jun 26, 18 17:16 **Walk.h** Page 1/1

```
#ifndef _PLAYERWALKLEFT_H
#define _PLAYERWALKLEFT_H

#include "../Config/Config.h"
#include "PlayerState.h"

namespace Worms {
class Walk : public State {
   public:
    Walk();
    ~Walk() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    virtual void pointUp(Player &p) override;
    virtual void pointDown(Player &p) override;

   private:
    const float walkVelocity;
    float timeElapsed{0.0f};
};
}

#endif  //_PLAYERWALKLEFT_H
```

jun 26, 18 17:16 **Walk.cpp** Page 1/2

```

#include <cmath>
#include <iostream>
#include <memory>

#include "../Player.h"
#include "Still.h"
#include "Walk.h"

void Worms::Walk::update(Player &p, float dt, b2Body *body) {
    float32 mass = body→GetMass();
    b2Vec2 vel = body→GetLinearVelocity();

    float final_vel{0.0f};

    if (¬p.isOnGround()) {
        this→impulses[0] = −vel.x * mass;
        body→ApplyLinearImpulse(b2Vec2(impulses[0], impulses[1]), body→GetWorl
dCenter(), true);
        p.notify(p, Event::WormFalling);
        p.setState(Worm::StateID::Falling);
        return;
    }

    if (p.direction ≡ Worm::Direction::left) {
        final_vel = −this→walkVelocity;
    } else {
        final_vel = this→walkVelocity;
    }

    this→impulses[0] = mass * (final_vel − vel.x);
    body→ApplyLinearImpulse(b2Vec2(this→impulses[0], this→impulses[1]), body→
GetWorldCenter(),
                            true);

    p.lastWalkDirection = p.direction;

    this→timeElapsed += dt;
}

void Worms::Walk::moveRight(Worms::Player &p) {
    p.direction = Worm::Direction::right;
}

void Worms::Walk::moveLeft(Worms::Player &p) {
    p.direction = Worm::Direction::left;
}

void Worms::Walk::stopMove(Worms::Player &p) {
    p.setState(Worm::StateID::Still);
}

void Worms::Walk::jump(Worms::Player &p) {}

Worms::Walk::Walk()
    : State(Worm::StateID::Walk), walkVelocity(Game::Config::getInstance().getWa
lkVelocity()) {}

void Worms::Walk::backFlip(Worms::Player &p) {}

void Worms::Walk::bazooka(Worms::Player &p) {}

void Worms::Walk::pointUp(Worms::Player &p) {}

void Worms::Walk::pointDown(Worms::Player &p) {}

```

```
64 void Worms::Walk::startShot(Worms::Player &p) {}
65
66 void Worms::Walk::endShot(Worms::Player &p) {}
67
68 void Worms::Walk::grenade(Worms::Player &p) {}
69
70 void Worms::Walk::cluster(Worms::Player &p) {}
71
72 void Worms::Walk::mortar(Worms::Player &p) {}
73
74 void Worms::Walk::banana(Worms::Player &p) {}
75
76 void Worms::Walk::holy(Worms::Player &p) {}
77
78 void Worms::Walk::setTimeout(Worms::Player &p, uint8_t time) {}
79
80 void Worms::Walk::aerialAttack(Worms::Player &p) {}
81
82 void Worms::Walk::dynamite(Worms::Player &p) {}
83
84 void Worms::Walk::teleport(Worms::Player &p) {}
85
86 void Worms::Walk::baseballBat(Worms::Player &p) {}
```



```
1  //
2  // Created by rodrigo on 16/06/18.
3  //
4
5  #ifndef INC_4_WORMS_TELEPORTING_H
6  #define INC_4_WORMS_TELEPORTING_H
7
8  #include <Camera.h>
9  #include <stdint-gcc.h>
10 #include <cstdint>
11 #include "PlayerState.h"
12
13 namespace Worms {
14 class Teleporting : public State {
15    public:
16     Teleporting(GUI::Position p);
17     ~Teleporting() = default;
18     void update(Player &p, float dt, b2Body *body) override;
19     void moveRight(Player &p) override;
20     void moveLeft(Player &p) override;
21     void jump(Player &p) override;
22     void setTimeout(Player &p, uint8_t time) override;
23
24     void bazooka(Player &p) override;
25     void grenade(Player &p) override;
26     void cluster(Player &p) override;
27     void mortar(Player &p) override;
28     void banana(Player &p) override;
29     void holy(Player &p) override;
30     void aerialAttack(Player &p) override;
31     void dynamite(Player &p) override;
32     void baseballBat(Player &p) override;
33     void teleport(Player &p) override;
34
35     void startShot(Player &p) override;
36     void endShot(Player &p) override;
37     void backFlip(Player &p) override;
38     void stopMove(Player &p) override;
39     void pointUp(Player &p) override;
40     void pointDown(Player &p) override;
41
42    private:
43     float timeElapsed{0.0f};
44     GUI::Position newPosition;
45     float teleportTime;
46 };
47 }
48
49 #endif  // INC_4_WORMS_TELEPORTING_H
```

jun 26, 18 17:16 **Teleporting.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 16/06/18.
//

#include "Teleporting.h"
#include <Camera.h>
#include "../Config/Config.h"
#include "../Player.h"

Worms::Teleporting::Teleporting(GUI::Position p)
    : State(Worm::StateID::Teleporting),
      newPosition(p),
      teleportTime(Game::Config::getInstance().getTeleportTime()) {}

void Worms::Teleporting::update(Worms::Player &p, float dt, b2Body *body) {
    this→timeElapsed += dt;
    if (this→timeElapsed ≥ this→teleportTime) {
        p.setPosition(this→newPosition);
        p.setState(Worm::StateID::Teleported);
    }
}

void Worms::Teleporting::moveRight(Worms::Player &p) {}

void Worms::Teleporting::moveLeft(Worms::Player &p) {}

void Worms::Teleporting::jump(Worms::Player &p) {}

void Worms::Teleporting::stopMove(Worms::Player &p) {}

void Worms::Teleporting::backFlip(Worms::Player &p) {}

void Worms::Teleporting::bazooka(Worms::Player &p) {}

void Worms::Teleporting::pointUp(Worms::Player &p) {}

void Worms::Teleporting::pointDown(Worms::Player &p) {}

void Worms::Teleporting::startShot(Worms::Player &p) {}

void Worms::Teleporting::endShot(Worms::Player &p) {}

void Worms::Teleporting::grenade(Worms::Player &p) {}

void Worms::Teleporting::cluster(Worms::Player &p) {}

void Worms::Teleporting::mortar(Worms::Player &p) {}

void Worms::Teleporting::banana(Worms::Player &p) {}

void Worms::Teleporting::holy(Worms::Player &p) {}

void Worms::Teleporting::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::Teleporting::aerialAttack(Worms::Player &p) {}

void Worms::Teleporting::dynamite(Worms::Player &p) {}

void Worms::Teleporting::teleport(Worms::Player &p) {}

void Worms::Teleporting::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **Teleported.h** Page 1/1

```cpp
//
// Created by rodrigo on 16/06/18.
//

#ifndef INC_4_WORMS_TELEPORTED_H
#define INC_4_WORMS_TELEPORTED_H

#include <stdint-gcc.h>
#include <cstdint>
#include "PlayerState.h"

namespace Worms {
class Teleported : public State {
   public:
    Teleported();
    ~Teleported() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    void pointUp(Player &p) override;
    void pointDown(Player &p) override;

   private:
    float timeElapsed{0.0f};
    float teleportTime;
};
}

#endif  // INC_4_WORMS_TELEPORTED_H
```

jun 26, 18 17:16 **Teleported.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 16/06/18.
//

#include "Teleported.h"
#include "../Config/Config.h"
#include "../Player.h"

Worms::Teleported::Teleported()
    : State(Worm::StateID::Teleported),
      teleportTime(Game::Config::getInstance().getTeleportTime()) {}

void Worms::Teleported::update(Worms::Player &p, float dt, b2Body *body) {
    this→timeElapsed += dt;
    if (this→timeElapsed ≥ this→teleportTime){
        p.setState(Worm::StateID::Falling);
    }
}

void Worms::Teleported::moveRight(Worms::Player &p) {}

void Worms::Teleported::moveLeft(Worms::Player &p) {}

void Worms::Teleported::jump(Worms::Player &p) {}

void Worms::Teleported::stopMove(Worms::Player &p) {}

void Worms::Teleported::backFlip(Worms::Player &p) {}

void Worms::Teleported::bazooka(Worms::Player &p) {}

void Worms::Teleported::pointUp(Worms::Player &p) {}

void Worms::Teleported::pointDown(Worms::Player &p) {}

void Worms::Teleported::startShot(Worms::Player &p) {}

void Worms::Teleported::endShot(Worms::Player &p) {}

void Worms::Teleported::grenade(Worms::Player &p) {}

void Worms::Teleported::cluster(Worms::Player &p) {}

void Worms::Teleported::mortar(Worms::Player &p) {}

void Worms::Teleported::banana(Worms::Player &p) {}

void Worms::Teleported::holy(Worms::Player &p) {}

void Worms::Teleported::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::Teleported::aerialAttack(Worms::Player &p) {}

void Worms::Teleported::dynamite(Worms::Player &p) {}

void Worms::Teleported::teleport(Worms::Player &p) {}

void Worms::Teleported::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **Still.h** Page 1/1

```cpp
//
// Created by Gorco on 19/05/18.
//

#ifndef INC_4_WORMS_STOPMOVE_H
#define INC_4_WORMS_STOPMOVE_H

#include <Box2D/Common/b2Math.h>
#include <vector>

#include "PlayerState.h"

namespace Worms {
class Still : public State {
   public:
    Still();
    ~Still() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    void pointUp(Player &p) override;
    void pointDown(Player &p) override;
};
}

#endif  // INC_4_WORMS_STOPMOVE_H
```

jun 26, 18 17:16 **Still.cpp** Page 1/2

```
//
// Created by Gorco on 19/05/18.
//

#include <cstdint>
#include <iostream>
#include <memory>

#include "../Player.h"
#include "Still.h"
#include "Walk.h"

Worms::Still::Still() : State(Worm::StateID::Still) {}

void Worms::Still::update(Player &p, float dt, b2Body *body) {
    float32 mass = body→GetMass();
    b2Vec2 vel = body→GetLinearVelocity();

    this→impulses[0] = −vel.x * mass;
    body→ApplyLinearImpulse(b2Vec2(impulses[0], impulses[1]), body→GetWorldCen
ter(), true);
}

void Worms::Still::moveRight(Worms::Player &p) {
    p.direction = Worm::Direction::right;
    p.setState(Worm::StateID::Walk);
}

void Worms::Still::moveLeft(Worms::Player &p) {
    p.direction = Worm::Direction::left;
    p.setState(Worm::StateID::Walk);
}

void Worms::Still::stopMove(Worms::Player &p) {}

void Worms::Still::jump(Worms::Player &p) {
    p.notify(p, Event::WormFalling);
    p.setState(Worm::StateID::StartJump);
}

void Worms::Still::backFlip(Worms::Player &p) {
    p.notify(p, Event::WormFalling);
    p.setState(Worm::StateID::StartBackFlip);
}

void Worms::Still::bazooka(Worms::Player &p) {
    p.setWeapon(Worm::WeaponID::WBazooka);
}

void Worms::Still::pointUp(Worms::Player &p) {
    p.increaseWeaponAngle();
}

void Worms::Still::pointDown(Worms::Player &p) {
    p.decreaseWeaponAngle();
}

void Worms::Still::startShot(Worms::Player &p) {
    p.startShot();
}

void Worms::Still::endShot(Worms::Player &p) {
    p.endShot();
}

void Worms::Still::grenade(Worms::Player &p) {
```

jun 26, 18 17:16 **Still.cpp** Page 2/2

```
    p.setWeapon(Worm::WeaponID::WGrenade);
}

void Worms::Still::cluster(Worms::Player &p) {
    p.setWeapon(Worm::WeaponID::WCluster);
}

void Worms::Still::mortar(Worms::Player &p) {
    p.setWeapon(Worm::WeaponID::WMortar);
}

void Worms::Still::banana(Worms::Player &p) {
    p.setWeapon(Worm::WeaponID::WBanana);
}

void Worms::Still::holy(Worms::Player &p) {
    p.setWeapon(Worm::WeaponID::WHoly);
}

void Worms::Still::setTimeout(Worms::Player &p, uint8_t time) {
    p.setWeaponTimeout(time);
}

void Worms::Still::aerialAttack(Worms::Player &p) {
    p.setWeapon(Worm::WeaponID::WAerial);
}

void Worms::Still::dynamite(Worms::Player &p) {
    p.setWeapon(Worm::WeaponID::WDynamite);
}

void Worms::Still::teleport(Worms::Player &p) {
    p.setWeapon(Worm::WeaponID::WTeleport);
}

void Worms::Still::baseballBat(Worms::Player &p) {
    p.setWeapon(Worm::WeaponID::WBaseballBat);
}
```

jun 26, 18 17:16 **StartJump.h** Page 1/1

```
//
// Created by Gorco on 19/05/18.
//

#ifndef __WORMS_PLAYER_JUMP_RIGHT_H__
#define __WORMS_PLAYER_JUMP_RIGHT_H__

#include <stdint-gcc.h>
#include <cstdint>
#include "../Config/Config.h"
#include "../Player.h"

namespace Worms {
class StartJump : public State {
   public:
    StartJump();
    ~StartJump() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void pointUp(Player &p) override;
    void pointDown(Player &p) override;

   private:
    float timeElapsed{0.0f};
    bool impulseApplied{false};
    const float jumpTime;
    const Math::Vector jumpVelocity;
};
}  // namespace Worms

#endif  // __WORMS_PLAYER_JUMP_RIGHT_H__
```

jun 26, 18 17:16 **StartJump.cpp** Page 1/2

```
//
// Created by Gorco on 19/05/18.
//

#include <iostream>

#include "../Config/Config.h"
#include "Direction.h"
#include "StartJump.h"

Worms::StartJump::StartJump()
    : State(Worm::StateID::StartJump),
      jumpTime(Game::Config::getInstance().getStartJumpTime()),
      jumpVelocity(Game::Config::getInstance().getJumpVelocity()) {}

void Worms::StartJump::update(Player &p, float dt, b2Body *body) {
    this→timeElapsed += dt;
    if (this→timeElapsed ≥ this→jumpTime) {
        if (¬this→impulseApplied) {
            float32 mass = body→GetMass();
            b2Vec2 impulses = {mass * this→jumpVelocity.x, mass * this→jumpVel
ocity.y};
            if (p.direction ≡ Worm::Direction::left) {
                impulses.x *= -1;
            }
            /* When the worm jumps, it needs an initial impulse in the y axis
             * that will never will be applied again. In the x axis, the worms
             * moves in RUM, so it needs an initial impulse (because his frictio
n
             * coeficient is 0) and then needs an end impulse, of equal absolute
             * value and different sign.
             */
            body→ApplyLinearImpulse(impulses, body→GetWorldCenter(), true);
            this→impulseApplied = true;
        } else if (¬p.isOnGround()) {
            p.setState(Worm::StateID::Jumping);
        } else if (this→timeElapsed > 0.9f) {
            p.setState(Worm::StateID::Still);
        }
    }
}

void Worms::StartJump::moveRight(Worms::Player &p) {}

void Worms::StartJump::moveLeft(Worms::Player &p) {}

void Worms::StartJump::jump(Worms::Player &p) {}

void Worms::StartJump::stopMove(Worms::Player &p) {}

void Worms::StartJump::backFlip(Worms::Player &p) {}

void Worms::StartJump::bazooka(Worms::Player &p) {}

void Worms::StartJump::pointUp(Worms::Player &p) {}

void Worms::StartJump::pointDown(Worms::Player &p) {}

void Worms::StartJump::startShot(Worms::Player &p) {}

void Worms::StartJump::endShot(Worms::Player &p) {}

void Worms::StartJump::grenade(Worms::Player &p) {}

void Worms::StartJump::cluster(Worms::Player &p) {}

```

jun 26, 18 17:16 **StartJump.cpp** Page 2/2

```
65  void Worms::StartJump::mortar(Worms::Player &p) {}
66
67  void Worms::StartJump::banana(Worms::Player &p) {}
68
69  void Worms::StartJump::holy(Worms::Player &p) {}
70
71  void Worms::StartJump::setTimeout(Worms::Player &p, uint8_t time) {}
72
73  void Worms::StartJump::aerialAttack(Worms::Player &p) {}
74
75  void Worms::StartJump::dynamite(Worms::Player &p) {}
76
77  void Worms::StartJump::teleport(Worms::Player &p) {}
78
79  void Worms::StartJump::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **StartBackFlip.h** Page 1/1

```
1   /*
2    *  Created by Rodrigo.
3    *  date: 20/05/18
4    */
5
6   #ifndef __PLAYER_START_BACK_FLIP_H__
7   #define __PLAYER_START_BACK_FLIP_H__
8
9   #include <stdint-gcc.h>
10  #include <cstdint>
11  #include "../Config/Config.h"
12  #include "../Player.h"
13
14  namespace Worms {
15  class StartBackFlip : public State {
16     public:
17      StartBackFlip();
18      ~StartBackFlip() = default;
19      void update(Player &p, float dt, b2Body *body) override;
20      void moveRight(Player &p) override;
21      void moveLeft(Player &p) override;
22      void jump(Player &p) override;
23      void backFlip(Player &p) override;
24      void stopMove(Player &p) override;
25      void setTimeout(Player &p, uint8_t time) override;
26
27      void bazooka(Player &p) override;
28      void grenade(Player &p) override;
29      void cluster(Player &p) override;
30      void mortar(Player &p) override;
31      void banana(Player &p) override;
32      void holy(Player &p) override;
33      void aerialAttack(Player &p) override;
34      void dynamite(Player &p) override;
35      void baseballBat(Player &p) override;
36      void teleport(Player &p) override;
37
38      void startShot(Player &p) override;
39      void endShot(Player &p) override;
40      void pointUp(Player &p) override;
41      void pointDown(Player &p) override;
42
43     private:
44      float timeElapsed{0.0f};
45      bool impulseApplied{false};
46      const Math::Vector backflipVelocity;
47      const float startJumpTime;
48  };
49  }
50
51  #endif  //__PLAYER_START_BACK_FLIP_H__
```

jun 26, 18 17:16 **StartBackFlip.cpp** Page 1/2

```
/*
 *  Created by Rodrigo.
 *  date: 20/05/18
 */

#include "StartBackFlip.h"
#include "Direction.h"

