r, int camX, int camY, worm_inclinatio
n_t incl, bool mirrored, int angle) {
  if (!this->walkingExpandSound.isPlaying() && !this->walkingSound.isPlaying())
{
    if (this->playedExpand) {
      this->walkingSound.playSound(0);
      this->playedExpand = false;
    } else {
      this->walkingExpandSound.playSound(0);
      this->playedExpand = true;
    }
  }

  SDL_Rect clip = this->sprites[incl].getNextClip();
  View::Texture & current = this->textures[incl];
  if (mirrored) {
    current.render(
      r,
      this->context->getX() - current.getWidth() / 2 - camX,
      this->context->getY() - current.getWidth() / 2 - camY,
      &clip,
      0,
      NULL,
      SDL_FLIP_HORIZONTAL
    );
  } else {
    current.render(
      r,
      this->context->getX() - current.getWidth() / 2 - camX,
      this->context->getY() - current.getWidth() / 2 - camY,
      &clip
    );
  }
}

void View::Walking::resetAnimation(void) {
  std::map<worm_inclination_t, SpriteAnimation>::iterator it = this->sprites.beg
in();
  for (; it != this->sprites.end() ; it++) {
    it->second.reset();
  }
}
```

jun 25, 18 20:09 **walking.h** Page 1/1

```
#ifndef __WALKING_H__
#define __WALKING_H__

#include <map>
#include "sprite_animation.h"
#include "texture.h"
#include "worm_state.h"
#include "worm.h"
#include "types.h"
#include "sound_effect.h"

namespace View {
  class Worm;

  class Walking: public WormState {
    private:
      std::map<worm_inclination_t, View::Texture> textures;
      std::map<worm_inclination_t, View::SpriteAnimation> sprites;
      SoundEffect walkingSound;
      SoundEffect walkingExpandSound;
      bool playedExpand;

    public:
      Walking(View::Worm * context, SDL_Renderer * r);
      ~Walking();
      virtual void render(SDL_Renderer *, int, int, worm_inclination_t, bool, in
t);
      virtual void resetAnimation(void);
  };
}


#endif
```

jun 25, 18 20:09 **water.cpp** Page 1/1

```
#include "water.h"

View::Water::Water() {}

View::Water::~Water() {}

void View::Water::init(SDL_Renderer * r, int x, int y, int levelWidth, int level
Height, const char * patterPath) {
  this->x = x;
  this->y = y;
  this->width = levelWidth;
  this->height = levelHeight - y;

  if (patterPath) {
    this->texture.loadFromFile(patterPath, r);
  } else {
    this->texture.loadFromFile(gPath.PATH_WATER_DEFAULT, r);
  }
}

void View::Water::render(SDL_Renderer * r, int camX, int camY) {
  int tileWidth = this->texture.getWidth();
  int tileHeight = this->texture.getHeight();
  for (int x = 0 ; x <= this->width / tileWidth ; x++) {
    for (int y = 0 ; y <= this->height / tileHeight ; y++) {
      this->texture.render(
        r,
        this->x + x * tileWidth - camX,
        this->y + y * tileHeight - camY);
    }
  }
}

int View::Water::getWidth(void) const {
  return this->width;
}
int View::Water::getHeight(void) const {
  return this->height;
}

int View::Water::getX(void) const {
  return this->x;
}

int View::Water::getY(void) const {
  return this->y;
}

void View::Water::setX(int x) {
  this->x = x;
}

void View::Water::setY(int y) {
  this->y = y;
}
```

jun 25, 18 20:09 **water.h** Page 1/1

```
#ifndef __WATER_H__
#define __WATER_H__

#include <SDL2/SDL.h>
#include "drawable.h"
#include "paths.h"
#include "texture.h"

#define MAX_FRAME 10
#define MAX_NUM_CLIP 9

namespace View {
  class Water: public Drawable {
    private:
      int x;
      int y;
      int width;
      int height;
      Texture texture;
    public:
      Water();
      ~Water();
      void init(SDL_Renderer * r, int x, int y, int levelWidth, int levelHeight,
 const char * waterPath = NULL);
      virtual void render(SDL_Renderer *, int, int);
      virtual int getWidth(void) const;
      virtual int getHeight(void) const;
      virtual int getX(void) const;
      virtual int getY(void) const;
      virtual void setX(int);
      virtual void setY(int);
  };
}