Worms::StartBackFlip::StartBackFlip()
    : State(Worm::StateID::StartBackFlip),
      backflipVelocity(Game::Config::getInstance().getBackflipVelocity()),
      startJumpTime(Game::Config::getInstance().getStartJumpTime()) {}

void Worms::StartBackFlip::update(Worms::Player &p, float dt, b2Body *body) {
    this→timeElapsed += dt;
    if (this→timeElapsed ≥ this→startJumpTime) {
        if (¬this→impulseApplied) {
            float32 mass = body→GetMass();
            b2Vec2 impulses = {mass * this→backflipVelocity.x, mass * this→bac
kflipVelocity.y};
            if (p.direction ≡ Worm::Direction::left) {
                impulses.x *= -1;
            }
            /* When the worm jumps, it needs an initial impulse in the y axis
             * that will never will be applied again. In the x axis, the worms
             * moves in RUM, so it needs an initial impulse (because his frictio
n
             * coeficient is 0) and then needs an end impulse, of equal absolute
             * value and different sign.
             */
            body→ApplyLinearImpulse(impulses, body→GetWorldCenter(), true);
            this→impulseApplied = true;
        } else if (¬p.isOnGround()) {
            p.setState(Worm::StateID::BackFlipping);
        } else if (this→timeElapsed > 0.9f) {
            p.setState(Worm::StateID::Still);
        }
    }
}

void Worms::StartBackFlip::moveRight(Worms::Player &p) {}

void Worms::StartBackFlip::moveLeft(Worms::Player &p) {}

void Worms::StartBackFlip::jump(Worms::Player &p) {}

void Worms::StartBackFlip::backFlip(Worms::Player &p) {}

void Worms::StartBackFlip::stopMove(Worms::Player &p) {}

void Worms::StartBackFlip::bazooka(Worms::Player &p) {}

void Worms::StartBackFlip::pointUp(Worms::Player &p) {}

void Worms::StartBackFlip::pointDown(Worms::Player &p) {}

void Worms::StartBackFlip::startShot(Worms::Player &p) {}

void Worms::StartBackFlip::endShot(Worms::Player &p) {}

void Worms::StartBackFlip::grenade(Worms::Player &p) {}

void Worms::StartBackFlip::cluster(Worms::Player &p) {}

void Worms::StartBackFlip::mortar(Worms::Player &p) {}

```

jun 26, 18 17:16 **StartBackFlip.cpp** Page 2/2

```
void Worms::StartBackFlip::banana(Worms::Player &p) {}

void Worms::StartBackFlip::holy(Worms::Player &p) {}

void Worms::StartBackFlip::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::StartBackFlip::aerialAttack(Worms::Player &p) {}

void Worms::StartBackFlip::dynamite(Worms::Player &p) {}

void Worms::StartBackFlip::teleport(Worms::Player &p) {}

void Worms::StartBackFlip::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **Sliding.h** Page 1/1

```
#ifndef _PLAYER_SLIDING_H
#define _PLAYER_SLIDING_H

#include "../Config/Config.h"
#include "PlayerState.h"

namespace Worms {
class Sliding : public State {
   public:
    Sliding();
    ~Sliding() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    virtual void pointUp(Player &p) override;
    virtual void pointDown(Player &p) override;

   private:
    float timeElapsed{0.0f};
};
}  // namespace Worms

#endif  //_PLAYER_SLIDING_H
```

jun 26, 18 17:16 **Sliding.cpp** Page 1/2

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 20/05/18
 */

#include <iostream>
#include <vector>

#include "../Player.h"
#include "Sliding.h"

Worms::Sliding::Sliding() : State(Worm::StateID::Sliding) {}

void Worms::Sliding::update(Worms::Player &p, float dt, b2Body *body) {
    if (¬p.isOnGround()) {
        p.setState(Worm::StateID::Falling);
        return;
    }

    float final_vel{0.0f};

    try {
        b2Vec2 normal = p.getGroundNormal();
        float slope = std::abs(std::atan2(normal.y, normal.x));
        if ((slope < PI / 4.0f) ∨ (slope > (PI * 3.0f) / 4.0f)) {
            final_vel = 3.0f * normal.x;
            float impulse = body→GetMass() * (final_vel − body→GetLinearVeloci
ty().x);

            body→ApplyLinearImpulse(b2Vec2(impulse, 0.0f), body→GetWorldCenter
(), true);
        } else {
            p.setState(Worm::StateID::Land);
        }
    } catch (const Exception &e) {
    }
}

void Worms::Sliding::moveRight(Worms::Player &p) {}

void Worms::Sliding::moveLeft(Worms::Player &p) {}

void Worms::Sliding::jump(Worms::Player &p) {}

void Worms::Sliding::stopMove(Worms::Player &p) {}

void Worms::Sliding::backFlip(Worms::Player &p) {}

void Worms::Sliding::bazooka(Worms::Player &p) {}

void Worms::Sliding::pointUp(Worms::Player &p) {}

void Worms::Sliding::pointDown(Worms::Player &p) {}

void Worms::Sliding::startShot(Worms::Player &p) {}

void Worms::Sliding::endShot(Worms::Player &p) {}

void Worms::Sliding::grenade(Worms::Player &p) {}

void Worms::Sliding::cluster(Worms::Player &p) {}

void Worms::Sliding::mortar(Worms::Player &p) {}

void Worms::Sliding::banana(Worms::Player &p) {}

```

jun 26, 18 17:16 **Sliding.cpp** Page 2/2

```
65 void Worms::Sliding::holy(Worms::Player &p) {}
66
67 void Worms::Sliding::setTimeout(Worms::Player &p, uint8_t time) {}
68
69 void Worms::Sliding::aerialAttack(Worms::Player &p) {}
70
71 void Worms::Sliding::dynamite(Worms::Player &p) {}
72
73 void Worms::Sliding::teleport(Worms::Player &p) {}
74
75 void Worms::Sliding::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **PlayerState.h** Page 1/1

```
1  #ifndef _PLAYERSTATE_H
2  #define _PLAYERSTATE_H
3
4  #include <Box2D/Common/b2Math.h>
5  #include <Box2D/Dynamics/b2Body.h>
6  #include <vector>
7
8  #include "GameStateMsg.h"
9
10 namespace Worms {
11 class Player;
12 class State {
13    public:
14     explicit State(Worm::StateID id);
15     virtual ~State() = default;
16     virtual void update(Player &p, float dt, b2Body *body) = 0;
17     virtual void moveRight(Player &p) = 0;
18     virtual void moveLeft(Player &p) = 0;
19     virtual void jump(Player &p) = 0;
20     virtual void setTimeout(Player &p, uint8_t time) = 0;
21
22     virtual void bazooka(Player &p) = 0;
23     virtual void grenade(Player &p) = 0;
24     virtual void cluster(Player &p) = 0;
25     virtual void mortar(Player &p) = 0;
26     virtual void banana(Player &p) = 0;
27     virtual void holy(Player &p) = 0;
28     virtual void aerialAttack(Player &p) = 0;
29     virtual void dynamite(Player &p) = 0;
30     virtual void baseballBat(Player &p) = 0;
31     virtual void teleport(Player &p) = 0;
32
33     virtual void startShot(Player &p) = 0;
34     virtual void endShot(Player &p) = 0;
35     virtual void backFlip(Player &p) = 0;
36     virtual void stopMove(Player &p) = 0;
37     virtual void pointUp(Player &p) = 0;
38     virtual void pointDown(Player &p) = 0;
39     virtual Worm::StateID getState() const;
40
41    protected:
42     Worm::StateID stateID;
43     std::vector<float> impulses{0.0f, 0.0f};
44 };
45 }
46
47 #endif  //_PLAYERSTATE_H
```

jun 26, 18 17:16 **PlayerState.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 20/05/18
 */

#include "PlayerState.h"
#include "GameStateMsg.h"

Worms::State::State(Worm::StateID id) : stateID(id) {}

Worm::StateID Worms::State::getState() const {
    return this→stateID;
}
```

jun 26, 18 17:16 **Land.h** Page 1/1

```cpp
//
// Created by rodrigo on 3/06/18.
//

#ifndef INC_4_WORMS_LAND_H
#define INC_4_WORMS_LAND_H

#include <cstdint>
#include "PlayerState.h"

namespace Worms {
class Land : public State {
   public:
    Land();
    ~Land() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void pointUp(Player &p) override;
    void pointDown(Player &p) override;

   private:
    float timeElapsed{0.0f};
    float landTime;
};
}  // namespace Worms

#endif  // INC_4_WORMS_LAND_H
```

jun 26, 18 17:16 **Land.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 3/06/18.
//

#include "Land.h"
#include "../Config/Config.h"
#include "../Player.h"
#include "PlayerState.h"

Worms::Land::Land()
    : State(Worm::StateID::Land), landTime(Game::Config::getInstance().getLandTime()) {}

void Worms::Land::update(Worms::Player &p, float dt, b2Body *body) {
    this→timeElapsed += dt;
    if (this→timeElapsed > this→landTime) {
        p.notify(p, Event::WormLanded);
        if (p.health ≤ 0) {
            p.notify(p, Event::Dying);
            p.setState(Worm::StateID::Die);
        } else {
            p.setState(Worm::StateID::Still);
        }
    }
}

void Worms::Land::moveRight(Worms::Player &p) {}

void Worms::Land::moveLeft(Worms::Player &p) {}

void Worms::Land::jump(Worms::Player &p) {}

void Worms::Land::stopMove(Worms::Player &p) {}

void Worms::Land::backFlip(Worms::Player &p) {}

void Worms::Land::bazooka(Worms::Player &p) {}

void Worms::Land::pointUp(Worms::Player &p) {}

void Worms::Land::pointDown(Worms::Player &p) {}

void Worms::Land::startShot(Worms::Player &p) {}

void Worms::Land::endShot(Worms::Player &p) {}

void Worms::Land::grenade(Worms::Player &p) {}

void Worms::Land::cluster(Worms::Player &p) {}

void Worms::Land::mortar(Worms::Player &p) {}

void Worms::Land::banana(Worms::Player &p) {}

void Worms::Land::holy(Worms::Player &p) {}

void Worms::Land::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::Land::aerialAttack(Worms::Player &p) {}

void Worms::Land::dynamite(Worms::Player &p) {}

void Worms::Land::teleport(Worms::Player &p) {}

void Worms::Land::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **Jumping.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 20/05/18
 */

#ifndef __PLAYER_JUMPING_H__
#define __PLAYER_JUMPING_H__

#include <Box2D/Dynamics/b2Body.h>
#include <Camera.h>

#include "PlayerState.h"

namespace Worms {
class Jumping : public State {
   public:
    Jumping(GUI::Position p);
    ~Jumping() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    virtual void pointUp(Player &p) override;
    virtual void pointDown(Player &p) override;

   private:
    float timeElapsed{0.0f};
    GUI::Position startPosition;
};
}  // namespace Worms

#endif  //__PLAYER_JUMPING_H__
```

jun 26, 18 17:16 **Jumping.cpp** Page 1/2

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 20/05/18
 */

#include <Box2D/Dynamics/b2Body.h>
#include <iostream>
#include <vector>

#include "../Player.h"
#include "Jumping.h"

Worms::Jumping::Jumping(GUI::Position p) : State(Worm::StateID::Jumping), startPosition(p) {}

void Worms::Jumping::update(Worms::Player &p, float dt, b2Body *body) {
    /*
     * when the worm lands (there was a collision between the worm and the
     * girder) it has to changes its state to endJump, and take an impulse
     * of equal absolute value and different sign of the impulse taken in
     * startJump stage (remember, the worm has a friction coefficient 0).
     *
     * In the y-axis there will be no impulse because its velocity was
     * cancelled because of the collision with the girder.
     */
    if (p.isOnGround()) {
        this->timeElapsed += dt;
    } else {
        this->timeElapsed = 0.0f;
    }
    if (p.isOnGround() ∨ this->timeElapsed > 0.2f) {
        float32 mass = body->GetMass();
        b2Vec2 previousVel = body->GetLinearVelocity();
        b2Vec2 impulses = {mass * (0.0f - previousVel.x), 0.0f};
        body->ApplyLinearImpulseToCenter(impulses, true);

        p.landDamage(this->startPosition.y - p.getPosition().y);
        p.setState(Worm::StateID::Land);
        //        p.setState(Worm::StateID::EndJump);
    }
}

void Worms::Jumping::moveRight(Worms::Player &p) {}

void Worms::Jumping::moveLeft(Worms::Player &p) {}

void Worms::Jumping::jump(Worms::Player &p) {}

void Worms::Jumping::stopMove(Worms::Player &p) {}

void Worms::Jumping::backFlip(Worms::Player &p) {}

void Worms::Jumping::bazooka(Worms::Player &p) {}

void Worms::Jumping::pointUp(Worms::Player &p) {}

void Worms::Jumping::pointDown(Worms::Player &p) {}

void Worms::Jumping::startShot(Worms::Player &p) {}

void Worms::Jumping::endShot(Worms::Player &p) {}

void Worms::Jumping::grenade(Worms::Player &p) {}

void Worms::Jumping::cluster(Worms::Player &p) {}

```

jun 26, 18 17:16 **Jumping.cpp** Page 2/2

```cpp
void Worms::Jumping::mortar(Worms::Player &p) {}

void Worms::Jumping::banana(Worms::Player &p) {}

void Worms::Jumping::holy(Worms::Player &p) {}

void Worms::Jumping::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::Jumping::aerialAttack(Worms::Player &p) {}

void Worms::Jumping::dynamite(Worms::Player &p) {}

void Worms::Jumping::teleport(Worms::Player &p) {}

void Worms::Jumping::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **Hit.h** Page 1/1

```
/*
 *  Created by Rodrigo.
 *  date: 28/05/18
 */

#ifndef __Hit_H__
#define __Hit_H__

#include <cstdint>
#include "PlayerState.h"

namespace Worms {
class Hit : public State {
   public:
    Hit();
    ~Hit() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void pointUp(Player &p) override;
    void pointDown(Player &p) override;

   private:
    float timeElapsed{0.0f};
};
}  // namespace Worms

#endif  //__Hit_H__
```

jun 26, 18 17:16 **Hit.cpp** Page 1/2

```
/*
 *  Created by Rodrigo.
 *  date: 28/05/18
 */

#include "Hit.h"
#include "../Player.h"

Worms::Hit::Hit() : State(Worm::StateID::Hit) {}

void Worms::Hit::update(Worms::Player &p, float dt, b2Body *body) {
    /*
     * when the worm lands (there was a collision between the worm and the
     * girder) it has to change its state to still, and take an impulse
     * of equal absolute value and different sign of the impulse taken in
     * hit stage (remember, the worm has a friction coefficient 0).
     *
     * In the y-axis there will be no impulse because its velocity was
     * cancelled because of the collision with the girder.
     */
    if (p.isOnGround()) {
        this→timeElapsed += dt;
        if (this→timeElapsed > 0.7f) {
            float32 mass = body→GetMass();
            b2Vec2 previousVel = body→GetLinearVelocity();
            b2Vec2 impulses = {mass * (0.0f - previousVel.x), 0.0f};
            body→ApplyLinearImpulseToCenter(impulses, true);

            p.notify(p, Event::EndHit);
            p.setState(Worm::StateID::Land);
        }
    } else {
        this→timeElapsed = 0.0f;
    }
}

void Worms::Hit::moveRight(Worms::Player &p) {}

void Worms::Hit::moveLeft(Worms::Player &p) {}

void Worms::Hit::jump(Worms::Player &p) {}

void Worms::Hit::stopMove(Worms::Player &p) {}

void Worms::Hit::backFlip(Worms::Player &p) {}

void Worms::Hit::bazooka(Worms::Player &p) {}

void Worms::Hit::pointUp(Worms::Player &p) {}

void Worms::Hit::pointDown(Worms::Player &p) {}

void Worms::Hit::startShot(Worms::Player &p) {}

void Worms::Hit::endShot(Worms::Player &p) {}

void Worms::Hit::grenade(Worms::Player &p) {}

void Worms::Hit::cluster(Worms::Player &p) {}

void Worms::Hit::mortar(Worms::Player &p) {}

void Worms::Hit::banana(Worms::Player &p) {}

void Worms::Hit::holy(Worms::Player &p) {}

```

jun 26, 18 17:16 **Hit.cpp** Page 2/2

```
67 void Worms::Hit::setTimeout(Worms::Player &p, uint8_t time) {}
68
69 void Worms::Hit::aerialAttack(Worms::Player &p) {}
70
71 void Worms::Hit::dynamite(Worms::Player &p) {}
72
73 void Worms::Hit::teleport(Worms::Player &p) {}
74
75 void Worms::Hit::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **Falling.h** Page 1/1

```
1  //
2  // Created by rodrigo on 3/06/18.
3  //
4
5  #ifndef INC_4_WORMS_FALLING_H
6  #define INC_4_WORMS_FALLING_H
7
8  #include <Camera.h>
9  #include <cstdint>
10 #include "../Player.h"
11
12 namespace Worms {
13 class Falling : public State {
14    public:
15     Falling(GUI::Position p);
16     ~Falling() = default;
17     void update(Player &p, float dt, b2Body *body) override;
18     void moveRight(Player &p) override;
19     void moveLeft(Player &p) override;
20     void jump(Player &p) override;
21     void backFlip(Player &p) override;
22     void stopMove(Player &p) override;
23     void setTimeout(Player &p, uint8_t time) override;
24     void bazooka(Player &p) override;
25     void grenade(Player &p) override;
26     void cluster(Player &p) override;
27     void mortar(Player &p) override;
28     void banana(Player &p) override;
29     void holy(Player &p) override;
30     void aerialAttack(Player &p) override;
31     void dynamite(Player &p) override;
32     void baseballBat(Player &p) override;
33     void teleport(Player &p) override;
34
35     void startShot(Player &p) override;
36     void endShot(Player &p) override;
37     void pointUp(Player &p) override;
38     void pointDown(Player &p) override;
39
40    private:
41     GUI::Position startPosition;
42 };
43 }  // namespace Worms
44
45 #endif  // INC_4_WORMS_FALLING_H
```

jun 26, 18 17:16 **Falling.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 3/06/18.
//

#include "Falling.h"

Worms::Falling::Falling(GUI::Position p) : State(Worm::StateID::Falling), startPosition(p) {}

void Worms::Falling::update(Player &p, float dt, b2Body *body) {
    if (p.isOnGround()) {
        p.landDamage(this→startPosition.y − p.getPosition().y);
        p.setState(Worm::StateID::Land);
    }
}

void Worms::Falling::moveRight(Worms::Player &p) {}

void Worms::Falling::moveLeft(Worms::Player &p) {}

void Worms::Falling::jump(Worms::Player &p) {}

void Worms::Falling::stopMove(Worms::Player &p) {}

void Worms::Falling::backFlip(Worms::Player &p) {}

void Worms::Falling::bazooka(Worms::Player &p) {}

void Worms::Falling::pointUp(Worms::Player &p) {}

void Worms::Falling::pointDown(Worms::Player &p) {}

void Worms::Falling::startShot(Worms::Player &p) {}

void Worms::Falling::endShot(Worms::Player &p) {}

void Worms::Falling::grenade(Worms::Player &p) {}

void Worms::Falling::cluster(Worms::Player &p) {}

void Worms::Falling::mortar(Worms::Player &p) {}

void Worms::Falling::banana(Worms::Player &p) {}

void Worms::Falling::holy(Worms::Player &p) {}

void Worms::Falling::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::Falling::aerialAttack(Worms::Player &p) {}

void Worms::Falling::dynamite(Worms::Player &p) {}

void Worms::Falling::teleport(Worms::Player &p) {}

void Worms::Falling::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **EndJump.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 20/05/18
 */

#ifndef __PLAYER_END_JUMP_H__
#define __PLAYER_END_JUMP_H__

#include "PlayerState.h"

namespace Worms {
class EndJump : public State {
   public:
    EndJump();
    ~EndJump() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    virtual void pointUp(Player &p) override;
    virtual void pointDown(Player &p) override;

   private:
    float timeElapsed{0.0f};
    const float landTime;
};
}  // namespace Worms

#endif  //__PLAYER_END_JUMP_H__
```

jun 26, 18 17:16 **EndJump.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 20/05/18
 */

#include "EndJump.h"
#include "../Config/Config.h"
#include "../Player.h"