#endif
```

```
#include "wind.h"

#define MAX_MOD_WIND_POWER 6
#define WHITE_TICKNESS_PERCENT 10

View::Wind::Wind(SDL_Renderer * r, int width, int height) :
  text(height / 2, 2, gPath.PATH_FONT_ARIAL_BOLD) {
  this->windPower = 0;
  this->width = width;
  this->height = height;
  this->x = 0;
  this->y = 0;

  this->whiteTickness = this->height / (100 / WHITE_TICKNESS_PERCENT);

  SDL_Color black = {0,0,0,0};
  this->text.setBackgroundColor(black);

  SDL_Color white = {255, 255, 255, 255};
  this->text.setTextColor(white);

  this->text.setText(r, "Wind force");
  this->windLeft.loadFromFile(gPath.PATH_WIND_LEFT, r);
  this->windRight.loadFromFile(gPath.PATH_WIND_RIGHT, r);
}

View::Wind::~Wind() {

}

int View::Wind::getWidth(void) const {
        return this->width;
}

int View::Wind::getHeight(void) const {
        return this->height;
}

int View::Wind::getX(void) const {
        return this->x;
}

int View::Wind::getY(void) const {
        return this->y;
}

void View::Wind::setX(int x) {
        this->x = x - this->width / 2;
  this->text.setX(this->x - this->text.getWidth() / 2 - this->whiteTickness);
}

void View::Wind::setY(int y) {
        this->y = y - this->height / 2;
  this->text.setY(this->y + this->height / 2);
}

void View::Wind::render(SDL_Renderer * r, int camX, int camY) {
  // White rect
  SDL_Rect whiteRect = {
    this->x,
    this->y,
    this->width,
```



```
    this->height,
  };

  SDL_SetRenderDrawColor(r, 0xFF, 0xFF, 0xFF, 0xFF);
  SDL_RenderFillRect(r, &whiteRect);

  // Black rects
  SDL_Rect blackRect = {
    this->x + this->whiteTickness,
    this->y + this->whiteTickness,
    this->width - this->whiteTickness - this->width / 2 - this->whiteTickness /
2,
    this->height - this->whiteTickness * 2,
  };

  SDL_SetRenderDrawColor(r, 0x00, 0x00, 0x00, 0xFF);
  SDL_RenderFillRect(r, &blackRect);

  blackRect = {
    this->x + this->width / 2 + this->whiteTickness / 2,
    this->y + this->whiteTickness,
    this->width - this->whiteTickness - this->width / 2 - this->whiteTickness /
2,
    this->height - this->whiteTickness * 2,
  };

  SDL_SetRenderDrawColor(r, 0x00, 0x00, 0x00, 0xFF);
  SDL_RenderFillRect(r, &blackRect);

  if (this->windPower > 0) {
    this->windRight.render(r, this->windRectX, this->windRectY, this->windRectWi
dth, this->height - this->whiteTickness * 2);
  } else {
    this->windLeft.render(r, this->windRectX, this->windRectY, this->windRectWid
th, this->height - this->whiteTickness * 2);
  }

  this->text.render(r, 0, 0);
}

void View::Wind::setWindPower(int newPower) {
  this->windPower = newPower;

  int maxWidth = this->width / 2 - this->whiteTickness - this->whiteTickness / 2
;
  this->windRectWidth = (abs(newPower) * maxWidth) / MAX_MOD_WIND_POWER;

  if (newPower > 0) {
    this->windRectX = this->x + this->width / 2 + this->whiteTickness / 2;
  }

  if (newPower <= 0) {
    this->windRectX = this->x + this->width / 2 - this->whiteTickness / 2 - this
->windRectWidth;
  }

  this->windRectY = this->y + this->whiteTickness;
}
```

```
#ifndef __WIND_H__
#define __WIND_H__

#include <stdlib.h>
#include "drawable.h"
#include "rectangle_text.h"
#include "paths.h"
#include "texture.h"

namespace View {
  class Wind: public Drawable {
    private:
      int windPower;
      int whiteTickness;
      int windRectWidth;
      int windRectX;
      int windRectY;

      Texture windLeft;
      Texture windRight;

      RectangleText text;

    public:
      Wind(SDL_Renderer * r, int, int);
      ~Wind();

      virtual int getWidth(void) const;
      virtual int getHeight(void) const;
      virtual int getX(void) const;
      virtual int getY(void) const;
      virtual void setX(int);
      virtual void setY(int);
      virtual void render(SDL_Renderer *, int, int);

      void setWindPower(int);
  };
}

#endif
```



```
#include <vector>
#include <iterator>
#include "window_game.h"
#include "yaml.h"
#include "worm.h"
#include "camera.h"
#include <limits.h>

#define MAP_WIDTH 2500
#define MAP_HEIGHT 1500
#define BACKGROUND_PATH "/usr/etc/worms/temp/background.png"

View::WindowGame::WindowGame(YAML::Node & staticNode, int w, int h, bool fs, boo
l ed_mode) : staticMap(staticNode) {
        this->screen_width = w;
        this->screen_height = h;
        this->full_screen = fs;
        this->edition_mode = ed_mode;
        this->init();
        if (this->edition_mode) {
                this->background.loadFromFile(this->staticMap["background"]["file"]
.as<std::string>(), renderer);
        } else {
                this->background.loadFromFile(BACKGROUND_PATH, renderer);
        }
        this->backgroundDisplayMode = this->staticMap["background"]["display"].as<st
d::string>();
        this->loadStaticObjects();
        this->water.init(
                this->renderer,
                0,
                MAP_HEIGHT - this->staticMap["water_level"].as<int>(),
                MAP_WIDTH,
                MAP_HEIGHT,
                gPath.PATH_WATER_2.c_str()
        );
}

void View::WindowGame::loadStaticObjects(void) {
        const YAML::Node & nodeShortGirders = this->staticMap["short_girders"];
        const YAML::Node & nodeLongGirders = this->staticMap["long_girders"];
        YAML::const_iterator it;

        for (it = nodeShortGirders.begin() ; it != nodeShortGirders.end() ; it++
) {
                const YAML::Node & eachGirder = *it;
                View::GirderShort * newShortGirder = new View::GirderShort(this-
>renderer, eachGirder["angle"].as<int>());
                newShortGirder->setX(eachGirder["x"].as<int>());
                newShortGirder->setY(eachGirder["y"].as<int>());
                this->shortGirders.push_back(newShortGirder);
        }

        for (it = nodeLongGirders.begin() ; it != nodeLongGirders.end() ; it++)
{
                const YAML::Node & eachGirder = *it;
                View::GirderLong * newLongGirder = new View::GirderLong(this->re
nderer, eachGirder["angle"].as<int>());
                newLongGirder->setX(eachGirder["x"].as<int>());
                newLongGirder->setY(eachGirder["y"].as<int>());
                this->longGirders.push_back(newLongGirder);
        }
```

```
}

View::WindowGame::~WindowGame() {
    this->close();
}

int View::WindowGame::getWidthResolution(void) {
        SDL_DisplayMode DM;
        SDL_GetCurrentDisplayMode(0, &DM);
        return DM.w;
}

int View::WindowGame::getHeightResolution(void) {
        SDL_DisplayMode DM;
        SDL_GetCurrentDisplayMode(0, &DM);
        return DM.h;
}

void View::WindowGame::init(void) {
        // Initialize SDL
        if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO) < 0) {
        throw View::Exception("%s. SDL Error: %s", ERR_MSG_SDL_INIT_VIDEO, SDL_GetEr
ror());
        } else {
                // Set texture filtering to linear
                if (!SDL_SetHint( SDL_HINT_RENDER_SCALE_QUALITY, "1")) {
                        printf( "Warning: Linear texture filtering not enabled!" );
                }

                // Si se inicio con los argumentos en 0 se inicializa con
                // el tamano de la pantalla
                if (this->full_screen) {
                        this->screen_width = this->getWidthResolution();
                        this->screen_height = this->getHeightResolution();
                }