Worms::EndJump::EndJump()
    : State(Worm::StateID::EndJump), landTime(Game::Config::getInstance().getLandTime()) {}

void Worms::EndJump::update(Worms::Player &p, float dt, b2Body *body) {
    this→timeElapsed += dt;
    if (this→timeElapsed > this→landTime) {
        p.setState(Worm::StateID::Still);
    }
}

void Worms::EndJump::moveRight(Worms::Player &p) {}

void Worms::EndJump::moveLeft(Worms::Player &p) {}

void Worms::EndJump::jump(Worms::Player &p) {}

void Worms::EndJump::stopMove(Worms::Player &p) {}

void Worms::EndJump::bazooka(Worms::Player &p) {}

void Worms::EndJump::pointUp(Worms::Player &p) {}

void Worms::EndJump::pointDown(Worms::Player &p) {}

void Worms::EndJump::backFlip(Worms::Player &p) {}

void Worms::EndJump::startShot(Worms::Player &p) {}

void Worms::EndJump::endShot(Worms::Player &p) {}

void Worms::EndJump::grenade(Worms::Player &p) {}

void Worms::EndJump::cluster(Worms::Player &p) {}

void Worms::EndJump::mortar(Worms::Player &p) {}

void Worms::EndJump::banana(Worms::Player &p) {}

void Worms::EndJump::holy(Worms::Player &p) {}

void Worms::EndJump::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::EndJump::aerialAttack(Worms::Player &p) {}

void Worms::EndJump::dynamite(Worms::Player &p) {}

void Worms::EndJump::teleport(Worms::Player &p) {}

void Worms::EndJump::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **EndBackFlip.h** Page 1/1

```cpp
/*
 *  Created by Rodrigo.
 *  date: 21/05/18
 */

#ifndef __PLAYER_END_BACKFLIP_H__
#define __PLAYER_END_BACKFLIP_H__

#include <cstdint>
#include "PlayerState.h"

namespace Worms {
class EndBackFlip : public State {
   public:
    EndBackFlip();
    ~EndBackFlip() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void pointUp(Player &p) override;
    void pointDown(Player &p) override;

   private:
    float timeElapsed{0.0f};
    float landTime;
};
}  // namespace Worms

#endif  //__PLAYER_END_BACKFLIP_H__
```

jun 26, 18 17:16 **EndBackFlip.cpp** Page 1/1

```cpp
/*
 *  Created by Rodrigo.
 *  date: 21/05/18
 */

#include "EndBackFlip.h"
#include "../Config/Config.h"
#include "../Player.h"
#include "PlayerState.h"

Worms::EndBackFlip::EndBackFlip()
    : State(Worm::StateID::EndBackFlip), landTime(Game::Config::getInstance().getLandTime()) {}

void Worms::EndBackFlip::update(Worms::Player &p, float dt, b2Body *body) {
    this→timeElapsed += dt;
    if (this→timeElapsed > this→landTime) {
        p.setState(Worm::StateID::Still);
    }
}

void Worms::EndBackFlip::moveRight(Worms::Player &p) {}

void Worms::EndBackFlip::moveLeft(Worms::Player &p) {}

void Worms::EndBackFlip::jump(Worms::Player &p) {}

void Worms::EndBackFlip::stopMove(Worms::Player &p) {}

void Worms::EndBackFlip::backFlip(Worms::Player &p) {}

void Worms::EndBackFlip::bazooka(Worms::Player &p) {}

void Worms::EndBackFlip::pointUp(Worms::Player &p) {}

void Worms::EndBackFlip::pointDown(Worms::Player &p) {}

void Worms::EndBackFlip::startShot(Worms::Player &p) {}

void Worms::EndBackFlip::endShot(Worms::Player &p) {}

void Worms::EndBackFlip::grenade(Worms::Player &p) {}

void Worms::EndBackFlip::cluster(Worms::Player &p) {}

void Worms::EndBackFlip::mortar(Worms::Player &p) {}

void Worms::EndBackFlip::banana(Worms::Player &p) {}

void Worms::EndBackFlip::holy(Worms::Player &p) {}

void Worms::EndBackFlip::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::EndBackFlip::aerialAttack(Worms::Player &p) {}

void Worms::EndBackFlip::dynamite(Worms::Player &p) {}

void Worms::EndBackFlip::teleport(Worms::Player &p) {}

void Worms::EndBackFlip::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **Drowning.h** Page 1/1

```cpp
/*
 *  Created by Rodrigo.
 *  date: 29/05/18
 */

#ifndef __Drown_H__
#define __Drown_H__

#include <cstdint>

#include "../Config/Config.h"
#include "PlayerState.h"

namespace Worms {
class Drowning : public State {
   public:
    Drowning();
    ~Drowning() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void pointUp(Player &p) override;
    void pointDown(Player &p) override;

    float timeElapsed{0.0f};
    float drowningTime;
};
}  // namespace Worms

#endif  //__Drown_H__
```

jun 26, 18 17:16 **Drowning.cpp** Page 1/1

```cpp
/*
 *  Created by Rodrigo.
 *  date: 29/05/18
 */

#include "Drowning.h"
#include "../Player.h"

Worms::Drowning::Drowning()
    : State(Worm::StateID::Drowning), drowningTime(Game::Config::getInstance().getDrowningTime()) {}

void Worms::Drowning::update(Worms::Player &p, float dt, b2Body *body) {
    this→timeElapsed += dt;
    if (this→timeElapsed ≥ this→drowningTime) {
        p.setState(Worm::StateID::Dead);
        p.notify(p, Event::Drowned);
    }
}

void Worms::Drowning::moveRight(Worms::Player &p) {}

void Worms::Drowning::moveLeft(Worms::Player &p) {}

void Worms::Drowning::jump(Worms::Player &p) {}

void Worms::Drowning::stopMove(Worms::Player &p) {}

void Worms::Drowning::backFlip(Worms::Player &p) {}

void Worms::Drowning::bazooka(Worms::Player &p) {}

void Worms::Drowning::pointUp(Worms::Player &p) {}

void Worms::Drowning::pointDown(Worms::Player &p) {}

void Worms::Drowning::startShot(Worms::Player &p) {}

void Worms::Drowning::endShot(Worms::Player &p) {}

void Worms::Drowning::grenade(Worms::Player &p) {}

void Worms::Drowning::cluster(Worms::Player &p) {}

void Worms::Drowning::mortar(Worms::Player &p) {}

void Worms::Drowning::banana(Worms::Player &p) {}

void Worms::Drowning::holy(Worms::Player &p) {}

void Worms::Drowning::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::Drowning::aerialAttack(Worms::Player &p) {}

void Worms::Drowning::dynamite(Worms::Player &p) {}

void Worms::Drowning::teleport(Worms::Player &p) {}

void Worms::Drowning::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **Die.h** Page 1/1

```cpp
/*
 *  Created by Rodrigo.
 *  date: 28/05/18
 */

#ifndef __DIE_H__
#define __DIE_H__

#include <cstdint>
#include "../Config/Config.h"
#include "PlayerState.h"

namespace Worms {
class Die : public State {
   public:
    Die();
    ~Die() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void pointUp(Player &p) override;
    void pointDown(Player &p) override;

   private:
    float timeElapsed{0.0f};
    float dyingTime{Game::Config::getInstance().getDyingTime()};
};
}  // namespace Worms

#endif  //__DIE_H__
```

jun 26, 18 17:16 **Die.cpp** Page 1/1

```cpp
/*
 *  Created by Rodrigo.
 *  date: 28/05/18
 */

#include "Die.h"
#include "../Player.h"

Worms::Die::Die() : State(Worm::StateID::Die) {}

void Worms::Die::update(Worms::Player &p, float dt, b2Body *body) {
    this→timeElapsed += dt;
    if (this→timeElapsed ≥ this→dyingTime) {
        if (p.dyingDisconnected) {
            p.notify(p, Event::DeadDueToDisconnection);
        } else {
            p.notify(p, Event::Dead);
        }
        p.setState(Worm::StateID::Dead);
    }
}

void Worms::Die::moveRight(Worms::Player &p) {}

void Worms::Die::moveLeft(Worms::Player &p) {}

void Worms::Die::jump(Worms::Player &p) {}

void Worms::Die::stopMove(Worms::Player &p) {}

void Worms::Die::backFlip(Worms::Player &p) {}

void Worms::Die::bazooka(Worms::Player &p) {}

void Worms::Die::pointUp(Worms::Player &p) {}

void Worms::Die::pointDown(Worms::Player &p) {}

void Worms::Die::startShot(Worms::Player &p) {}

void Worms::Die::endShot(Worms::Player &p) {}

void Worms::Die::grenade(Worms::Player &p) {}

void Worms::Die::cluster(Worms::Player &p) {}

void Worms::Die::mortar(Worms::Player &p) {}

void Worms::Die::banana(Worms::Player &p) {}

void Worms::Die::holy(Worms::Player &p) {}

void Worms::Die::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::Die::aerialAttack(Worms::Player &p) {}

void Worms::Die::dynamite(Worms::Player &p) {}

void Worms::Die::teleport(Worms::Player &p) {}

void Worms::Die::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **Dead.h** Page 1/1

```cpp
/*
 *  Created by Rodrigo.
 *  date: 28/05/18
 */

#ifndef __Dead_H__
#define __Dead_H__

#include <cstdint>
#include "PlayerState.h"

namespace Worms {
class Dead : public State {
   public:
    Dead();
    ~Dead() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void pointUp(Player &p) override;
    void pointDown(Player &p) override;
};
}  // namespace Worms

#endif  //__Dead_H__
```

jun 26, 18 17:16 **Dead.cpp** Page 1/1

```cpp
/*
 *  Created by Rodrigo.
 *  date: 28/05/18
 */

#include "Dead.h"
#include "../Player.h"

Worms::Dead::Dead() : State(Worm::StateID::Dead) {}

void Worms::Dead::update(Worms::Player &p, float dt, b2Body *body) {}

void Worms::Dead::moveRight(Worms::Player &p) {}

void Worms::Dead::moveLeft(Worms::Player &p) {}

void Worms::Dead::jump(Worms::Player &p) {}

void Worms::Dead::stopMove(Worms::Player &p) {}

void Worms::Dead::backFlip(Worms::Player &p) {}

void Worms::Dead::bazooka(Worms::Player &p) {}

void Worms::Dead::pointUp(Worms::Player &p) {}

void Worms::Dead::pointDown(Worms::Player &p) {}

void Worms::Dead::startShot(Worms::Player &p) {}

void Worms::Dead::endShot(Worms::Player &p) {}

void Worms::Dead::grenade(Worms::Player &p) {}

void Worms::Dead::cluster(Worms::Player &p) {}

void Worms::Dead::mortar(Worms::Player &p) {}

void Worms::Dead::banana(Worms::Player &p) {}

void Worms::Dead::holy(Worms::Player &p) {}

void Worms::Dead::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::Dead::aerialAttack(Worms::Player &p) {}

void Worms::Dead::dynamite(Worms::Player &p) {}

void Worms::Dead::teleport(Worms::Player &p) {}

void Worms::Dead::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **Batting.h** Page 1/1

```cpp
//
// Created by rodrigo on 23/06/18.
//

#ifndef INC_4_WORMS_BATTING_H
#define INC_4_WORMS_BATTING_H

#include "PlayerState.h"

namespace Worms {
class Batting : public State {
   public:
    Batting();
    ~Batting() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    void pointUp(Player &p) override;
    void pointDown(Player &p) override;

   private:
    float timeElapsed{0.0f};
    float battingTime;
};
}

#endif  // INC_4_WORMS_BATTING_H
```

jun 26, 18 17:16 **Batting.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 23/06/18.
//

#include "Batting.h"
#include "../Config/Config.h"
#include "../Player.h"

Worms::Batting::Batting()
    : State(Worm::StateID::Batting), battingTime(Game::Config::getInstance().get
BattingTime()) {}

void Worms::Batting::update(Worms::Player &p, float dt, b2Body *body) {
    this→timeElapsed += dt;
    if (this→timeElapsed ≥ this→battingTime) {
        p.setState(Worm::StateID::Still);
    }
}

void Worms::Batting::moveRight(Worms::Player &p) {}

void Worms::Batting::moveLeft(Worms::Player &p) {}

void Worms::Batting::jump(Worms::Player &p) {}

void Worms::Batting::stopMove(Worms::Player &p) {}

void Worms::Batting::backFlip(Worms::Player &p) {}

void Worms::Batting::bazooka(Worms::Player &p) {}

void Worms::Batting::pointUp(Worms::Player &p) {}

void Worms::Batting::pointDown(Worms::Player &p) {}

void Worms::Batting::startShot(Worms::Player &p) {}

void Worms::Batting::endShot(Worms::Player &p) {}

void Worms::Batting::grenade(Worms::Player &p) {}

void Worms::Batting::cluster(Worms::Player &p) {}

void Worms::Batting::mortar(Worms::Player &p) {}

void Worms::Batting::banana(Worms::Player &p) {}

void Worms::Batting::holy(Worms::Player &p) {}

void Worms::Batting::setTimeout(Worms::Player &p, uint8_t time) {}

void Worms::Batting::aerialAttack(Worms::Player &p) {}

void Worms::Batting::dynamite(Worms::Player &p) {}

void Worms::Batting::teleport(Worms::Player &p) {}

void Worms::Batting::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **BackFlipping.h** Page 1/1

```cpp
/*
 *  Created by Rodrigo.
 *  date: 21/05/18
 */

#ifndef __PLAYER_BACK_FLIPPING_H__
#define __PLAYER_BACK_FLIPPING_H__

#include <Camera.h>
#include <cstdint>
#include "PlayerState.h"

namespace Worms {
class BackFlipping : public State {
   public:
    BackFlipping(GUI::Position p);
    ~BackFlipping() = default;
    void update(Player &p, float dt, b2Body *body) override;
    void moveRight(Player &p) override;
    void moveLeft(Player &p) override;
    void jump(Player &p) override;
    void backFlip(Player &p) override;
    void stopMove(Player &p) override;
    void setTimeout(Player &p, uint8_t time) override;

    void bazooka(Player &p) override;
    void grenade(Player &p) override;
    void cluster(Player &p) override;
    void mortar(Player &p) override;
    void banana(Player &p) override;
    void holy(Player &p) override;
    void aerialAttack(Player &p) override;
    void dynamite(Player &p) override;
    void baseballBat(Player &p) override;
    void teleport(Player &p) override;

    void startShot(Player &p) override;
    void endShot(Player &p) override;
    void pointUp(Player &p) override;
    void pointDown(Player &p) override;

   private:
    float timeElapsed{0.0f};
    GUI::Position startPosition;
};
}  // namespace Worms

#endif  //__PLAYER_BACK_FLIPPING_H__
```

jun 26, 18 17:16 **BackFlipping.cpp** Page 1/2

```
1  /*
2   *  Created by Rodrigo.
3   *  date: 21/05/18
4   */
5
6  #include "BackFlipping.h"
7  #include "../Player.h"
8
9  Worms::BackFlipping::BackFlipping(GUI::Position p)
10     : State(Worm::StateID::BackFlipping), startPosition(p) {}
11
12 void Worms::BackFlipping::update(Worms::Player &p, float dt, b2Body *body) {
13     /*
14      * when the worm lands (there was a collision between the worm and the
15      * girder) it has to changes its state to endJump, and take an impulse
16      * of equal absolute value and different sign of the impulse taken in
17      * startJump stage (remember, the worm has a friction coefficient 0).
18      *
19      * In the y-axis there will be no impulse because its velocity was
20      * cancelled because of the collision with the girder.
21      */
22     this→timeElapsed += dt;
23
24     if (p.isOnGround()) {
25         float32 mass = body→GetMass();
26         b2Vec2 previousVel = body→GetLinearVelocity();
27         b2Vec2 impulses = {mass * (0.0f - previousVel.x), 0.0f};
28         body→ApplyLinearImpulseToCenter(impulses, true);
29
30         p.landDamage(this→startPosition.y - p.getPosition().y);
31         p.setState(Worm::StateID::Land);
32         //        p.setState(Worm::StateID::EndBackFlip);
33     }
34 }
35
36 void Worms::BackFlipping::moveRight(Worms::Player &p) {}
37
38 void Worms::BackFlipping::moveLeft(Worms::Player &p) {}
39
40 void Worms::BackFlipping::jump(Worms::Player &p) {}
41
42 void Worms::BackFlipping::stopMove(Worms::Player &p) {}
43
44 void Worms::BackFlipping::backFlip(Worms::Player &p) {}
45
46 void Worms::BackFlipping::bazooka(Worms::Player &p) {}
47
48 void Worms::BackFlipping::pointUp(Worms::Player &p) {}
49
50 void Worms::BackFlipping::pointDown(Worms::Player &p) {}
51
52 void Worms::BackFlipping::startShot(Worms::Player &p) {}
53
54 void Worms::BackFlipping::endShot(Worms::Player &p) {}
55
56 void Worms::BackFlipping::grenade(Worms::Player &p) {}
57
58 void Worms::BackFlipping::cluster(Worms::Player &p) {}
59
60 void Worms::BackFlipping::mortar(Worms::Player &p) {}
61
62 void Worms::BackFlipping::banana(Worms::Player &p) {}
63
64 void Worms::BackFlipping::holy(Worms::Player &p) {}
65
66 void Worms::BackFlipping::setTimeout(Worms::Player &p, uint8_t time) {}
```

jun 26, 18 17:16 **BackFlipping.cpp** Page 2/2

```
67
68 void Worms::BackFlipping::aerialAttack(Worms::Player &p) {}
69
70 void Worms::BackFlipping::dynamite(Worms::Player &p) {}
71
72 void Worms::BackFlipping::teleport(Worms::Player &p) {}
73
74 void Worms::BackFlipping::baseballBat(Worms::Player &p) {}
```

jun 26, 18 17:16 **WeaponNone.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 24/06/18
 */

#ifndef __WEAPON_NONE_H__
#define __WEAPON_NONE_H__

#include "Weapon.h"

namespace Weapon {
class WeaponNone : public Worms::Weapon {
   public:
    WeaponNone();
    ~WeaponNone() override = default;
    void update(float dt) override{};
    void increaseAngle() override{};
    void decreaseAngle() override{};
    void checkBoundaryAngles() override{};
    void startShot(Worms::Player *player) override{};
    void endShot() override{};
    void setTimeout(uint8_t time) override{};
    std::list<Worms::Bullet> onExplode(const Worms::Bullet &mainBullet,
                                       Worms::Physics &physics) override;
    void positionSelected(Worms::Player &p, Math::Point<float> point) override{}
;
};
}  // namespace Weapon

#endif  //__WEAPON_NONE_H__
```

jun 26, 18 17:16 **WeaponNone.cpp** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 24/06/18
 */

#include "WeaponNone.h"

#define CONFIG Game::Config::getInstance()

Weapon::WeaponNone::WeaponNone()
    : Weapon::Weapon(CONFIG.getTeleportConfig(), Worm::WeaponID::WNone, 0.0) {}

std::list<Worms::Bullet> Weapon::WeaponNone::onExplode(const Worms::Bullet &main
Bullet,
                                                       Worms::Physics &physics)
{
    return std::move(std::list<Worms::Bullet>());
}
```