                // Create window
                this->window = SDL_CreateWindow(
            "Worms Taller Party",
            SDL_WINDOWPOS_UNDEFINED,
            SDL_WINDOWPOS_UNDEFINED,
            this->screen_width,
            this->screen_height,
            this->full_screen ? SDL_WINDOW_FULLSCREEN : SDL_WINDOW_SHOWN
        );

                if (this->window == NULL) {
                        throw View::Exception("%s. SDL Error: %s", ERR_MSG_SDL_CREAT
E_WINDOW, SDL_GetError());
                } else {
                        //Create this->renderer for window
                        //this->renderer = SDL_CreateRenderer( gWindow, -1, SDL_
RENDERER_ACCELERATED );
                        this->renderer = SDL_CreateRenderer(this->window, -1, SD
L_RENDERER_ACCELERATED);
                        if (this->renderer == NULL) {
                                throw View::Exception("%s. SDL Error: %s", ERR_MSG_S
DL_CREATE_RENDERER, SDL_GetError());
                        } else {
                                //Initialize this->renderer color
                                SDL_SetRenderDrawColor( this->renderer, 0xFF, 0x
FF, 0xFF, 0xFF );
```



```
                                //Initialize PNG loading
                                int imgFlags = IMG_INIT_PNG;
                                if (!(IMG_Init(imgFlags) & imgFlags)) {
                                        throw View::Exception("%s. SDL Error: %s", E
RR_MSG_SDL_IMAGE_INIT, IMG_GetError());
                                }

                                if (Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2,
2048) < 0) {
                                        throw View::Exception("%s. SDL Error: %s", E
RR_MSG_SDL_MIXER_INIT, Mix_GetError());
                                }

                                // Inicializamos TTF
                                if (TTF_Init() == -1) {
                                        throw View::Exception("%s. SDL Error: %s", E
RR_MSG_SDL_TTF_INIT, TTF_GetError());
                                }
                        }
                }
        }
}

void View::WindowGame::close(void) {
        //Destroy window
        SDL_DestroyRenderer(this->renderer);
        SDL_DestroyWindow(this->window);
        this->window = NULL;
        this->renderer = NULL;

        //Destroy girders
        for (size_t i = 0 ; i < this->longGirders.size() ; i++) {
                delete longGirders[i];
        }

        for (size_t i = 0 ; i < this->shortGirders.size() ; i++) {
                delete shortGirders[i];
        }

        //Quit SDL subsystems
        IMG_Quit();
        SDL_Quit();
        Mix_Quit();
        TTF_Quit();
}

SDL_Renderer * View::WindowGame::getRenderer(void) const {
        return this->renderer;
}

int View::WindowGame::getScreenWidth(void) const {
        return this->screen_width;
}

int View::WindowGame::getScreenHeight(void) const {
        return this->screen_height;
}

int View::WindowGame::getBgWidth(void) const {
        //return this->background.getWidth();
        return MAP_WIDTH;
}
```

```
int View::WindowGame::getBgHeight(void) const {
        //return this->background.getHeight();
        return MAP_HEIGHT;
}

void View::WindowGame::render(View::Camera & camera) {
        this->renderBackground(camera);

        std::vector<View::GirderLong *>::iterator it_l;
        for (it_l = this->longGirders.begin(); it_l != this->longGirders.end();
it_l++) {
                (*it_l)->render(this->renderer, camera.getX(), camera.getY());
        }

        std::vector<View::GirderShort *>::iterator it_s;
        for (it_s = this->shortGirders.begin(); it_s != this->shortGirders.end()
; it_s++) {
                (*it_s)->render(this->renderer, camera.getX(), camera.getY());
        }
}

void View::WindowGame::renderWater(View::Camera & camera) {
        this->water.render(this->renderer, camera.getX(), camera.getY());
}

void View::WindowGame::renderBackground(Camera & c) {
        // Expandida
        if (this->backgroundDisplayMode == "expanded") {
                this->background.render(this->renderer, 0 - c.getX(), 0 - c.getY
(), MAP_WIDTH, MAP_HEIGHT);
                return;
        }

        // Mosaico
        int bgW = this->background.getWidth();
        int bgH = this->background.getHeight();
        if (this->backgroundDisplayMode == "mosaic") {
                for (size_t i = 0 ; i * bgW < MAP_WIDTH ; i++) {
                        for (size_t j = 0 ; j * bgH < MAP_HEIGHT ; j++) {
                                this->background.render(this->renderer, i * bgW
- c.getX(), j * bgH - c.getY());
                        }
                }
        }