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 28/05/18
 */

#ifndef __WEAPON_H__
#define __WEAPON_H__

#include <GameStateMsg.h>
#include <list>
#include <memory>

#include "../Config/Config.h"
#include "Bullet.h"
#include "../Config/WeaponConfig.h"

namespace Worms {
class Player;
class Weapon {
   public:
    Weapon(const Config::Weapon &config, Worm::WeaponID id, float angle);
    virtual ~Weapon() = default;

    const Worm::WeaponID &getWeaponID() const;
    /**
     * If was an event of startShot, then increase its power shot until
     * reach its limit.
     * @param dt
     */
    virtual void update(float dt) = 0;
    /**
     * @brief increases the angle of the aim. If the angle exceeds the limit
     * then it will be changed to the maximum possible
     */
    virtual void increaseAngle();
    /**
     * @brief decreases the angle of the aim. If the angle exceeds the limit
     * then it will be changed to the maximum possible
     */
    virtual void decreaseAngle();
    float getAngle() const;
    void setAngle(float angle);
    virtual void startShot(Worms::Player *player) = 0;
    virtual void endShot() = 0;
    BulletInfo getBulletInfo();
    virtual void setTimeout(uint8_t time) = 0;
    /**
     * @brief check if the weapon is person to preson or not
     * @return
     */
    bool isP2PWeapon();
    /**
     * Used by te remote control weapons. Sends to the weapon the coordinates
     * of the deploy of the bullets, and a reference of Player so that the
     * weapons, if they are remote control. calls the appropiate method.
     * @param player to call deploy method (if the weapon has this feature)
     * @param point
     */
    virtual void positionSelected(Worms::Player &p, Math::Point<float> point) =
0;
    /**
     * Function that returns, using move semantics, a list of bullets
     * depending on weapon's behavior after the main bullet explode.
     * @return
     */
    virtual std::list<Worms::Bullet> onExplode(const Worms::Bullet &mainBullet,
```



```
                                               Worms::Physics &physics) = 0;

    protected:
     bool increaseShotPower{false};
     float shotPower{0};
     bool isP2P{false};
     const Config::Weapon &config;
     Worm::WeaponID id;
     float angle{0};
     uint8_t timeLimit;

    private:
     /**
      * When weapons change, their own limit angles may crash the game.
      * To avoid this, this function checks and correct angles between changes.
      */
     virtual void checkBoundaryAngles();
};
}  // namespace Worms

#endif  //_WEAPON_H__
```

jun 26, 18 17:16 **Weapon.cpp** Page 1/2

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 28/05/18
 */

#include "Weapon.h"
#include "../Config/Config.h"
#include "../Player.h"
#include "../Config/WeaponConfig.h"

Worms::Weapon::Weapon(const Config::Weapon &config, Worm::WeaponID id, float angle)
    : config(config), id(id), angle(angle) {
    this→angle = angle;
    this→timeLimit = this→config.explotionInitialTimeout;
    /*
     * Because the limit angles between weapons are
     * differents, it is necesary to check boundaries angles.
     * If not, the game could crash in rendering time.
     */
    this→checkBoundaryAngles();
}

const Worm::WeaponID &Worms::Weapon::getWeaponID() const {
    return this→id;
}

void Worms::Weapon::decreaseAngle() {
    this→angle −= this→config.angleStep;
    if (this→angle < this→config.minAngle) {
        this→angle = this→config.minAngle;
    }
}

void Worms::Weapon::increaseAngle() {
    this→angle += this→config.angleStep;
    if (this→angle > this→config.maxAngle) {
        this→angle = this→config.maxAngle;
    }
}

float Worms::Weapon::getAngle() const {
    return this→angle;
}

void Worms::Weapon::checkBoundaryAngles() {
    if (this→angle > this→config.maxAngle) {
        this→angle = this→config.maxAngle;
    } else if (this→angle < this→config.minAngle) {
        this→angle = this→config.minAngle;
    }
}

Worms::BulletInfo Worms::Weapon::getBulletInfo() {
    return Worms::BulletInfo{this→config.dmgInfo,
                             Math::Point<float>{0, 0},
                             angle,
                             this→shotPower,
                             0,
                             this→config.restitution,
                             this→config.friction,
                             this→timeLimit,
                             this→config.hasAfterExplode ? Event::OnExplode : Event::Explode,
                             this→config.bulletRadius,
                             this→config.bulletDampingRatio,
```

jun 26, 18 17:16 **Weapon.cpp** Page 2/2

```cpp
                             this→config.windAffected};
}

void Worms::Weapon::setAngle(float angle) {
    this→angle = angle;
}

bool Worms::Weapon::isP2PWeapon() {
    return this→isP2P;
}
```

jun 26, 18 17:16 **Teleport.h** Page 1/1

```
1  //
2  // Created by rodrigo on 16/06/18.
3  //
4
5  #ifndef INC_4_WORMS_TELEPORT_H
6  #define INC_4_WORMS_TELEPORT_H
7
8  #include "../Player.h"
9  #include "Weapon.h"
10
11 namespace Weapon {
12 class Teleport : public Worms::Weapon {
13    public:
14     Teleport();
15     ~Teleport() override = default;
16     void update(float dt) override;
17     void increaseAngle() override;
18     void decreaseAngle() override;
19     void startShot(Worms::Player *player) override;
20     void endShot() override;
21     void setTimeout(uint8_t time) override;
22     std::list<Worms::Bullet> onExplode(const Worms::Bullet &mainBullet,
23                                        Worms::Physics &physics) override;
24     void positionSelected(Worms::Player &p, Math::Point<float> point) override;
25 };
26 }  // namespace Weapon
27
28 #endif  // INC_4_WORMS_TELEPORT_H
```

jun 26, 18 17:16 **Teleport.cpp** Page 1/1

```
1  //
2  // Created by rodrigo on 16/06/18.
3  //
4
5  #include "Teleport.h"
6
7  #define CONFIG Game::Config::getInstance()
8
9  Weapon::Teleport::Teleport()
10     : Weapon::Weapon(CONFIG.getTeleportConfig(), Worm::WeaponID::WTeleport, 0.0)
    {}
11
12 void Weapon::Teleport::update(float dt) {}
13
14 void Weapon::Teleport::startShot(Worms::Player *player) {}
15
16 void Weapon::Teleport::endShot() {}
17
18 void Weapon::Teleport::setTimeout(uint8_t time) {}
19
20 std::list<Worms::Bullet> Weapon::Teleport::onExplode(const Worms::Bullet &mainBu
   llet,
21                                                      Worms::Physics &physics) {
22     return std::move(std::list<Worms::Bullet>());
23 }
24
25 void Weapon::Teleport::positionSelected(Worms::Player &p, Math::Point<float> poi
   nt) {
26     p.teleportPosition = point;
27     p.notify(p, Event::Teleported);
28     p.setState(Worm::StateID::Teleporting);
29 }
30
31 void Weapon::Teleport::increaseAngle() {}
32
33 void Weapon::Teleport::decreaseAngle() {}
```

jun 26, 18 17:16 **Mortar.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#ifndef __Mortar_H__
#define __Mortar_H__

#include "Weapon.h"

namespace Weapon {
class Mortar : public Worms::Weapon {
   public:
    Mortar(float angle);
    ~Mortar() override = default;
    void update(float dt) override;
    void startShot(Worms::Player *player) override;
    void endShot() override;
    void setTimeout(uint8_t time) override;
    std::list<Worms::Bullet> onExplode(const Worms::Bullet &bullet,
                                       Worms::Physics &physics) override;
    void positionSelected(Worms::Player &p, Math::Point<float> point) override;

   private:
    const Config::Weapon &fragmentConfig;
};
}  // namespace Weapon

#endif  //__Mortar_H__
```

jun 26, 18 17:16 **Mortar.cpp** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#include "Mortar.h"
#include "../Player.h"

Weapon::Mortar::Mortar(float angle)
    : Worms::Weapon(Game::Config::getInstance().getMortarConfig(), Worm::WeaponID::WMortar, angle),
      fragmentConfig(Game::Config::getInstance().getMortarFragmentConfig()) {}

void Weapon::Mortar::update(float dt) {
    if (this→increaseShotPower) {
        if (this→shotPower ≥ this→config.maxShotPower) {
            this→shotPower = this→config.maxShotPower;
        } else {
            this→shotPower++;
        }
    }
}

void Weapon::Mortar::startShot(Worms::Player *player) {
    this→increaseShotPower = true;
}

void Weapon::Mortar::endShot() {
    this→increaseShotPower = false;
    this→shotPower = 0;
}

void Weapon::Mortar::setTimeout(uint8_t time) {}

std::list<Worms::Bullet> Weapon::Mortar::onExplode(const Worms::Bullet &mainBullet,
                                                   Worms::Physics &physics) {
    uint8_t fragmentQuantity = Game::Config::getInstance().getMortarFragmentQuantity();
    Math::Point<float> p = mainBullet.getPosition();
    Worms::BulletInfo bulletInfo = {this→fragmentConfig.dmgInfo,
                                    p,
                                    this→fragmentConfig.minAngle,
                                    (float)this→fragmentConfig.maxShotPower,
                                    this→fragmentConfig.bulletRadius * 6,
                                    this→fragmentConfig.restitution,
                                    this→fragmentConfig.friction,
                                    this→fragmentConfig.explotionInitialTimeout,
                                    Event::Explode,
                                    this→fragmentConfig.bulletRadius,
                                    this→fragmentConfig.bulletDampingRatio,
                                    this→config.windAffected};

    std::list<Worms::Bullet> ret;
    for (int i = 0; i < fragmentQuantity; i++) {
        bulletInfo.angle = i * this→fragmentConfig.angleStep + this→fragmentConfig.minAngle;
        ret.emplace_back(bulletInfo, physics, Worm::WeaponID::WFragment);
    }

    return std::move(ret);
}

void Weapon::Mortar::positionSelected(Worms::Player &p, Math::Point<float> point) {}
```

jun 26, 18 17:16 **Holy.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#ifndef __Holy_H__
#define __Holy_H__

#include "Weapon.h"

namespace Weapon {
class Holy : public Worms::Weapon {
   public:
    Holy(float angle);
    ~Holy() override = default;
    void update(float dt) override;
    void startShot(Worms::Player *player) override;
    void endShot() override;
    void setTimeout(uint8_t time) override;
    std::list<Worms::Bullet> onExplode(const Worms::Bullet &bullet,
                                       Worms::Physics &physics) override;
    void positionSelected(Worms::Player &p, Math::Point<float> point) override;

   private:
    float powerChargeTime{0.0f};
};
}  // namespace Weapon

#endif  //__Holy_H__
```

jun 26, 18 17:16 **Holy.cpp** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#include "Holy.h"
#include "../Player.h"

Weapon::Holy::Holy(float angle)
    : Worms::Weapon(Game::Config::getInstance().getHolyConfig(), Worm::WeaponID:
:WHoly, angle) {
    this→powerChargeTime = Game::Config::getInstance().getPowerChargeTime();
}

void Weapon::Holy::update(float dt) {
    if (this→increaseShotPower) {
        if (this→shotPower < this→config.maxShotPower) {
            this→shotPower += dt / this→powerChargeTime * this→config.maxShotP
ower;
        }
    }
}

void Weapon::Holy::startShot(Worms::Player *player) {
    this→increaseShotPower = true;
}

void Weapon::Holy::endShot() {
    this→increaseShotPower = false;
    this→shotPower = 0;
}

void Weapon::Holy::setTimeout(uint8_t time) {
    this→timeLimit = time;
}

std::list<Worms::Bullet> Weapon::Holy::onExplode(const Worms::Bullet &bullet,
                                                 Worms::Physics &physics) {
    return std::move(std::list<Worms::Bullet>());
}

void Weapon::Holy::positionSelected(Worms::Player &p, Math::Point<float> point)
{}
```

jun 26, 18 17:16 **Grenade.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#ifndef __GRENADE_H__
#define __GRENADE_H__

#include "Weapon.h"

namespace Weapon {
class Grenade : public Worms::Weapon {
   public:
    Grenade(float angle);
    ~Grenade() override = default;
    void update(float dt) override;
    void startShot(Worms::Player *player) override;
    void endShot() override;
    void setTimeout(uint8_t time) override;
    std::list<Worms::Bullet> onExplode(const Worms::Bullet &bullet,
                                       Worms::Physics &physics) override;
    void positionSelected(Worms::Player &p, Math::Point<float> point) override;

   private:
    float powerChargeTime{0.0f};
};
}  // namespace Weapon

#endif  //__GRENADE_H__
```

jun 26, 18 17:16 **Grenade.cpp** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#include "Grenade.h"
#include "../Player.h"

Weapon::Grenade::Grenade(float angle)
    : Worms::Weapon(Game::Config::getInstance().getGreenGrenadeConfig(), Worm::WeaponID::WGrenade,
                    angle) {
    this→powerChargeTime = Game::Config::getInstance().getPowerChargeTime();
}

void Weapon::Grenade::update(float dt) {
    if (this→increaseShotPower) {
        if (this→shotPower < this→config.maxShotPower) {
            this→shotPower += dt / this→powerChargeTime * this→config.maxShotPower;
        }
    }
}

void Weapon::Grenade::startShot(Worms::Player *player) {
    this→increaseShotPower = true;
}

void Weapon::Grenade::endShot() {
    this→increaseShotPower = false;
    this→shotPower = 0;
}

void Weapon::Grenade::setTimeout(uint8_t time) {
    this→timeLimit = time;
}

std::list<Worms::Bullet> Weapon::Grenade::onExplode(const Worms::Bullet &bullet,
                                                    Worms::Physics &physics) {
    return std::move(std::list<Worms::Bullet>());
}

void Weapon::Grenade::positionSelected(Worms::Player &p, Math::Point<float> point) {}
```

jun 26, 18 17:16 **Dynamite.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 16/06/18
 */

#ifndef __TNT_H__
#define __TNT_H__

#include "Weapon.h"

namespace Weapon {
class Dynamite : public Worms::Weapon {
   public:
    Dynamite();
    ~Dynamite() override = default;
    void update(float dt) override;
    void startShot(Worms::Player *player) override;
    void endShot() override;
    void setTimeout(uint8_t time) override;
    std::list<Worms::Bullet> onExplode(const Worms::Bullet &mainBullet,
                                       Worms::Physics &physics) override;
    void positionSelected(Worms::Player &p, Math::Point<float> point) override;
    void increaseAngle() override;
    void decreaseAngle() override;
};
}  // namespace Weapon

#endif  //__TNT_H__
```

jun 26, 18 17:16 **Dynamite.cpp** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 16/06/18
 */

#include "Dynamite.h"
#include "../Player.h"

#define CONFIG Game::Config::getInstance()

Weapon::Dynamite::Dynamite()
    : Worms::Weapon(CONFIG.getDynamiteConfig(), Worm::WeaponID::WDynamite, 0.0)
{}

void Weapon::Dynamite::update(float dt) {}

void Weapon::Dynamite::startShot(Worms::Player *player) {}

void Weapon::Dynamite::endShot() {}

void Weapon::Dynamite::setTimeout(uint8_t time) {
    this→timeLimit = time;
}

std::list<Worms::Bullet> Weapon::Dynamite::onExplode(const Worms::Bullet &mainBullet,
                                                     Worms::Physics &physics) {
    return std::list<Worms::Bullet>();
}

void Weapon::Dynamite::positionSelected(Worms::Player &p, Math::Point<float> point) {}

void Weapon::Dynamite::increaseAngle() {}

void Weapon::Dynamite::decreaseAngle() {}
```

jun 26, 18 17:16 **Cluster.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#ifndef __CLUSTER_H__
#define __CLUSTER_H__

#include "../Physics.h"
#include "../Player.h"
#include "Weapon.h"

namespace Weapon {
class Cluster : public Worms::Weapon {
   public:
    Cluster(float angle);
    ~Cluster() override = default;
    void update(float dt) override;
    void startShot(Worms::Player *player) override;
    void endShot() override;
    void setTimeout(uint8_t time) override;
    std::list<Worms::Bullet> onExplode(const Worms::Bullet &mainBullet,
                                       Worms::Physics &physics) override;
    void positionSelected(Worms::Player &p, Math::Point<float> point) override;

   private:
    const Config::Weapon &fragmentConfig;
    float powerChargeTime{0.0f};
};
}  // namespace Weapon

#endif  //__CLUSTER_H__
```

jun 26, 18 17:16 **Cluster.cpp** Page 1/2

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#include "Cluster.h"

#define CONFIG Game::Config::getInstance()

Weapon::Cluster::Cluster(float angle)
    : Worms::Weapon(CONFIG.getClusterConfig(), Worm::WeaponID::WCluster, angle),
      fragmentConfig(CONFIG.getClusterFragmentConfig()) {
    this→powerChargeTime = CONFIG.getPowerChargeTime();
}

void Weapon::Cluster::update(float dt) {
    if (this→increaseShotPower) {
        if (this→shotPower < this→config.maxShotPower) {
            this→shotPower += dt / this→powerChargeTime * this→config.maxShotPower;
        }
    }
}

void Weapon::Cluster::startShot(Worms::Player *player) {
    this→increaseShotPower = true;
}

void Weapon::Cluster::endShot() {
    this→increaseShotPower = false;
    this→shotPower = 0;
}

void Weapon::Cluster::setTimeout(uint8_t time) {
    this→timeLimit = time;
}

std::list<Worms::Bullet> Weapon::Cluster::onExplode(const Worms::Bullet &mainBullet,
                                                    Worms::Physics &physics) {
    uint8_t fragmentQuantity = CONFIG.getClusterFragmentQuantity();
    Math::Point<float> p = mainBullet.getPosition();
    Worms::BulletInfo bulletInfo = {this→fragmentConfig.dmgInfo,
                                    p,
                                    this→fragmentConfig.minAngle,
                                    (float)this→fragmentConfig.maxShotPower,
                                    this→fragmentConfig.bulletRadius * 6,
                                    this→fragmentConfig.restitution,
                                    this→fragmentConfig.friction,
                                    this→fragmentConfig.explotionInitialTimeout,
                                    Event::Explode,
                                    this→fragmentConfig.bulletRadius,
                                    this→fragmentConfig.bulletDampingRatio,
                                    this→config.windAffected};

    std::list<Worms::Bullet> ret;
    for (int i = 0; i < fragmentQuantity; i++) {
        bulletInfo.angle = i * this→fragmentConfig.angleStep + this→fragmentConfig.minAngle;
        ret.emplace_back(bulletInfo, physics, Worm::WeaponID::WFragment);
    }

    return std::move(ret);
}