        // Centrado
        if (this->backgroundDisplayMode == "centered") {
                if (bgW < MAP_WIDTH && bgH < MAP_HEIGHT) {
                        this->background.render(this->renderer, (MAP_WIDTH - bgW
) / 2 - c.getX(), (MAP_HEIGHT - bgH) / 2 - c.getY());
                } else {
                        // Si la imagen es mas grande que el mapa
                        // se dibuja el fondo centrado
                        this->background.render(this->renderer, 0 - c.getX(), 0
- c.getY(), MAP_WIDTH, MAP_HEIGHT);
                        return;
                }
        }
}

void View::WindowGame::hide(void) {
```



```
        SDL_HideWindow(this->window);
}

void View::WindowGame::show(void) {
        SDL_ShowWindow(this->window);
}
```

```
#ifndef __WINDOW_GAME_H__
#define __WINDOW_GAME_H__

#include <SDL2/SDL.h>
#include <SDL2/SDL_mixer.h>
#include <vector>
#include <string>
#include "girder_short.h"
#include "girder_long.h"
#include "camera.h"
#include "texture.h"
#include "water.h"
#include "yaml.h"

namespace View {
class WindowGame;
}

class View::WindowGame {
  private:
    SDL_Renderer * renderer;
    SDL_Window* window;
     // Vector de vigas creadas
    std::vector<View::GirderLong *> longGirders;
    std::vector<View::GirderShort *> shortGirders;
    int screen_width;
    int screen_height;
    bool full_screen;
    bool edition_mode;

    YAML::Node & staticMap;
    Water water;

    // TODO: Crear clase background y encapsular esto
    View::Texture background;
    std::string backgroundDisplayMode;

    void init(void);
    bool loadMedia(void);
    void close(void);
    int getWidthResolution(void);
    int getHeightResolution(void);
    void loadStaticObjects(void);

  public:
    // Constructor para el cliente
    WindowGame(YAML::Node &, int w = 0, int h = 0, bool fs = false, bool ed_mode
 = false);

    // Constructor para el editor de mapas
    WindowGame(std::string pathToBg, int waterLevel);
    ~WindowGame();
    SDL_Renderer * getRenderer(void) const;
    int getScreenWidth(void) const;
    int getScreenHeight(void) const;
    int getBgWidth(void) const;
    int getBgHeight(void) const;
    void render(View::Camera &);

    // El agua debe ser lo ultimo que se dibuja
    void renderWater(View::Camera &);
```



```
    // Renderiza el fondo. Se puede programar
    // para que sea en mosaico, centrado o expandido
    void renderBackground(View::Camera &);

    // Esconde la ventana
    void hide(void);

    // Muestra la ventana
    void show(void);
};

#endif
```

```
#include <SDL2/SDL.h>
#include <string>
#include "worm.h"

#define PADDING 1
#define RECTANGLE_HEIGHT 22
#define DISTANCE_TEXT_FROM_WORM 30
#define FPC 2

View::Worm::Worm(SDL_Renderer * r, std::string name, size_t team, int health) :
  sight(r),
  team(team),
  health(health),
  font(gPath.PATH_FONT_ARIAL_BOLD, 20),
  healthTxt(RECTANGLE_HEIGHT, PADDING, gPath.PATH_FONT_ARIAL_BOLD),
  nameTxt(RECTANGLE_HEIGHT, PADDING, gPath.PATH_FONT_ARIAL_BOLD) {

  SDL_Color colors[] = {
    {0, 0, 0, 0},
    {255, 0, 0, 0},
    {0, 255, 0, 0},
    {0, 0, 255, 0}
  };

  this->mirrored = false;
  this->walking = false;
  this->alive = true;
  this->falling = false;
  this->protagonic = false;

  this->x = 0;
  this->y = 0;
  this->inclination = NONE;
  this->angleDirection = 0;
  this->name = name;

  this->nameText.loadFromRenderedText(r, this->font, name, colors[this->team]);
  this->healthText.loadFromRenderedText(r, this->font, std::to_string(this->heal
th), colors[this->team]);

  this->healthTxt.setTextColor(colors[this->team]);
  this->nameTxt.setTextColor(colors[this->team]);

  this->healthTxt.setText(r, std::to_string(this->health));
  this->nameTxt.setText(r, name);

  this->dataConfiguration = ALL;

  this->states[WS_BREATHING] = new View::Breathing(this, r);
  this->states[WS_WALKING] = new View::Walking(this, r);
  this->states[WS_FALLING] = new View::Falling(this, r);
  this->states[WS_FLYING] = new View::Flying(this, r);
  this->states[WS_DEAD] = new View::Dead(this, r);

  this->state = this->states[WS_BREATHING];
  this->stateName = WS_BREATHING;
}

View::Worm::~Worm() {
  std::map<view_worm_state_t, WormState *>::iterator it = this->states.begin();
  for (; it != this->states.end() ; it++) {
    delete it->second;
  }
}

int View::Worm::getWidth(void) const {
  return 60;
}

int View::Worm::getHeight(void) const {
  return 60;
}

int View::Worm::getX(void) const {
  return this->x;
}

int View::Worm::getY(void) const {
  return this->y;
}

void View::Worm::setX(int x) {
  this->healthTxt.setX(x);
  this->nameTxt.setX(x);
  this->x = x;
}

void View::Worm::setY(int y) {
  this->healthTxt.setY(y - this->healthTxt.getHeight() / 2 - DISTANCE_TEXT_FROM_
WORM);
  this->nameTxt.setY(this->healthTxt.getY() - this->nameTxt.getHeight() / 2);
  this->y = y;
}

void View::Worm::setState(view_worm_state_t newState) {
  this->state->resetAnimation();
  this->stateName = newState;
  this->state = this->states[newState];
}

void View::Worm::updateState(const YAML::Node & status) {
  if (this->stateName == WS_DEAD) {
    return;
  }

  this->mirrored = status["mirrored"].as<int>();
  this->inclination = (worm_inclination_t)status["inclination"].as<int>();
  bool walking = status["walking"].as<int>();
  bool falling = status["falling"].as<int>();
  bool grounded = status["grounded"].as<int>();
  this->affectedByExplosion = status["affected_by_explosion"].as<int>();
  this->angleDirection = status["angle_direction"].as<int>();

  if (affectedByExplosion) {
    if (this->stateName != WS_FLYING) {
      this->setState(WS_FLYING);
      return;
    }
  }

  if (grounded && !walking && !affectedByExplosion) {
    if (this->stateName != WS_BREATHING) {
      this->setState(WS_BREATHING);
      return;
```

```
    }
  }

  if (walking && !affectedByExplosion) {
    if (this->stateName != WS_WALKING) {
      this->setState(WS_WALKING);
      return;
    }
  }

  if (falling && !affectedByExplosion) {
    if (this->stateName != WS_FALLING) {
      this->setState(WS_FALLING);
      return;
    }
  }
}

void View::Worm::render(SDL_Renderer * r, int camX, int camY) {
  this->state->render(r, camX, camY, this->inclination, this->mirrored, this->an
gleDirection);

  // Display de la data
  this->renderWormData(r, camX, camY);

  // Display sight if protagonic
  if (this->protagonic) {
    this->sight.setXYcenter(this->x, this->y);
    this->sight.setMirrored(this->mirrored);
    this->sight.render(r, camX, camY);
  }
}

void View::Worm::renderWormData(SDL_Renderer * r, int camX, int camY) {
  if (this->stateName == WS_DEAD) {
    return;
  }

  if (this->dataConfiguration != NO_DATA) {
    this->healthTxt.setText(r, std::to_string(this->health));
    this->healthTxt.render(r, camX, camY);

    if (this->dataConfiguration == ALL) {
      this->nameTxt.render(r, camX, camY);
    }
  }
}

void View::Worm::setHealth(int newHealth) {
  if (this->stateName != WS_DEAD) {
    this->health = newHealth;
    if (this->health <= 0) {
      this->setState(WS_DEAD);
    }
  }
}