```

jun 26, 18 17:16 **Cluster.cpp** Page 2/2

```
63  void Weapon::Cluster::positionSelected(Worms::Player &p, Math::Point<float> poin
    t) {}
```

jun 26, 18 17:16 **Bullet.h** Page 1/2

```
1   /*
2    *  Created by Federico Manuel Gomez Peter.
3    *  date: 26/05/18
4    */
5
6   #ifndef __BULLET_H__
7   #define __BULLET_H__
8
9   #include <GameStateMsg.h>
10
11  #include "../Config/Config.h"
12  #include "../Config/WindConfig.h"
13  #include "../../../libs/Observer.h"
14  #include "../Physics.h"
15  #include "../PhysicsEntity.h"
16  #include "Point.h"
17
18  namespace Worms {
19  struct BulletInfo {
20      Config::Bullet::DamageInfo dmgInfo;
21      Math::Point<float> point;
22      float angle;
23      float power;
24      float safeNonContactDistance;
25      float restitution;
26      float friction;
27      uint8_t explotionTimeout;
28      Event explodeEvent;
29      float radius;
30      float dampingRatio;
31      bool windAffected;
32  };
33  /**
34   * forward declaration of weapon.
35   */
36  class Weapon;
37  class Bullet : public PhysicsEntity {
38     public:
39      Bullet(BulletInfo &i, Worms::Physics &physics, Worm::WeaponID weaponID);
40      ~Bullet();
41      /**
42       * Apply initial impulse in the first iteration, or estimate the
43       * bullet's tangential velocity to guide the animation. Finally, checks if
44       * an Explode event ocurred, and notify his observer if so.
45       * @param dt
46       * @param w
47       */
48      void update(float dt, Config::Wind wind);
49      Math::Point<float> getPosition() const;
50      float getAngle() const;
51      /**
52       * Sets its impact boolean to true. Usefull for detecting explosion in
53       * bullets that explode on first impact.
54       * @param physicsEntity
55       */
56      virtual void startContact(Worms::PhysicsEntity *physicsEntity) override;
57      virtual void endContact(Worms::PhysicsEntity *physicsEntity) override;
58      /**
59       * return true if the bullet is under the water, if its timeout (in the
60       * case that it have it) has been reached, or if it has collided with
61       * something
62       * @return
63       */
64      bool hasExploded() const;
65      Config::Bullet::DamageInfo getDamageInfo() const;
66      bool operator<(Worms::Bullet &other);
```

jun 26, 18 17:16 **Bullet.h** Page 2/2

```
67      Worm::WeaponID getWeaponID() const;
68
69     private:
70      b2Body *body{nullptr};
71      b2BodyDef bodyDef;
72      b2CircleShape shape;
73      b2FixtureDef fixture;
74      Worms::Physics &physics;
75      bool impulseApplied{false};
76      float timeElapsed{0.0f};
77      bool madeImpact{false};
78      Worm::WeaponID weaponID;
79      BulletInfo info;
80      bool keepUpdating{true};
81      Math::Point<float> lastPosition{0, 0};
82
83      void destroyBody();
84  };
85  }  // namespace Worms
86
87  #endif  //__BULLET_H__
```

jun 26, 18 17:16 **Bullet.cpp** Page 1/3

```
1   /*
2    *  Created by Federico Manuel Gomez Peter.
3    *  date: 26/05/18
4    */
5
6   #include <cmath>
7   #include <iostream>
8
9   #include "../Config/Config.h"
10  #include "Bullet.h"
11  #include "Weapon.h"
12  #include "../Physics.h"
13  #include "../PhysicsEntity.h"
14
15  Worms::Bullet::Bullet(BulletInfo &info, Worms::Physics &physics, Worm::WeaponID
    weapon)
16      : PhysicsEntity(Worms::EntityID::EtBullet), physics(physics), weaponID(weapo
    n), info(info) {
17      float distance = info.safeNonContactDistance + info.radius;
18      this→bodyDef.type = b2_dynamicBody;
19      this→bodyDef.position.Set(info.point.x + distance * cos(info.angle * PI / 1
    80.0f),
20                                info.point.y + distance * sin(info.angle * PI / 1
    80.0f));
21      this→bodyDef.fixedRotation = true;
22
23      this→body = this→physics.createBody(this→bodyDef);
24      this→shape.m_p.Set(0.0f, 0.0f);
25      this→shape.m_radius = info.radius;
26      this→fixture.shape = &this→shape;
27      this→fixture.density = 1.0f;
28      this→fixture.restitution = info.restitution;
29      this→fixture.friction = info.friction;
30
31      this→body→CreateFixture(&this→fixture);
32      this→body→SetUserData(this);
33
34      //    this->body->SetTransform(this->body->GetPosition(), info.angle);
35  }
36
37  void Worms::Bullet::update(float dt, Config::Wind wind) {
38      if (this→keepUpdating) {
39          this→timeElapsed += dt;
40          if (¬this→impulseApplied) {
41              float32 mass = this→body→GetMass();
42              b2Vec2 impulses = {mass * float32(this→info.power * this→info.damp
    ingRatio *
43                                                cos(this→info.angle * PI / 180.0f
    )),
44                                 mass * float32(this→info.power * this→info.damp
    ingRatio *
45                                                sin(this→info.angle * PI / 180.0f
    ))};
46              b2Vec2 position = this→body→GetWorldCenter();
47              this→body→ApplyLinearImpulse(impulses, position, true);
48              this→impulseApplied = true;
49          } else {
50              b2Vec2 velocity = this→body→GetLinearVelocity();
51              this→info.angle = atan2(velocity.y, velocity.x) * 180.0f / PI;
52              if (this→info.angle < 0) {
53                  this→info.angle += 360.0f;
54              }
55          }
56
57          if (this→info.windAffected) {
58              this→body→ApplyForceToCenter(b2Vec2{wind.instensity * wind.xDirect
```

```
    ion, 0.0f}, true);
59          }
60
61          if (this→hasExploded()) {
62              this→notify(*this, this→info.explodeEvent);
63              this→weaponID = Worm::WeaponID::WExplode;
64              this→keepUpdating = false;
65              b2Vec2 lastP = this→body→GetPosition();
66              this→lastPosition = {lastP.x, lastP.y};
67              this→destroyBody();
68          }
69      }
70  }
71
72  Math::Point<float> Worms::Bullet::getPosition() const {
73      if (this→keepUpdating) {
74          b2Vec2 p = this→body→GetPosition();
75          return Math::Point<float>(p.x, p.y);
76      } else {
77          return this→lastPosition;
78      }
79  }
80
81  float Worms::Bullet::getAngle() const {
82      return (this→info.angle ≥ 0 ∧ this→info.angle < 90) ? this→info.angle + 36
    0.0f
83                                                          : this→info.angle;
84  }
85
86  void Worms::Bullet::startContact(Worms::PhysicsEntity *physicsEntity) {
87      this→madeImpact = true;
88  }
89
90  void Worms::Bullet::endContact(Worms::PhysicsEntity *physicsEntity) {}
91
92  Worms::Bullet::~Bullet() {
93      this→destroyBody();
94  }
95
96  bool Worms::Bullet::hasExploded() const {
97      if (this→getPosition().y < Game::Config::getInstance().getWaterLevel()) {
98          return true;
99      }
100     if (this→info.explotionTimeout > 0) {
101         return this→timeElapsed ≥ this→info.explotionTimeout;
102     } else {
103         return this→madeImpact;
104     }
105 }
106
107 Config::Bullet::DamageInfo Worms::Bullet::getDamageInfo() const {
108     return this→info.dmgInfo;
109 }
110
111 bool Worms::Bullet::operator<(Worms::Bullet &other) {
112     return this→timeElapsed > other.timeElapsed;
113 }
114
115 Worm::WeaponID Worms::Bullet::getWeaponID() const {
116     return this→weaponID;
117 }
118
119 void Worms::Bullet::destroyBody() {
120     if (this→body ≠ nullptr) {
121         this→body→GetWorld()→DestroyBody(this→body);
122         this→body = nullptr;
```



```
123     }
124 }
```

jun 26, 18 17:16 **Bazooka.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#ifndef __BAZOOKA_H__
#define __BAZOOKA_H__

#include "Weapon.h"

namespace Weapon {
class Bazooka : public Worms::Weapon {
   public:
    Bazooka(float angle);
    ~Bazooka() = default;
    void update(float dt) override;
    void startShot(Worms::Player *player) override;
    void endShot() override;
    void setTimeout(uint8_t time) override;
    std::list<Worms::Bullet> onExplode(const Worms::Bullet &mainBullet,
                                       Worms::Physics &physics) override;
    void positionSelected(Worms::Player &p, Math::Point<float> point) override;

   private:
    float powerChargeTime{0.0f};
    Worms::Player *player;
};
}  // namespace Weapon

#endif  //__BAZOOKA_H__
```

jun 26, 18 17:16 **Bazooka.cpp** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#include "Bazooka.h"
#include "../Player.h"

Weapon::Bazooka::Bazooka(float angle)
    : Worms::Weapon(Game::Config::getInstance().getBazookaConfig(), Worm::WeaponID::WBazooka,
                    angle) {
    this→powerChargeTime = Game::Config::getInstance().getPowerChargeTime();
}

void Weapon::Bazooka::update(float dt) {
    if (this→increaseShotPower) {
        if (this→shotPower < this→config.maxShotPower) {
            this→shotPower += dt / this→powerChargeTime * this→config.maxShotPower;
        } else {
            this→player→endShot();
        }
    }
}

void Weapon::Bazooka::startShot(Worms::Player *player) {
    this→increaseShotPower = true;
    this→player = player;
}

void Weapon::Bazooka::endShot() {
    this→increaseShotPower = false;
    this→shotPower = 0;
}

void Weapon::Bazooka::setTimeout(uint8_t time) {}

std::list<Worms::Bullet> Weapon::Bazooka::onExplode(const Worms::Bullet &mainBullet,
                                                    Worms::Physics &physics) {
    return std::move(std::list<Worms::Bullet>());
}

void Weapon::Bazooka::positionSelected(Worms::Player &p, Math::Point<float> point) {}
```

jun 26, 18 17:16 **BaseballBat.h** Page 1/1

```
1  //
2  // Created by rodrigo on 16/06/18.
3  //
4
5  #ifndef INC_4_WORMS_BASEBALLBAT_H
6  #define INC_4_WORMS_BASEBALLBAT_H
7
8  #include "../Config/P2PWeapon.h"
9  #include "../Physics.h"
10 #include "Weapon.h"
11
12 namespace Weapon {
13 class BaseballBat : public Worms::Weapon {
14    public:
15     BaseballBat(float angle);
16     ~BaseballBat() = default;
17     void update(float dt) override;
18     void startShot(Worms::Player *player) override;
19     void endShot() override;
20     void setTimeout(uint8_t time) override;
21     std::list<Worms::Bullet> onExplode(const Worms::Bullet &mainBullet,
22                                        Worms::Physics &physics) override;
23     void positionSelected(Worms::Player &p, Math::Point<float> point) override;
24     Config::P2PWeapon &getWeaponInfo();
25
26    private:
27     Config::P2PWeapon weaponInfo;
28 };
29 }  // namespace Weapon
30
31 #endif  // INC_4_WORMS_BASEBALLBAT_H
```

jun 26, 18 17:16 **BaseballBat.cpp** Page 1/1

```
1  //
2  // Created by rodrigo on 16/06/18.
3  //
4
5  #include "BaseballBat.h"
6  #include "../Player.h"
7  #include "Direction.h"
8
9  Weapon::BaseballBat::BaseballBat(float angle)
10     : Worms::Weapon(Game::Config::getInstance().getBaseballBatConfig(),
11                     Worm::WeaponID::WBaseballBat, angle),
12       weaponInfo{this→config.dmgInfo, Worm::Direction::left, {0, 0}} {
13     this→isP2P = true;
14 }
15
16 void Weapon::BaseballBat::update(float dt) {}
17
18 void Weapon::BaseballBat::startShot(Worms::Player *player) {
19     this→weaponInfo.position = player→getPosition();
20     this→weaponInfo.direction = player→direction;
21     this→weaponInfo.angle = this→angle;
22 }
23
24 void Weapon::BaseballBat::endShot() {}
25
26 void Weapon::BaseballBat::setTimeout(uint8_t time) {}
27
28 std::list<Worms::Bullet> Weapon::BaseballBat::onExplode(const Worms::Bullet &mai
   nBullet,
29                                                         Worms::Physics &physics)
    {
30     return std::move(std::list<Worms::Bullet>());
31 }
32
33 void Weapon::BaseballBat::positionSelected(Worms::Player &p, Math::Point<float>
   point) {}
34
35 Config::P2PWeapon &Weapon::BaseballBat::getWeaponInfo() {
36     return this→weaponInfo;
37 }
```

jun 26, 18 17:16 **Banana.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#ifndef __Banana_H__
#define __Banana_H__

#include "Weapon.h"

namespace Weapon {
class Banana : public Worms::Weapon {
   public:
    Banana(float angle);
    ~Banana() override = default;
    void update(float dt) override;
    void startShot(Worms::Player *player) override;
    void endShot() override;
    void setTimeout(uint8_t time) override;
    std::list<Worms::Bullet> onExplode(const Worms::Bullet &mainBullet,
                                       Worms::Physics &physics) override;
    void positionSelected(Worms::Player &p, Math::Point<float> point) override;

   private:
    float powerChargeTime{0.0f};
};
}  // namespace Weapon

#endif  //__Banana_H__
```

jun 26, 18 17:16 **Banana.cpp** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 03/06/18
 */

#include "Banana.h"
#include "../Player.h"

Weapon::Banana::Banana(float angle)
    : Worms::Weapon(Game::Config::getInstance().getBananaConfig(), Worm::WeaponID::WBanana, angle) {
    this→powerChargeTime = Game::Config::getInstance().getPowerChargeTime();
}

void Weapon::Banana::update(float dt) {
    if (this→increaseShotPower) {
        if (this→shotPower < this→config.maxShotPower) {
            this→shotPower += dt / this→powerChargeTime * this→config.maxShotPower;
        }
    }
}

void Weapon::Banana::startShot(Worms::Player *player) {
    this→increaseShotPower = true;
}

void Weapon::Banana::endShot() {
    this→increaseShotPower = false;
    this→shotPower = 0;
}

void Weapon::Banana::setTimeout(uint8_t time) {
    this→timeLimit = time;
}

std::list<Worms::Bullet> Weapon::Banana::onExplode(const Worms::Bullet &mainBullet,
                                                   Worms::Physics &physics) {
    return std::move(std::list<Worms::Bullet>());
}

void Weapon::Banana::positionSelected(Worms::Player &p, Math::Point<float> point) {}
```

jun 26, 18 17:16 **AerialAttack.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 16/06/18
 */

#ifndef __AERIAL_ATTACK_H__
#define __AERIAL_ATTACK_H__

#include "../Player.h"
#include "Weapon.h"

namespace Weapon {
class AerialAttack : public Worms::Weapon {
   public:
    AerialAttack();
    ~AerialAttack() override = default;
    void update(float dt) override;
    void startShot(Worms::Player *player) override;
    void endShot() override;
    void setTimeout(uint8_t time) override;
    std::list<Worms::Bullet> onExplode(const Worms::Bullet &mainBullet,
                                       Worms::Physics &physics) override;
    void positionSelected(Worms::Player &p, Math::Point<float> point) override;

   private:
    const uint8_t bulletsQuantity{0};
    const float missileSeparation{0};
};
}  // namespace Weapon

#endif  //__AERIAL_ATTACK_H__
```

jun 26, 18 17:16 **AerialAttack.cpp** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 16/06/18
 */

#include "AerialAttack.h"
#define CONFIG Game::Config::getInstance()

Weapon::AerialAttack::AerialAttack()
    : Weapon::Weapon(CONFIG.getAerialAttackConfig(), Worm::WeaponID::WAerial, 0.0),
      bulletsQuantity(CONFIG.getAerialAttackMissileQuantity()),
      missileSeparation(CONFIG.getAerialAttackMissileSeparation()) {}

void Weapon::AerialAttack::update(float dt) {}

void Weapon::AerialAttack::startShot(Worms::Player *player) {}

void Weapon::AerialAttack::endShot() {}

void Weapon::AerialAttack::setTimeout(uint8_t time) {}

std::list<Worms::Bullet> Weapon::AerialAttack::onExplode(const Worms::Bullet &mainBullet,
                                                         Worms::Physics &physics) {
    return std::move(std::list<Worms::Bullet>());
}

void Weapon::AerialAttack::positionSelected(Worms::Player &p, Math::Point<float> point) {
    point.y += CONFIG.getAerialAttackLaunchHeight();
    point.x -= this→missileSeparation * (this→bulletsQuantity + 1) / 2;

    Worms::BulletInfo bulletInfo = {this→config.dmgInfo,
                                    point,
                                    this→config.minAngle,
                                    (float)this→config.maxShotPower,
                                    0,
                                    this→config.restitution,
                                    this→config.friction,
                                    this→config.explotionInitialTimeout,
                                    Event::Explode,
                                    this→config.bulletRadius,
                                    this→config.bulletDampingRatio,
                                    this→config.windAffected};

    std::list<Worms::Bullet> ret;
    for (int i = 0; i < this→bulletsQuantity; i++) {
        point.x += this→missileSeparation;
        bulletInfo.point = point;
        ret.emplace_back(bulletInfo, p.getPhysics(), Worm::WeaponID::WAerial);
    }

    p.endShot(ret);
}
```

jun 26, 18 17:16 **TouchSensor.h** Page 1/1

```
#ifndef TOUCH_SENSOR_H_
#define TOUCH_SENSOR_H_

#include <unordered_map>
#include <vector>

#include "Physics.h"
#include "PhysicsEntity.h"

namespace Worms {
class TouchSensor : public PhysicsEntity {
   public:
    using iterator = std::unordered_map<PhysicsEntity *, b2Vec2>::iterator;

    TouchSensor(b2Body &body, b2Shape &shape);
    ~TouchSensor() = default;

    iterator begin();
    iterator end();

    bool isActive() const;
    void ignore(PhysicsEntity &other);

    void startContact(PhysicsEntity *physicsEntity, b2Contact &contact);
    void endContact(PhysicsEntity *physicsEntity, b2Contact &contact);

   private:
    bool isIgnored(PhysicsEntity *entity);

    b2Fixture *fixture{nullptr};
    std::vector<PhysicsEntity *> ignoredEntities;
    std::unordered_map<PhysicsEntity *, b2Vec2> contacts;
    std::unordered_map<PhysicsEntity *, b2Fixture *> contactFixtures;
};
}  // namespace Worms

#endif
```

jun 26, 18 17:16 **TouchSensor.cpp** Page 1/2

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

/**
 * @brief Construct a TouchSensor for the given body and shape.
 *
 * @param body Body that the touch sensor belongs to.
 * @param shape Sensor shape.
 */
Worms::TouchSensor::TouchSensor(b2Body &body, b2Shape &shape) : PhysicsEntity(EntityID::Sensor) {
    /* fixture definition using the given shape */
    b2FixtureDef fixtureDef;
    fixtureDef.shape = &shape;
    fixtureDef.density = 1;
    fixtureDef.isSensor = true;

    this→fixture = body.CreateFixture(&fixtureDef);
    this→fixture→SetUserData(this);
}

Worms::TouchSensor::iterator Worms::TouchSensor::begin() {
    return this→contacts.begin();
}

Worms::TouchSensor::iterator Worms::TouchSensor::end() {
    return this→contacts.end();
}

/**
 * @brief Called whenever the sensor started contacting another entity.
 *
 * @param Contacted entity.
 */
void Worms::TouchSensor::startContact(PhysicsEntity *physicsEntity, b2Contact &contact) {
    /* checks if the entity is in the ignore list */
    if (¬this→isIgnored(physicsEntity)) {
        b2Manifold manifold;
        const b2Transform t1 = contact.GetFixtureA()→GetBody()→GetTransform();
        const b2Transform t2 = contact.GetFixtureB()→GetBody()→GetTransform();

        contact.Evaluate(&manifold, t1, t2);

        if (contact.GetFixtureA()→GetUserData() ≡ physicsEntity) {
            this→contacts[physicsEntity] = −manifold.localNormal;
        } else {
            this→contacts[physicsEntity] = manifold.localNormal;
        }
    }
}

/**
 * @brief Called whenever the sensor stopped contacting another entity.
 *
 * @param Entity.
 */
void Worms::TouchSensor::endContact(PhysicsEntity *physicsEntity, b2Contact &contact) {
    /* checks if the entity is in the ignore list */
    if (¬this→isIgnored(physicsEntity)) {
        this→contacts.erase(physicsEntity);
    }
}

/**
```

jun 26, 18 17:16 **TouchSensor.cpp** Page 2/2

```
64   * @brief Whether the sensor is active or not (i.e. touching another body).
65   *
66   * @return true is active.
67   */
68  bool Worms::TouchSensor::isActive() const {
69      return (this→contacts.size() > 0);
70  }
71
72  /**
73   * @brief Adds an entity that should be ignored by the sensor.
74   *
75   * @param other Entity to ignore.
76   */
77  void Worms::TouchSensor::ignore(PhysicsEntity &other) {
78      this→ignoredEntities.push_back(&other);
79  }
80
81  /**
82   * @brief Checks if a given entity is in the ignore list.
83   *
84   * @param entity Entity to check.
85   * @return true if the given entity is ignored by this sensor.
86   */
87  bool Worms::TouchSensor::isIgnored(PhysicsEntity *entity) {
88      return std::find(this→ignoredEntities.begin(), this→ignoredEntities.end(),
     entity) ≠
89              this→ignoredEntities.end();
90  }
```

jun 26, 18 17:16 **Team.h** Page 1/1