int View::Worm::getHealth(void) {
  return this->health;
}

bool View::Worm::isAlive(void) {
```



```
  return this->alive;
}

void View::Worm::setMirrored(bool mirr) {
  this->mirrored = mirr;
}

void View::Worm::setWalking(bool walk) {
  this->walking = walk;
}

void View::Worm::setFalling(bool fall) {
  this->falling = fall;
}

void View::Worm::setGrounded(bool grd) {
  this->grounded = grd;
}

void View::Worm::setProtagonic(bool p) {
  this->protagonic = p;
}

void View::Worm::setSightAngle(int angle) {
  this->sight.setAngle(angle);
}

void View::Worm::setAffectedByExplosion(bool af) {
  this->affectedByExplosion = af;
}

void View::Worm::setDataConfiguration(worm_data_cfg_t config) {
  this->dataConfiguration = config;
}

bool View::Worm::isAffectedByExplosion() {
  return this->affectedByExplosion;
}

std::string View::Worm::getName() const {
  return this->name;
}
```

```
#ifndef __VIEW_WORM_H__
#define __VIEW_WORM_H__

#include <SDL2/SDL.h>
#include <map>
#include <string>

// States
#include "worm_state.h"
#include "breathing.h"
#include "walking.h"
#include "falling.h"
#include "flying.h"
#include "dead.h"

#include "texture.h"
#include "drawable.h"
#include "paths.h"
#include "types.h"
#include "rectangle_text.h"
#include "sprite_animation.h"
#include "sound_effect.h"
#include "font.h"
#include "sight.h"
#include "yaml.h"

namespace View {
  class WormState;

  class Worm: public Drawable {
    private:
      // Animation
      std::map<view_worm_state_t, WormState *> states;

      View::WormState * state;
      view_worm_state_t stateName;

      // Animation state
      bool grounded;
      bool mirrored;
      bool walking;
      bool falling;
      bool alive;
      bool protagonic;
      bool affectedByExplosion;
      int angleDirection;
      worm_inclination_t inclination;
      std::string name;



      Sight sight;


      // Worm data
      size_t team;
      int health;

      // Worm data UI
      worm_data_cfg_t dataConfiguration;
      Font font;
      Texture nameText;
```



```
      Texture healthText;
      RectangleText healthTxt;
      RectangleText nameTxt;

      // Setea el nuevo state del worm
      void setState(view_worm_state_t);

    public:
      Worm(SDL_Renderer *, std::string, size_t, int);
      virtual ~Worm(void);
      virtual int getWidth(void) const;
      virtual int getHeight(void) const;
      virtual int getX(void) const;
      virtual int getY(void) const;
      std::string getName(void) const;
      virtual void setX(int);
      virtual void setY(int);
      virtual void render(SDL_Renderer *, int, int);
      void updateState(const YAML::Node &);
      void setProtagonic(bool);
      void setMirrored(bool);
      void setWalking(bool);
      void renderWormData(SDL_Renderer *, int, int);
      void setHealth(int);
      void setFalling(bool);
      void setGrounded(bool);
      void setAffectedByExplosion(bool);
      void setSightAngle(int);
      void setDataConfiguration(worm_data_cfg_t);
      int getHealth(void);
      bool isAlive(void);
      bool isAffectedByExplosion(void);
  };
}

#endif
```

```
#include <string>
#include <SDL2/SDL.h>
#include "worms_status.h"
#include "worm.h"

View::WormsStatus::WormsStatus(YAML::Node & nodeWorms, SDL_Renderer * rend) {
        YAML::const_iterator itTeam;
        for (itTeam = nodeWorms.begin() ; itTeam != nodeWorms.end() ; itTeam++)
{
                int teamId = itTeam->first.as<int>();
                const YAML::Node & eachTeam = itTeam->second["worms"];
                YAML::const_iterator itWorms;
                for (itWorms = eachTeam.begin() ; itWorms != eachTeam.end() ; it
Worms++) {
                        const YAML::Node & eachWorm = *itWorms;
                        View::Worm * newWorm = new View::Worm(rend, eachWorm["na
me"].as<std::string>(), teamId, eachWorm["health"].as<int>());
                        newWorm->setX(eachWorm["x"].as<int>());
                        newWorm->setY(eachWorm["y"].as<int>());
                        this->worms[eachWorm["id"].as<size_t>()] = newWorm;
                }
        }
}

void View::WormsStatus::render(SDL_Renderer * renderer, View::Camera & camera) {
        std::map<size_t, View::Worm *>::iterator it;

        for (it = this->worms.begin(); it != this->worms.end(); it++) {
                it->second->render(renderer, camera.getX(), camera.getY());
        }
}

void View::WormsStatus::update(const YAML::Node & wormsNode) {
        View::Worm * worm;
        YAML::const_iterator itTeam;
        for (itTeam = wormsNode.begin() ; itTeam != wormsNode.end() ; itTeam++)
{
                const YAML::Node & eachTeam = itTeam->second["worms"];
                YAML::const_iterator itWorms;
                for (itWorms = eachTeam.begin() ; itWorms != eachTeam.end() ; it
Worms++) {
                        const YAML::Node & eachWorm = *itWorms;
                        worm = this->worms[eachWorm["id"].as<size_t>()];
                        worm->setX(eachWorm["x"].as<int>());
                        worm->setY(eachWorm["y"].as<int>());
                        worm->setHealth(eachWorm["health"].as<int>());
                        worm->updateState(eachWorm["status"]);
                }
        }
}

void View::WormsStatus::updateWormProtagonic(size_t wormId) {
        std::map<size_t, View::Worm *>::const_iterator it = this->worms.begin();
        for (; it != this->worms.end() ; it++) {
                View::Worm * eachWorm = it->second;
                eachWorm->setProtagonic(it->first == wormId);
        }
}

void View::WormsStatus::updateWormsClientConfiguration(ClientConfiguration & cfg
) {
        std::map<size_t, View::Worm *>::const_iterator it = this->worms.begin();
```



```
        for (; it != this->worms.end() ; it++) {
                View::Worm * eachWorm = it->second;
                eachWorm->setDataConfiguration(cfg.getWormDataConfiguration());
                eachWorm->setSightAngle(cfg.getSightAngle());
        }
}

View::WormsStatus::~WormsStatus(void) {
        std::map<size_t, View::Worm *>::iterator it;

        for (it = this->worms.begin(); it != this->worms.end(); it++) {
                delete it->second;
        }
}

const View::Worm * View::WormsStatus::getWormView(size_t id) {
        if (this->worms.find(id) != this->worms.end()) {
                return this->worms[id];
        } else {
                return nullptr;
        }
}

const View::Worm * View::WormsStatus::getWormAffectedByExplosion() {
        std::map<size_t, View::Worm *>::iterator it;
        for (it = this->worms.begin(); it != this->worms.end(); it++) {
                if (it->second->isAffectedByExplosion()) {
                        return this->getWormView(it->first);
                }
        }
        return nullptr;
}
```

```
#ifndef __WORMS_STATUS_H__
#define __WORMS_STATUS_H__

#include <SDL2/SDL.h>
#include <map>
#include "yaml.h"
#include "worm.h"
#include "camera.h"
#include "client_configuration.h"

namespace View {
    class WormsStatus;
}

class View::WormsStatus {
    private:
        std::map<size_t, View::Worm *> worms;
    public:
        ~WormsStatus(void);
        WormsStatus(YAML::Node &, SDL_Renderer * r);
        void render(SDL_Renderer *, View::Camera &);
        void update(const YAML::Node &);
        void updateWormProtagonic(size_t);
        void updateWormsClientConfiguration(ClientConfiguration &);

        // Devuelve un puntero constante a la vista
        // del worm con el id pasado por parametro
        // en caso de no existir devuelve NULL
        const View::Worm * getWormView(size_t id);
        const View::Worm * getWormAffectedByExplosion();

};

#endif
```



```
#ifndef __WORM_STATE_H__
#define __WORM_STATE_H__

#include <SDL2/SDL.h>
#include "types.h"
#include "paths.h"

namespace View {
  class Worm;

  class WormState {
    protected:
      View::Worm * context;
      view_worm_state_t state;

    public:
      virtual ~WormState() {};
      virtual void render(SDL_Renderer *, int, int, worm_inclination_t, bool, in
t a = -1) = 0;
      virtual void resetAnimation(void) = 0;
      view_worm_state_t getState(void) {
        return this->state;
      }
  };
}

#endif
```

jun 26, 18 12:47 **Table of Content** Page 1/2

Table of Contents

jun 26, 18 12:47 **Table of Content** Page 2/2