```
1   //
2   // Created by rodrigo on 3/06/18.
3   //
4
5   #ifndef INC_4_WORMS_TEAM_H
6   #define INC_4_WORMS_TEAM_H
7
8   #include <stdint.h>
9   #include <cstdint>
10  #include <map>
11  #include <vector>
12
13  #include "Weapons/Weapon.h"
14
15  namespace Worms {
16  class Player;
17  class Team {
18     public:
19      Team(std::vector<uint8_t> &playerIDs, std::vector<Player> &players,
20           const std::map<Worm::WeaponID, std::int16_t> &stageAmmo);
21      ~Team() = default;
22      void checkAlive(std::vector<Player> &players);
23      bool isAlive();
24      uint8_t getCurrentPlayerID();
25      void setCurrentPlayer(uint8_t currentPlayer);
26      void endTurn(std::vector<Worms::Player> &players);
27      std::uint32_t calculateTotalHealth(std::vector<Worms::Player> &players);
28      std::shared_ptr<Weapon> getWeapon(const Worm::WeaponID &id);
29      void weaponUsed(const Worm::WeaponID weaponID);
30      void serialize(IO::GameStateMsg &msg) const;
31      void kill(std::vector<Worms::Player> &players);
32
33     private:
34      std::vector<uint8_t> playerIDs;
35      uint8_t currentPlayer{0};
36      bool alive{true};
37      std::shared_ptr<Weapon> aerialAttack{nullptr};
38      std::shared_ptr<Weapon> banana{nullptr};
39      std::shared_ptr<Weapon> baseballBat{nullptr};
40      std::shared_ptr<Weapon> bazooka{nullptr};
41      std::shared_ptr<Weapon> cluster{nullptr};
42      std::shared_ptr<Weapon> dynamite{nullptr};
43      std::shared_ptr<Weapon> grenade{nullptr};
44      std::shared_ptr<Weapon> holy{nullptr};
45      std::shared_ptr<Weapon> mortar{nullptr};
46      std::shared_ptr<Weapon> teleport{nullptr};
47      std::map<Worm::WeaponID, std::int16_t> ammunitionCounter;
48      std::shared_ptr<Weapon> weaponNone;
49
50      void initializeWeapons();
51  };
52  }  // namespace Worms
53
54  #endif  // INC_4_WORMS_TEAM_H
```

```
//
// Created by rodrigo on 3/06/18.
//

#include "Team.h"
#include "Weapons/AerialAttack.h"
#include "Weapons/Banana.h"
#include "Weapons/BaseballBat.h"
#include "Weapons/Bazooka.h"
#include "Weapons/Cluster.h"
#include "Weapons/Dynamite.h"
#include "Weapons/Grenade.h"
#include "Weapons/Holy.h"
#include "Weapons/Mortar.h"
#include "Weapons/Teleport.h"
#include "Weapons/WeaponNone.h"

Worms::Team::Team(std::vector<uint8_t> &playerIDs, std::vector<Player> &players,
                  const std::map<Worm::WeaponID, std::int16_t> &stageAmmo)
    : playerIDs(std::move(playerIDs)), ammunitionCounter(stageAmmo) {
    for (auto id : this→playerIDs) {
        players[id].setTeam(this);
    }
    this→initializeWeapons();
}

void Worms::Team::checkAlive(std::vector<Player> &players) {
    if (this→alive) {
        bool teamAlive = false;
        for (auto teamPlayerID : this→playerIDs) {
            if (players[teamPlayerID].getStateId() ≠ Worm::StateID::Dead) {
                teamAlive = true;
            }
        }
        if (¬teamAlive) {
            this→alive = false;
        }
    }
}

bool Worms::Team::isAlive() {
    return this→alive;
}

uint8_t Worms::Team::getCurrentPlayerID() {
    return this→playerIDs[this→currentPlayer];
}

void Worms::Team::setCurrentPlayer(uint8_t currentPlayer) {
    this→currentPlayer = currentPlayer;
}

void Worms::Team::endTurn(std::vector<Worms::Player> &players) {
    do {
        this→currentPlayer = (this→currentPlayer + 1) % this→playerIDs.size();
    } while (players[this→getCurrentPlayerID()].getStateId() ≡ Worm::StateID::Dead);
}

std::uint32_t Worms::Team::calculateTotalHealth(std::vector<Worms::Player> &players) {
    std::uint32_t total{0};
    for (auto playerID : this→playerIDs) {
        for (auto &player : players) {
            if (player.getId() ≡ playerID) {
                total += (std::uint32_t)std::floor(player.health);
```



```
            }
        }
    }
    return total;
}

std::shared_ptr<Worms::Weapon> Worms::Team::getWeapon(const Worm::WeaponID &id)
{
    if (this→ammunitionCounter.at(id) ≡ 0) {
        return this→weaponNone;
    }

    switch (id) {
        case Worm::WeaponID::WBazooka:
            return this→bazooka;
        case Worm::WeaponID::WGrenade:
            return this→grenade;
        case Worm::WeaponID::WCluster:
            return this→cluster;
        case Worm::WeaponID::WMortar:
            return this→mortar;
        case Worm::WeaponID::WBanana:
            return this→banana;
        case Worm::WeaponID::WHoly:
            return this→holy;
        case Worm::WeaponID::WAerial:
            return this→aerialAttack;
        case Worm::WeaponID::WDynamite:
            return this→dynamite;
        case Worm::WeaponID::WBaseballBat:
            return this→baseballBat;
        case Worm::WeaponID::WTeleport:
            return this→teleport;
        default:
            return this→weaponNone;
    }
}

void Worms::Team::initializeWeapons() {
    this→aerialAttack = std::shared_ptr<Worms::Weapon>(new ::Weapon::AerialAttack());
    this→banana = std::shared_ptr<Worms::Weapon>(new ::Weapon::Banana(0.0f));
    this→baseballBat = std::shared_ptr<Worms::Weapon>(new ::Weapon::BaseballBat(0.0f));
    this→bazooka = std::shared_ptr<Worms::Weapon>(new ::Weapon::Bazooka(0.0f));
    this→cluster = std::shared_ptr<Worms::Weapon>(new ::Weapon::Cluster(0.0f));
    this→dynamite = std::shared_ptr<Worms::Weapon>(new ::Weapon::Dynamite());
    this→grenade = std::shared_ptr<Worms::Weapon>(new ::Weapon::Grenade(0.0f));
    this→holy = std::shared_ptr<Worms::Weapon>(new ::Weapon::Holy(0.0f));
    this→mortar = std::shared_ptr<Worms::Weapon>(new ::Weapon::Mortar(0.0f));
    this→teleport = std::shared_ptr<Worms::Weapon>(new ::Weapon::Teleport());
    this→weaponNone = std::shared_ptr<Worms::Weapon>(new ::Weapon::WeaponNone());
}

void Worms::Team::weaponUsed(const Worm::WeaponID weaponID) {
    if (this→ammunitionCounter.at(weaponID) > 0) {
        this→ammunitionCounter.at(weaponID)−−;
    }
}

void Worms::Team::serialize(IO::GameStateMsg &msg) const {
    Worm::WeaponID weapons[] = {Worm::WBazooka,     Worm::WGrenade, Worm::WCluster, Worm::WMortar,
                                Worm::WBanana,      Worm::WHoly,    Worm::WAerial,  Worm::WDynamite,
```

jun 26, 18 17:16 **Team.cpp** Page 3/3

```
                                Worm::WBaseballBat, Worm::WTeleport};

    for (int i = 0; i < 10; i++) {
        msg.weaponAmmunition[i] = this→ammunitionCounter.at(weapons[i]);
    }
}

void Worms::Team::kill(std::vector<Worms::Player> &players) {
    for (auto &playerID : this→playerIDs) {
        players[playerID].die();
    }
    this→alive = false;
}
```

jun 26, 18 17:16 **ServerSocket.h** Page 1/1

```
/*
 * Created by Federico Manuel Gomez Peter
 * Date: 02/05/2018.
 */

#ifndef __SERVERSOCKET_H__
#define __SERVERSOCKET_H__

#include <string>

#include "CommunicationSocket.h"
#include "Socket.h"

class ServerSocket : public Socket {
   public:
    explicit ServerSocket(const char *port);
    /**
     * Acepta una conexiÃ³n y devuelve un CommunicationSocket por movimiento.
     * @return Socket para comunicacion
     */
    CommunicationSocket accept();
    void bindAndListen(const char *port);
};

#endif  //__SERVERSOCKET_H__
```

jun 26, 18 17:16 **ServerSocket.cpp** Page 1/2

```
/*
 * Created by Federico Manuel Gomez Peter
 * Date: 02/05/2018.
 */

#include <netdb.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#include <cstring>

#include "ErrorMessages.h"
#include "Exception.h"
#include "ServerSocket.h"

ServerSocket::ServerSocket(const char *port) {
    this→bindAndListen(port);
}

void ServerSocket::bindAndListen(const char *port) {
    int status = 0;
    int option_value = 1;
    bool is_bound = false;
    /*
    * inicializo el bloque de memoria de addrinfo,
    * lo configuro para que result sea una lista de
    * address pertenecientes a IPv4, y que sean TCP.
    */
    struct addrinfo hints = {AI_PASSIVE, AF_INET, SOCK_STREAM, 0, 0, nullptr, nu
llptr, nullptr};
    struct addrinfo *result, *ptr;

    status = getaddrinfo(nullptr, port, &hints, &result);
    if (status ≠ 0) {
        throw Exception(ERR_MSG_SOCKET_INVALID_PORT, port, gai_strerror(status))
;
    }
    /*
    * Recorro los resultados posibles, hasta poder bindear
    */
    for (ptr = result; ptr ≠ nullptr ∧ ¬is_bound; ptr = ptr→ai_next) {
        this→fd = ::socket(ptr→ai_family, ptr→ai_socktype, ptr→ai_protocol);
        /*
        * si la creaciÃ³n del socket falla, no debo hacer nada mas
        * en el ciclo (ya que no se abrio ningun fd)
        */
        if (this→fd ≡ −1) {
            continue;
        }
        /*
        * Del ejemplo del echoserver, se obtuvo la forma de
        * configurar la reutilizaciÃ³n de la direcciÃ³n
        * que no se encuentre disponible por un TIME_WAIT.
        * si la configuraciÃ³n falla, debo liberar
        * el socket (segun la documentaciÃ³n y el ejemplo
        * que se encuentra en el manual de getaddrinfo)
        */
        status =
            setsockopt(this→fd, SOL_SOCKET, SO_REUSEADDR, &option_value, sizeof
(option_value));
        if (status ≡ −1) {
            this→close();
            continue;
        }
        /*
```

jun 26, 18 17:16 **ServerSocket.cpp** Page 2/2

```
        * Si logro bindear, salgo del ciclo, sino, cierro el socket
        * y pruebo en el siguiente resultado.
        */
        status = bind(this→fd, result→ai_addr, result→ai_addrlen);
        if (status ≡ −1) {
            this→close();
        } else {
            is_bound = true;
        }
    }
    freeaddrinfo(result);

    if (¬is_bound) {
        throw Exception(ERR_MSG_SOCKET_BINDING, port);
    }

    status = listen(this→fd, 20);
    if (status ≡ −1) {
        throw Exception(ERR_MSG_SOCKET_LISTEN, strerror(errno));
    }
}

CommunicationSocket ServerSocket::accept() {
    int fd = ::accept(this→fd, nullptr, nullptr);
    if (fd ≡ −1) {
        throw Exception(ERR_MSG_SOCKET_ACCEPT, strerror(errno));
    }
    return std::move(CommunicationSocket(fd));
}
```

```
1   /*
2    *  Created by Federico Manuel Gomez Peter.
3    *  date: 18/05/18
4    */
5
6   #ifndef __PLAYER_H__
7   #define __PLAYER_H__
8
9   #define PLAYER_WIDTH 0.8f
10
11  #define PLAYER_HEIGHT 2.0f
12
13  #include <list>
14
15  #include "Config/Config.h"
16  #include "Config/P2PWeapon.h"
17  #include "Direction.h"
18  #include "GameStateMsg.h"
19  #include "Physics.h"
20  #include "Point.h"
21  #include "Stream.h"
22  #include "Team.h"
23  #include "TouchSensor.h"
24  #include "Weapons/Bullet.h"
25  #include "Weapons/Weapon.h"
26  #include "WormStates/PlayerState.h"
27
28  enum class PlayerState { movingRight, movingLeft, still };
29
30  namespace Worms {
31
32  class Player : public PhysicsEntity {
33     public:
34      Worm::Direction direction{Worm::Direction::left};
35      Worm::Direction lastWalkDirection;
36      std::uint16_t health{0};
37      Math::Point<float> teleportPosition{0.0f, 0.0f};
38      bool dyingDisconnected{false};
39
40      explicit Player(Physics &physics);
41      Player(Player ∧player) noexcept;
42      Player(Player &copy) = delete;
43
44      ~Player();
45
46      /* contact handlers */
47      virtual void contactWith(PhysicsEntity &other, b2Contact &contact);
48
49      bool isOnGround() const;
50
51      /**
52       * Updates its state, its weapon
53       * @param dt
54       */
55      void update(float dt);
56      void serialize(IO::Stream<IO::GameStateMsg> &s) const {}
57      /**
58       * @brief moves the player to newPos position
59       * @param newPos
60       */
61      void setPosition(const Math::Point<float> &newPos);
62      b2Vec2 getGroundNormal() const;
63      void startContact(Worms::PhysicsEntity *physicsEntity, b2Contact &contact);
64
65      /**
66       * @brief asks box2D from current position.
```



```
67       * @return
68       */
69      Math::Point<float> getPosition() const;
70      /**
71       * @brief given playerInput, changes its state (or its weapon) accordingly
72       * @param pi
73       */
74      void handleState(IO::PlayerMsg pi);
75      const std::shared_ptr<Worms::Weapon> getWeapon() const;
76      Worm::StateID getStateId() const;
77      void setState(Worm::StateID stateID);
78      float getWeaponAngle() const;
79      const Worm::WeaponID &getWeaponID() const;
80      void setWeapon(const Worm::WeaponID &id);
81      /**
82       * @brief delegates on its weapon the action of increase the angle, if
83       * the weapon handles it.
84       */
85      void increaseWeaponAngle();
86      /**
87       * @brief delegates on its weapon the action of decrease the angle, if
88       * the weapon handles it.
89       */
90      void decreaseWeaponAngle();
91      /**
92       * @brief delegates on its weapon the action of starting a shot, increasing
93       * its powerShot if it handles it
94       */
95      void startShot();
96      /**
97       * @brief creates a bullet that needs to be moved using getBullet()
98       */
99      void endShot();
100     void acknowledgeDamage(Config::Bullet::DamageInfo damageInfo, Math::Point<fl
    oat> epicenter);
101     void acknowledgeDamage(const Config::P2PWeapon &info, Math::Point<float> sho
    oterPosition,
102                            Worm::Direction shooterDirection);
103     void landDamage(float yDistance);
104     void setTeamID(uint8_t team);
105     void setTeam(Worms::Team *team);
106     void increaseHealth(float extraPoints);
107     uint8_t getTeam() const;
108     void setId(uint8_t id);
109     uint8_t getId() const;
110     Physics &getPhysics();
111     void setWeaponTimeout(uint8_t time);
112     /**
113      * Moves the bullets to the caller (the Game)
114      * @return bullets
115      */
116     std::list<Bullet> getBullets();
117     /**
118      * Resets the weapon's powershot and erase every possible bullet
119      * inside his container.
120      */
121     void reset();
122     /**
123      * calls weapon's onExplode and get new bullets if it is necesary.
124      */
125     std::list<Bullet> onExplode(const Bullet &bullet, Physics &physics);
126
127     bool operator≠(const Player &other);
128     bool operator≡(const Player &other);
129
130     void endShot(std::list<Worms::Bullet> &bullets);
```

jun 26, 18 17:16 **Player.h** Page 3/3

```
131     void die();
132
133    private:
134     b2Body *createBody(b2BodyType type);
135
136     b2Body *body{nullptr};
137     b2Body *body_kinematic{nullptr};
138     TouchSensor *footSensor;
139
140     std::shared_ptr<Worms::State> state{nullptr};
141     std::shared_ptr<Worms::Weapon> weapon{nullptr};
142     Physics &physics;
143     const int waterLevel;
144     uint8_t teamID;
145     uint8_t id;
146     std::list<Bullet> bullets;
147     bool isP2PWeapon{false};
148     b2Vec2 lastGroundNormal{0.0f, 0.0f};
149     Team *team{nullptr};
150 };
151 }  // namespace Worms
152
153 #endif  //__PLAYER_H__
```

jun 26, 18 17:16 **Player.cpp** Page 1/10

```
1  /*
2   *  Created by Federico Manuel Gomez Peter.
3   *  date: 18/05/18
4   */
5
6  #include <Box2D/Box2D.h>
7  #include <iostream>
8
9  #include "Direction.h"
10 #include "Girder.h"
11 #include "Physics.h"
12 #include "Player.h"
13 #include "Weapons/AerialAttack.h"
14 #include "Weapons/Banana.h"
15 #include "Weapons/BaseballBat.h"
16 #include "Weapons/Bazooka.h"
17 #include "Weapons/Cluster.h"
18 #include "Weapons/Dynamite.h"
19 #include "Weapons/Grenade.h"
20 #include "Weapons/Holy.h"
21 #include "Weapons/Mortar.h"
22 #include "Weapons/Teleport.h"
23 #include "Weapons/Weapon.h"
24 #include "WormStates/BackFlipping.h"
25 #include "WormStates/Batting.h"
26 #include "WormStates/Dead.h"
27 #include "WormStates/Die.h"
28 #include "WormStates/Drowning.h"
29 #include "WormStates/EndBackFlip.h"
30 #include "WormStates/EndJump.h"
31 #include "WormStates/Falling.h"
32 #include "WormStates/Hit.h"
33 #include "WormStates/Jumping.h"
34 #include "WormStates/Land.h"
35 #include "WormStates/Sliding.h"
36 #include "WormStates/StartBackFlip.h"
37 #include "WormStates/StartJump.h"
38 #include "WormStates/Still.h"
39 #include "WormStates/Teleported.h"
40 #include "WormStates/Teleporting.h"
41 #include "WormStates/Walk.h"
42 #include "Weapons/WeaponNone.h"
43
44 #define CONFIG Game::Config::getInstance()
45
46 Worms::Player::Player(Physics &physics)
47     : PhysicsEntity(Worms::EntityID::EtWorm), physics(physics), waterLevel(CONFI
   G.getWaterLevel()) {
48     /* creates 2 bodies so players cannot move each other */
49     this→body = this→createBody(b2_dynamicBody);
50     this→body_kinematic = this→createBody(b2_kinematicBody);
51
52     /* creates the sensor as a circle */
53     b2CircleShape sensorShape;
54     sensorShape.m_radius = PLAYER_HEIGHT / 4;
55     sensorShape.m_p.Set(0.0f, -PLAYER_HEIGHT / 4 - 0.2);
56
57     /* allocated in heap because it's address shouldn't change */
58     this→footSensor = new TouchSensor{*this→body, sensorShape};
59     this→footSensor→ignore(*this);
60
61     this→setState(Worm::StateID::Falling);
62     this→weapon = std::shared_ptr<Worms::Weapon>(new ::Weapon::Bazooka(0.0f));
63 }
64
65 Worms::Player::~Player() {
```

```
66      delete this→footSensor;
67  }
68
69  /**
70   * @brief "Not equal" operator.
71   *
72   * @param other Other instance to compare.
73   * @return true if not equal.
74   */
75  bool Worms::Player::operator≠(const Player &other) {
76      return ¬(*this ≡ other);
77  }
78
79  /**
80   * @brief Comparisson operator.
81   *
82   * @param other Other instance to compare.
83   * @return true if equal.
84   */
85  bool Worms::Player::operator≡(const Player &other) {
86      return (this→id ≡ other.id) ∧ (this→teamID ≡ other.teamID);
87  }
88
89  /**
90   * @brief Handles player−entity contact.
91   *
92   * @param other Other player that made contact.
93   * @param contact box2D collision contact.
94   */
95  void Worms::Player::contactWith(PhysicsEntity &entity, b2Contact &contact) {
96      if (entity.getEntityId() ≡ Worms::EntityID::EtGirder) {
97          Worms::Girder &girder = dynamic_cast<Worms::Girder &>(entity);
98          if (std::abs(girder.angle) > PI / 4.0f) {
99              this→lastGroundNormal = contact.GetManifold()→localNormal;
100         } else {
101             this→lastGroundNormal = {0.0f, 0.1f};
102         }
103     }
104
105     if (entity.getEntityId() ≠ Worms::EntityID::EtWorm) {
106         return;
107     }
108
109     /* checks if it's the player itself */
110     if (&entity ≡ this) {
111         /* checks if it's the kinematic and dynamic bodies colliding */
112         if (contact.GetFixtureA()→GetBody()→GetType() ≠
113             contact.GetFixtureB()→GetBody()→GetType()) {
114             contact.SetEnabled(false);
115         }
116     }
117 }
118
119 void Worms::Player::update(float dt) {
120     /* sets the kinematic body to the position of the dynamic body */
121     this→body_kinematic→SetTransform(this→body→GetTransform().p, this→body→G
    etAngle());
122
123     this→state→update(*this, dt, this→body);
124     this→weapon→update(dt);
125
126     if (this→getPosition().y ≤ this→waterLevel ∧ this→getStateId() ≠ Worm::Sta
    teID::Dead ∧
127         this→getStateId() ≠ Worm::StateID::Drowning) {
128         this→health = 0;
129         if (this→getStateId() ≡ Worm::StateID::Hit) {
```



```
130             this→notify(*this, Event::EndHit);
131         }
132         this→setState(Worm::StateID::Drowning);
133         this→notify(*this, Event::Drowning);
134     } else if (this→isOnGround()) {
135         /* checks if the ground slope is too tilted */
136         try {
137             b2Vec2 normal = this→getGroundNormal();
138             float slope = std::abs(std::atan2(normal.y, normal.x));
139             if ((slope < PI / 4.0f) ∨ (slope > (PI * 3.0f) / 4.0f)) {
140                 if (this→getStateId() ≡ Worm::StateID::Hit) {
141                     this→notify(*this, Event::EndHit);
142                 }
143                 this→setState(Worm::StateID::Sliding);
144                 return;
145             }
146         } catch (const Exception &e) {
147         }
148     }
149 }
150
151 /**
152  * @brief Whether the player is touching the ground or not.
153  *
154  * @return true is touching the ground.
155  */
156 bool Worms::Player::isOnGround() const {
157     return this→footSensor→isActive();
158 }
159
160 void Worms::Player::setPosition(const Math::Point<float> &new_pos) {
161     this→body→SetTransform(b2Vec2(new_pos.x, new_pos.y), body→GetAngle());
162 }
163
164 /**
165  * @brief Returns a unit vector with the direction normal to the floor where the
     player is standing.
166  *
167  * @return b2Vec2 Floor normal.
168  */
169 b2Vec2 Worms::Player::getGroundNormal() const {
170     for (auto &contact : *this→footSensor) {
171         if (contact.first→getEntityId() ≡ Worms::EntityID::EtGirder) {
172             return this→lastGroundNormal;
173         }
174     }
175     throw Exception{"No ground normal"};
176 }
177
178 void Worms::Player::startContact(Worms::PhysicsEntity *physicsEntity, b2Contact
    &contact) {}
179
180 Math::Point<float> Worms::Player::getPosition() const {
181     const b2Vec2 &pos = this→body→GetPosition();
182     return Math::Point<float>{pos.x, pos.y};
183 }
184
185 Worm::StateID Worms::Player::getStateId() const {
186     return this→state→getState();
187 }
188
189 void Worms::Player::handleState(IO::PlayerMsg pi) {
190     switch (pi.input) {
191         case IO::PlayerInput::moveLeft:
192             this→state→moveLeft(*this);
193             break;
```

```
194         case IO::PlayerInput::moveRight:
195             this→state→moveRight(*this);
196             break;
197         case IO::PlayerInput::startJump:
198             this→state→jump(*this);
199             break;
200         case IO::PlayerInput::startBackFlip:
201             this→state→backFlip(*this);
202             break;
203         case IO::PlayerInput::stopMove:
204             this→state→stopMove(*this);
205             break;
206         case IO::PlayerInput::bazooka:
207             this→state→bazooka(*this);
208             break;
209         case IO::PlayerInput::grenade:
210             this→state→grenade(*this);
211             break;
212         case IO::PlayerInput::cluster:
213             this→state→cluster(*this);
214             break;
215         case IO::PlayerInput::mortar:
216             this→state→mortar(*this);
217             break;
218         case IO::PlayerInput::banana:
219             this→state→banana(*this);
220             break;
221         case IO::PlayerInput::holy:
222             this→state→holy(*this);
223             break;
224         case IO::PlayerInput::moveNone:
225             break;
226         case IO::PlayerInput::pointUp:
227             this→state→pointUp(*this);
228             break;
229         case IO::PlayerInput::pointDown:
230             this→state→pointDown(*this);
231             break;
232         case IO::PlayerInput::startShot:
233             this→state→startShot(*this);
234             break;
235         case IO::PlayerInput::endShot:
236             this→state→endShot(*this);
237             break;
238         case IO::PlayerInput::timeout1:
239             this→state→setTimeout(*this, 1);
240             break;
241         case IO::PlayerInput::timeout2:
242             this→state→setTimeout(*this, 2);
243             break;
244         case IO::PlayerInput::timeout3:
245             this→state→setTimeout(*this, 3);
246             break;
247         case IO::PlayerInput::timeout4:
248             this→state→setTimeout(*this, 4);
249             break;
250         case IO::PlayerInput::timeout5:
251             this→state→setTimeout(*this, 5);
252             break;
253         case IO::PlayerInput::positionSelected:
254             this→weapon→positionSelected(*this, pi.position);
255             break;
256         case IO::PlayerInput::aerialAttack:
257             this→state→aerialAttack(*this);
258             break;
259         case IO::PlayerInput::dynamite:
```



```
260             this→state→dynamite(*this);
261             break;
262         case IO::PlayerInput::baseballBat:
263             this→state→baseballBat(*this);
264             break;
265         case IO::PlayerInput::teleport:
266             this→state→teleport(*this);
267             break;
268         default:
269             break;
270     }
271 }
272
273 void Worms::Player::setState(Worm::StateID stateID) {
274     if (this→state ≡ nullptr ∨ this→state→getState() ≠ stateID) {
275         /* creates the right state type */
276         this→body→SetType(b2_dynamicBody);
277         switch (stateID) {
278             case Worm::StateID::Still:
279                 //                this->body->SetType(b2_staticBody);
280                 this→state = std::shared_ptr<State>(new Still());
281                 break;
282             case Worm::StateID::Walk:
283                 this→state = std::shared_ptr<State>(new Walk());
284                 break;
285             case Worm::StateID::StartJump:
286                 this→state = std::shared_ptr<State>(new StartJump());
287                 break;
288             case Worm::StateID::Jumping:
289                 this→state = std::shared_ptr<State>(new Jumping(this→getPositi
    on()));
290                 break;
291             case Worm::StateID::EndJump:
292                 this→state = std::shared_ptr<State>(new EndJump());
293                 break;
294             case Worm::StateID::StartBackFlip:
295                 this→state = std::shared_ptr<State>(new StartBackFlip());
296                 break;
297             case Worm::StateID::BackFlipping:
298                 this→state = std::shared_ptr<State>(new BackFlipping(this→getP
    osition()));
299                 break;
300             case Worm::StateID::EndBackFlip:
301                 this→state = std::shared_ptr<State>(new EndBackFlip());
302                 break;
303             case Worm::StateID::Falling:
304                 this→state = std::shared_ptr<State>(new Falling(this→getPositi
    on()));
305                 break;
306             case Worm::StateID::Land:
307                 this→state = std::shared_ptr<State>(new Land());
308                 break;
309             case Worm::StateID::Batting:
310                 this→state = std::shared_ptr<State>(new Batting());
311                 break;
312             case Worm::StateID::Teleporting:
313                 this→state = std::shared_ptr<State>(new Teleporting(this→telep
    ortPosition));
314                 break;
315             case Worm::StateID::Teleported:
316                 this→state = std::shared_ptr<State>(new Teleported());
317                 break;
318             case Worm::StateID::Hit:
319                 this→state = std::shared_ptr<State>(new Hit());
320                 break;
321             case Worm::StateID::Die:
```

```
322                 this→state = std::shared_ptr<State>(new Die());
323                 break;
324             case Worm::StateID::Drowning:
325                 this→state = std::shared_ptr<State>(new Drowning());
326                 break;
327             case Worm::StateID::Dead:
328                 this→state = std::shared_ptr<State>(new Dead());
329                 this→body→SetType(b2_staticBody);
330                 break;
331             case Worm::StateID::Sliding:
332                 this→notify(*this, Event::WormFalling);
333                 this→state = std::shared_ptr<State>(new Sliding());
334                 break;
335         }
336     }
337 }
338
339 std::list<Worms::Bullet> Worms::Player::getBullets() {
340     return std::move(this→bullets);
341 }
342
343 void Worms::Player::acknowledgeDamage(Config::Bullet::DamageInfo damageInfo,
344                                       Math::Point<float> epicenter) {
345     if (this→getStateId() ≠ Worm::StateID::Dead) {
346         double distanceToEpicenter = this→getPosition().distance(epicenter);
347         if (distanceToEpicenter ≤ damageInfo.radius) {
348             this→body→SetType(b2_dynamicBody);
349             double inflictedDamage =
350                 (1.0f − (distanceToEpicenter / (damageInfo.radius * 1.01f))) * d
    amageInfo.damage;
351             this→health −= inflictedDamage;
352
353             Math::Point<float> positionToEpicenter = this→getPosition() − epice
    nter;
354             float xImpactDirection = (positionToEpicenter.x > 0) − (positionToEp
    icenter.x < 0);
355             float yImpactDirection = (positionToEpicenter.y > 0) − (positionToEp
    icenter.y < 0);
356             float32 mass = this→body→GetMass();
357             b2Vec2 impulses = {
358                 mass * float32(inflictedDamage) * xImpactDirection * damageInfo.
    impulseDampingRatio,
359                 mass * float32(inflictedDamage) * yImpactDirection *
360                     damageInfo.impulseDampingRatio};
361             b2Vec2 position = this→body→GetWorldCenter();
362             this→body→ApplyLinearImpulse(impulses, position, true);
363             this→notify(*this, Event::Hit);
364             this→setState(Worm::StateID::Hit);
365             this→health = (this→health < 0) ? 0 : this→health;
366         }
367     }
368 }
369
370 void Worms::Player::acknowledgeDamage(const Config::P2PWeapon &info,
371                                       Math::Point<float> shooterPosition,
372                                       Worm::Direction shooterDirection) {
373     if (this→getStateId() ≠ Worm::StateID::Dead) {
374         if ((shooterDirection ≡ Worm::Direction::right ∧
375              this→getPosition().x − shooterPosition.x > 0) ∨
376             (shooterDirection ≡ Worm::Direction::left ∧
377              this→getPosition().x − shooterPosition.x < 0)) {
378             double distanceToTheWeapon = this→getPosition().distance(info.posit
    ion);
379             if (distanceToTheWeapon ≤ info.dmgInfo.radius ∧ distanceToTheWeapon
    > 0) {
380                 this→body→SetType(b2_dynamicBody);
```



```
381                 this→health −= info.dmgInfo.damage;
382                 this→health = (this→health < 0) ? 0 : this→health;
383
384                 float32 mass = this→body→GetMass();
385                 Math::Point<float> direction{0, 0};
386                 direction.x = info.dmgInfo.radius * cos(info.angle * PI / 180.0f
    );
387                 direction.y = info.dmgInfo.radius * sin(info.angle * PI / 180.0f
    );
388                 Math::Point<float> positionToShooter = this→getPosition() − sho
    oterPosition;
389                 float xImpactDirection = (positionToShooter.x > 0) − (positionTo
    Shooter.x < 0);
390                 float yImpactDirection = (direction.y > 0) − (direction.y < 0);
391                 b2Vec2 impulses = {mass * float32(info.dmgInfo.damage) * directi
    on.x *
392                                                 xImpactDirection * info.dmgInfo.impulseDa
    mpingRatio,
393                                               mass * float32(info.dmgInfo.damage) * directi
    on.y *
394                                                 yImpactDirection * info.dmgInfo.impulseDa
    mpingRatio};
395
396                 this→body→ApplyLinearImpulse(impulses, this→body→GetWorldCent
    er(), true);
397                 this→notify(*this, Event::Hit);
398                 this→setState(Worm::StateID::Hit);
399             }
400         }
401     }
402 }
403
404 float Worms::Player::getWeaponAngle() const {
405     return this→weapon→getAngle();
406 }
407
408 const Worm::WeaponID &Worms::Player::getWeaponID() const {
409     return this→weapon→getWeaponID();
410 }
411
412 void Worms::Player::setWeapon(const Worm::WeaponID &id) {
413     // keep the last angle
414     float lastAngle = this→weapon→getAngle();
415     this→weapon = this→team→getWeapon(id);
416     this→weapon→setAngle(lastAngle);
417     this→isP2PWeapon = this→weapon→isP2PWeapon();
418 }
419
420 void Worms::Player::increaseWeaponAngle() {
421     this→weapon→increaseAngle();
422 }
423
424 void Worms::Player::decreaseWeaponAngle() {
425     this→weapon→decreaseAngle();
426 }
427
428 void Worms::Player::startShot() {
429     this→weapon→startShot(this);
430 }
431
432 void Worms::Player::endShot() {
433     if (this→weapon→getWeaponID() ≠ Worm::WeaponID::WTeleport ∧
434         this→weapon→getWeaponID() ≠ Worm::WeaponID::WAerial ∧
435             this→weapon→getWeaponID() ≠ Worm::WeaponID::WNone) {
436         if (¬this→isP2PWeapon) {
437             Math::Point<float> position = this→getPosition();
```

```
438             float safeNonContactDistance = sqrt((PLAYER_WIDTH / 2) * (PLAYER_WID
    TH / 2) +
439                                                 (PLAYER_HEIGHT / 2) * (PLAYER_HE
    IGHT / 2)) + 0.1;
440             BulletInfo info = this→weapon→getBulletInfo();
441             info.point = position;
442             info.safeNonContactDistance = safeNonContactDistance;
443             if (this→direction ≡ Worm::Direction::right) {
444                 if (info.angle < 0.0f) {
445                     info.angle += 360.0f;
446                 }
447             } else {
448                 info.angle = 180.0f − info.angle;
449             }
450             this→bullets.emplace_back(info, this→physics, this→weapon→getWeap
    onID());
451             this→weapon→endShot();
452             this→notify(*this, Event::Shot);
453         } else {
454             this→setState(Worm::StateID::Batting);
455             this→notify(*this, Event::P2PWeaponUsed);
456         }
457         this→team→weaponUsed(this→getWeaponID());
458     }
459 }
460
461 void Worms::Player::endShot(std::list<Worms::Bullet> &bullets) {
462     this→bullets = std::move(bullets);
463     this→notify(*this, Event::Shot);
464     this→team→weaponUsed(this→getWeaponID());
465 }
466
467 void Worms::Player::setTeamID(uint8_t team) {
468     this→teamID = team;
469 }
470
471 void Worms::Player::increaseHealth(float extraPoints) {
472 //     this->health += (percentage / 100.0f) * this->health; //    25% more
473     this→health += extraPoints; //    25 points more
474 }
475
476 uint8_t Worms::Player::getTeam() const {
477     return this→teamID;
478 }
479
480 void Worms::Player::setId(uint8_t id) {
481     this→id = id;
482 }
483
484 uint8_t Worms::Player::getId() const {
485     return this→id;
486 }
487
488 void Worms::Player::setWeaponTimeout(uint8_t time) {
489     this→weapon→setTimeout(time);
490 }
491
492 void Worms::Player::landDamage(float yDistance) {
493     if (yDistance > CONFIG.getSafeFallDistance()) {
494         this→health −=
495             (yDistance > CONFIG.getMaxFallDamage()) ? CONFIG.getMaxFallDamage()
    : yDistance;
496         this→health = (this→health < 0) ? 0 : this→health;
497         if (this→health > 0) {
498             this→notify(*this, Event::DamageOnLanding);
499         }
```



```
500     }
501 }
502
503 /**
504  * @brief Creates a player's body with the given type.
505  *
506  * @param type Body type.
507  * @return Created body.
508  */
509 b2Body *Worms::Player::createBody(b2BodyType type) {
510     /* the players consists of a rectangle as the upper part of the body and a c
    icle for the
511      * bottom */
512     b2BodyDef bodyDef;
513     bodyDef.type = type;
514     bodyDef.position.Set(0.0f, 0.0f);
515     bodyDef.fixedRotation = true;
516     b2Body *new_body = this→physics.createBody(bodyDef);
517
518     b2PolygonShape shape;
519     shape.SetAsBox(PLAYER_WIDTH / 2, PLAYER_HEIGHT / 4, b2Vec2{0.0f, PLAYER_HEIG
    HT / 4}, 0.0f);
520
521     /* creates the upper square */
522     b2FixtureDef fixture;
523     fixture.shape = &shape;
524     fixture.density = 1.0f;
525     fixture.restitution = 0.1f;
526     fixture.friction = 1.0f;
527     new_body→CreateFixture(&fixture);
528
529     /* creates the bottom circle */
530     b2CircleShape bottom;
531     bottom.m_radius = PLAYER_HEIGHT / 4;
532     bottom.m_p.Set(0.0f, −PLAYER_HEIGHT / 4);
533     fixture.shape = &bottom;
534     new_body→CreateFixture(&fixture);
535
536     new_body→SetUserData(this);
537     return new_body;
538 }
539
540 std::list<Worms::Bullet> Worms::Player::onExplode(const Bullet &b, Physics &phys
    ics) {
541     return std::move(this→weapon→onExplode(b, physics));
542 }
543
544 void Worms::Player::reset() {
545     this→weapon→endShot();
546     /*
547      * If the weapon has no more ammunition, returns weaponNone
548      */
549     this→setWeapon(this→getWeaponID());
550     this→bullets.erase(this→bullets.begin(), this→bullets.end());
551 }
552
553 Worms::Physics &Worms::Player::getPhysics() {
554     return this→physics;
555 }
556
557 const std::shared_ptr<Worms::Weapon> Worms::Player::getWeapon() const {
558     return this→weapon;
559 }
560
561 void Worms::Player::setTeam(Worms::Team *team) {
562     this→team = team;
```

jun 26, 18 17:16 **Player.cpp** Page 10/10

```
563 }
564
565 Worms::Player::Player(Worms::Player ∧player) noexcept: PhysicsEntity(std::move(
    player)), physics(player.physics), waterLevel(player.waterLevel) {
566
567     this→body = player.body;
568     this→body_kinematic = player.body_kinematic;
569     this→footSensor = player.footSensor;
570
571     this→state = player.state;
572     this→weapon = player.weapon;
573     this→team = player.team;
574     this→id = player.id;
575     this→bullets = std::move(player.bullets);
576
577     player.body = nullptr;
578     player.body_kinematic = nullptr;
579     player.footSensor = nullptr;
580     player.state = nullptr;
581     player.weapon = nullptr;
582     player.team = 0;
583     player.id = 0;
584 }
585
586 void Worms::Player::die() {
587     this→setState(Worm::StateID::Die);
588     this→health = 0;
589     this→dyingDisconnected = true;
590     this→notify(*this, Event::DyingDueToDisconnection);
591 }
```

jun 26, 18 17:16 **Physics.h** Page 1/1

```
1  /*
2   *  Created by Federico Manuel Gomez Peter.
3   *  date: 18/05/18
4   */
5
6  #ifndef __Physics_H__
7  #define __Physics_H__
8
9  #include "Box2D/Box2D.h"
10
11 #include <memory.h>
12 #include "ContactEventListener.h"
13
14 namespace Worms {
15 class Physics {
16    public:
17     Physics(b2Vec2 gravity, float timeStep);
18     ~Physics() = default;
19     void update(float dt);
20     b2Body *createBody(b2BodyDef &bodyDef);
21
22    private:
23     float timeStep;
24     float accumTime{0.0f};
25     b2Vec2 gravity;
26     b2World world;
27     std::shared_ptr<ContactEventListener> contactEventListener;
28     int32 vIterations{6};
29     int32 pIterations{2};
30 };
31 }  // namespace Worms
32
33 #endif  //__Physics_H__
```

jun 26, 18 17:16 **PhysicsEntity.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 26/05/18
 */

#ifndef PHYSICS_ENTITY_H_
#define PHYSICS_ENTITY_H_

#include "Box2D/Box2D.h"

#include "Subject.h"

namespace Worms {
enum EntityID { EtWorm, EtBullet, Sensor, EtGirder };
class PhysicsEntity : public Subject {
   public:
    explicit PhysicsEntity(EntityID id);
    PhysicsEntity(PhysicsEntity ∧other);
    PhysicsEntity(PhysicsEntity &copy) = delete;

    virtual EntityID getEntityId();
    virtual void startContact(Worms::PhysicsEntity *physicsEntity) {}
    virtual void startContact(Worms::PhysicsEntity *physicsEntity, b2Contact &co
ntact) {}
    virtual void endContact(Worms::PhysicsEntity *physicsEntity) {}
    virtual void endContact(Worms::PhysicsEntity *physicsEntity, b2Contact &cont
act) {}
    virtual void contactWith(PhysicsEntity &physicsEntity, b2Contact &contact) {
}

   protected:
    EntityID id;
    bool handlingContact{false};
};
}  // namespace Worms

#endif
```

jun 26, 18 17:16 **PhysicsEntity.cpp** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 26/05/18
 */

#include "PhysicsEntity.h"

Worms::PhysicsEntity::PhysicsEntity(Worms::EntityID id) : id(id) {}

Worms::EntityID Worms::PhysicsEntity::getEntityId() {
    return this→id;
}

Worms::PhysicsEntity::PhysicsEntity(Worms::PhysicsEntity ∧other){
    this→id = other.id;
    this→handlingContact = other.handlingContact;

    other.handlingContact = false;
}
```

jun 26, 18 17:16 **Physics.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 18/05/18
 */

#include "Physics.h"

Worms::Physics::Physics(b2Vec2 gravity, float timeStep)
    : timeStep(timeStep),
      gravity(gravity),
      world(this→gravity),
      contactEventListener(new ContactEventListener) {
    this→world.SetContactListener(this→contactEventListener.get());
}

/**
 * @brief Updates the physics engine.
 *
 * @param dt Seconds elapsed since last call to this function.
 */
void Worms::Physics::update(float dt) {
    this→accumTime += dt;

    /* updates the physics engine */
    for (int i = 0; i < 5 ∧ this→accumTime > this→timeStep; i++) {
        this→world.Step(this→timeStep, this→vIterations, this→pIterations);
        this→accumTime -= this→timeStep;
    }
}

/**
 * @brief Creates a new physical body.
 *
 * @param bodyDef Body definition.
 * @return new body.
 */
b2Body* Worms::Physics::createBody(b2BodyDef& bodyDef) {
    return this→world.CreateBody(&bodyDef);
}
```

jun 26, 18 17:16 **main.cpp** Page 1/2

```cpp
/*
 * Created by Federico Manuel Gomez Peter
 * Date: 02/05/2018.
 */

#include <signal.h>
#include <unistd.h>
#include <cstdlib>
#include <iostream>
#include <string>
#include <thread>
#include <vector>

#include "CommunicationSocket.h"
#include "Game.h"
#include "ServerSocket.h"
#include "GameLobby.h"

static volatile bool quit = false;

/**
 * @brief Signal handler.
 *
 * @param _ unused.
 */
static void _signal_handler(int _) {
    quit = true;
}

///**
// * @brief Thread handler that signals the Game to exit.
// *
// * @param game
// */
//static void _exit_handler(Worms::Game &game) {
//    while (!quit) {
//        usleep(100000);
//    }
//    game.exit();
//}

int main(int argc, const char *argv[]) {
    if (argc ≠ 2) {
        std::cout << "Usage: ./server PORT" << std::endl;
        return EXIT_FAILURE;
    }

    try {
        /* sets a signal handler to exit the program gracefully */
        signal(SIGINT, _signal_handler);
        signal(SIGTERM, _signal_handler);

        std::string port(argv[1]);
        Worms::GameLobby gameLobby{port};

        gameLobby.start();
        char quit{0};
        while (quit ≠ 'q'){
            std::cin >> quit;
        }

        gameLobby.stop();
        gameLobby.join();

    } catch (std::exception &e) {
        std::cerr << "In main()" << std::endl;
```

jun 26, 18 17:16 **main.cpp** Page 2/2

```
67          std::cerr << e.what() << std::endl;
68          return 1;
69      } catch (...) {
70          std::cerr << "Unkown error in main thread" << std::endl;
71          return 1;
72      }
73
74      return EXIT_SUCCESS;
75  }
```

jun 26, 18 17:16 **LobbyJoiner.h** Page 1/1

```
1   //
2   // Created by rodrigo on 19/06/18.
3   //
4
5   #ifndef INC_4_WORMS_LOBBYJOINER_H
6   #define INC_4_WORMS_LOBBYJOINER_H
7
8
9   #include <GameStateMsg.h>
10  #include <Stream.h>
11  #include "Thread.h"
12  #include "Lobbies.h"
13
14  namespace Worms {
15      class LobbyJoiner : public Thread {
16      public:
17          explicit LobbyJoiner(Worms::Lobbies &lobbies, IO::Stream<IO::ServerInter
    nalMsg> &serverInput);
18
19          void run() override;
20          void stop() override;
21
22      private:
23          std::list<Lobby> &lobbies;
24          IO::Stream<IO::ServerInternalMsg> &serverInput;
25          bool finished{false};
26
27          void handleServerInput(IO::ServerInternalMsg &msg);
28
29          void killLobbies();
30      };
31  }
32
33
34  #endif //INC_4_WORMS_LOBBYJOINER_H
```

```
1  //
2  // Created by rodrigo on 19/06/18.
3  //
4
5  #include <GameStateMsg.h>
6  #include <iostream>
7  #include "LobbyJoiner.h"
8
9  Worms::LobbyJoiner::LobbyJoiner(Worms::Lobbies &lobbies, IO::Stream<IO::ServerIn
   ternalMsg> &serverInput) :
10         lobbies(lobbies.getLobbies()),
11         serverInput(serverInput) {
12 }
13
14 void Worms::LobbyJoiner::run() {
15     try{
16         while (¬this→finished) {
17             IO::ServerInternalMsg msg;
18             if (this→serverInput.pop(msg)) {
19                 this→handleServerInput(msg);
20             }
21         }
22     } catch (std::exception &e){
23         if(¬this→finished){
24             std::cerr << "In LobbyJoiner::run()" << std::endl;
25             std::cerr << e.what() << std::endl;
26         }
27     } catch (...){
28         std::cerr << "Unknown error in LobbyJoiner::run()" << std::endl;
29     }
30
31     this→killLobbies();
32 }
33
34 void Worms::LobbyJoiner::stop() {
35     this→finished = true;
36 }
37
38 void Worms::LobbyJoiner::handleServerInput(IO::ServerInternalMsg &msg) {
39     switch (msg.action) {
40         case IO::ServerInternalAction::lobbyFinished: {
41             std::list<Lobby>::iterator lobbyIt;
42             lobbyIt = this→lobbies.begin();
43             while (lobbyIt ≠ this→lobbies.end()) {
44                 if (lobbyIt→itsOver()) {
45                     lobbyIt→join();
46                     lobbyIt = this→lobbies.erase(lobbyIt);
47                 } else {
48                     lobbyIt++;
49                 }
50             }
51             break;
52         }
53         case IO::ServerInternalAction::quit: {
54             this→finished = true;
55             break;
56         }
57     }
58 }
59
60 void Worms::LobbyJoiner::killLobbies(){
61     for (auto &lobby: this→lobbies){
62         if (lobby.started()) {
63             lobby.stop();
64             lobby.join();
65         }
```



```
66         }
67         this→lobbies.erase(this→lobbies.begin(), this→lobbies.end());
68 }
```

jun 26, 18 17:16 **Lobby.h** Page 1/1

```
1  //
2  // Created by rodrigo on 16/06/18.
3  //
4
5  #ifndef INC_4_WORMS_LOBBY_H
6  #define INC_4_WORMS_LOBBY_H
7
8
9  #include <stdint-gcc.h>
10 #include <string>
11 #include <vector>
12 #include <mutex>
13 #include <GameStateMsg.h>
14
15 #include "CommunicationSocket.h"
16 #include "Subject.h"
17 #include "Thread.h"
18
19 namespace Worms {
20     class Lobby : public Thread, public Subject {
21     public:
22         Lobby(int playerID, std::uint8_t id, std::vector<Observer *> obs, const
   IO::LevelData level,
23               const IO::LevelInfo levelInfo);
24         Lobby(Lobby &&other) noexcept;
25         Lobby(Lobby &copy) = delete;
26
27         void run() override;
28         void stop() override;
29         bool itsOver();
30
31         void joinGame(int playerID);
32         const IO::LevelInfo & getLevelInfo() const;
33         std::uint8_t getActualPlayers() const;
34         const std::vector<int> &getPlayerIDs() const;
35         std::uint8_t getID() const;
36         void addLobbyObserver(Observer *lobbyObserver);
37         bool started();
38         void addPlayerSocket(CommunicationSocket &&player);
39
40     private:
41         std::mutex mutex;
42         const std::uint8_t id;
43         std::uint8_t actualPlayers{0};
44         std::vector<int> playerIDs;
45         std::vector<CommunicationSocket> players;
46         std::vector<Observer *> obs;
47         const IO::LevelData level;
48         const IO::LevelInfo levelInfo;
49
50         bool finished{false};
51         bool gameStarted{false};
52     };
53 }
54
55
56 #endif //INC_4_WORMS_LOBBY_H
```

jun 26, 18 17:16 **Lobby.cpp** Page 1/2

```
1  //
2  // Created by rodrigo on 16/06/18.
3  //
4
5  #include <iostream>
6  #include <string>
7
8  #include "Lobby.h"
9  #include "Stage.h"
10 #include "Game.h"
11
12
13 /** Copio por Â¿posible race condition?
14  */
15 Worms::Lobby::Lobby(int playerID, std::uint8_t id, std::vector<Observer *> obs,
   const IO::LevelData level,
16                     const IO::LevelInfo levelInfo) :
17         id(id),
18         level(level),
19         levelInfo(levelInfo) {
20     for (auto *lobbyObserver : obs) {
21         this→obs.emplace_back(lobbyObserver);
22         this→addObserver(lobbyObserver);
23     }
24     this→joinGame(playerID);
25 }
26
27 void Worms::Lobby::joinGame(int playerID) {
28 //    std::lock_guard<std::mutex> lock(this->mutex);
29     this→playerIDs.emplace_back(playerID);
30     this→actualPlayers++;
31     this→notify(*this, Event::NewPlayer);
32     if (this→actualPlayers ≡ levelInfo.playersQuantity) {
33         this→notify(*this, Event::StartGame);
34         std::uint8_t i{0};
35         for (auto *obs : this→obs) {
36             if (i ≠ 0) {
37                 this→removeObserver(obs);
38             }
39             i++;
40         }
41         this→gameStarted = true;
42     }
43 }
44
45 const IO::LevelInfo & Worms::Lobby::getLevelInfo() const{
46     return this→levelInfo;
47 }
48
49 std::uint8_t Worms::Lobby::getActualPlayers() const{
50     return this→actualPlayers;
51 }
52
53 const std::vector<int> &Worms::Lobby::getPlayerIDs() const{
54     return this→playerIDs;
55 }
56
57 std::uint8_t Worms::Lobby::getID() const{
58     return this→id;
59 }
60
61 void Worms::Lobby::addPlayerSocket(CommunicationSocket &&player) {
62     this→players.emplace_back(std::move(player));
63 }
64
65 Worms::Lobby::Lobby(Worms::Lobby &&other) noexcept :
```

jun 26, 18 17:16 **Lobby.cpp** Page 2/2

```
66          id(other.id),
67          level(other.level),
68          levelInfo(other.levelInfo) {
69      if (this ≠ &other){
70          this→actualPlayers = other.actualPlayers;
71          this→playerIDs = std::move(other.playerIDs);
72          this→players = std::move(other.players);
73      }
74  }
75
76  void Worms::Lobby::run() {
77      try {
78          while (¬this→finished) {
79              if (this→gameStarted) {
80                  for (std::uint8_t i = 0; i < levelInfo.playersQuantity; i++) {
81                      char buffer[1];
82                      buffer[0] = i;
83                      this→players[i].send(buffer, sizeof(buffer));
84                  }
85                  Worms::Game game{Worms::Stage::fromFile(this→level.levelPath),
    this→players};
86                  game.start();
87                  this→notify(*this, Event::EndGame);
88                  this→gameStarted = false;
89                  this→finished = true;
90              }
91          }
92      } catch (std::exception &e){
93          if (¬this→finished){
94              std::cerr << "In Lobby::run()" << std::endl;
95              std::cerr << e.what() << std::endl;
96          }
97      } catch (...){
98          std::cerr << "Unkown error in Lobby::run()" << std::endl;
99      }
100 }
101
102 void Worms::Lobby::stop() {
103     this→finished = true;
104     for(auto &player: this→players){
105         player.shutdown();
106     }
107 }
108
109 bool Worms::Lobby::itsOver() {
110     return this→finished;
111 }
112
113 void Worms::Lobby::addLobbyObserver(Observer *lobbyObserver) {
114     this→obs.emplace_back(lobbyObserver);
115     this→addObserver(lobbyObserver);
116 }
117
118 bool Worms::Lobby::started() {
119     return this→gameStarted;
120 }
```

jun 26, 18 17:16 **Lobbies.h** Page 1/1

```
1   //
2   // Created by rodrigo on 15/06/18.
3   //
4
5   #ifndef INC_4_WORMS_LOBBIES_H
6   #define INC_4_WORMS_LOBBIES_H
7
8
9   #include <list>
10  #include <mutex>
11
12  #include "GamesGetter.h"
13  #include "Lobby.h"
14  #include "Observer.h"
15
16  namespace Worms {
17      class Lobbies {
18      public:
19          explicit Lobbies(const std::vector<IO::LevelData> &levels);
20          ~Lobbies();
21          void configure();
22          void createGame(int playerID, std::vector<Observer *> lobbyObservers, ui
    nt8_t levelSelected);
23          void getGames(GamesGetter &getter);
24          void joinGame(int gameID, int playerID, Observer *lobbyObserver);
25          const std::vector<IO::LevelInfo> &getLevels();
26          const IO::LevelData & getLevelData(uint8_t levelSelected);
27          std::list<Lobby> &getLobbies();
28
29      private:
30          std::mutex mutex;
31          std::list<Lobby> lobbies;
32          uint8_t idLobby{0};
33          const std::vector<IO::LevelData> &levels;
34          std::vector<IO::LevelInfo> levelsInfo;
35      };
36  }
37
38
39  #endif //INC_4_WORMS_LOBBIES_H
```

jun 26, 18 17:16 **Lobbies.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 15/06/18.
//

#include <iostream>
#include <yaml-cpp/yaml.h>
#include "GamesGetter.h"
#include "Lobbies.h"

void Worms::Lobbies::createGame(int playerID, std::vector<Observer *> lobbyObservers, uint8_t levelSelected) {
    std::lock_guard<std::mutex> lock(this→mutex);
    this→lobbies.emplace_back(playerID, this→idLobby, lobbyObservers, this→levels[levelSelected], this→levelsInfo[levelSelected]);
    this→idLobby++;
}

void Worms::Lobbies::getGames(GamesGetter &getter) {
    std::lock_guard<std::mutex> lock(this→mutex);
    getter(this→lobbies);
}

void Worms::Lobbies::joinGame(int gameID, int playerID, Observer *lobbyObserver) {
    std::lock_guard<std::mutex> lock(this→mutex);
    auto it = this→lobbies.begin();
    while ((*it).getID() ≠ gameID ∧ it ≠ this→lobbies.end()){
        it++;
    }
    (*it).addLobbyObserver(lobbyObserver);
    (*it).joinGame(playerID);
}

std::list<Worms::Lobby> &Worms::Lobbies::getLobbies() {
    return this→lobbies;
}

Worms::Lobbies::Lobbies(const std::vector<IO::LevelData> &levels) :
        levels(levels) {}

const std::vector<IO::LevelInfo> &Worms::Lobbies::getLevels() {
    return this→levelsInfo;
}

const IO::LevelData & Worms::Lobbies::getLevelData(uint8_t levelSelected) {
    return this→levels[levelSelected];
}

void Worms::Lobbies::configure(){
    uint8_t id{0};
    for (auto &level : this→levels) {
        YAML::Node data = YAML::LoadFile(level.levelPath);
        std::string name = data["name"].as<std::string>();
        uint8_t playersQuantity = static_cast<uint8_t>(data["numPlayers"].as<int>());
        IO::LevelInfo levelInfo{id, name, playersQuantity};
        this→levelsInfo.emplace_back(levelInfo);
        id++;
    }
}

Worms::Lobbies::~Lobbies(){}

```

jun 26, 18 17:16 **Girder.h** Page 1/1

```cpp
#ifndef GIRDER_H_
#define GIRDER_H_

#include "Physics.h"
#include "PhysicsEntity.h"
#include "Stage.h"

namespace Worms {
class Girder : public PhysicsEntity {
   public:
    const float angle;

    Girder(const Worms::GirderData &data, Physics &physics);
    Girder(Girder &copy) = delete;
    Girder(Girder ∧other) noexcept;
    ~Girder() = default;
};
}  // namespace Worms

#endif
```

jun 26, 18 17:16 **Girder.cpp** Page 1/1

```cpp
#include "Girder.h"

Worms::Girder::Girder(const Worms::GirderData &data, Worms::Physics &physics)
    : PhysicsEntity(EntityID::EtGirder), angle(data.angle) {
    b2PolygonShape poly;

    b2BodyDef bdef;
    bdef.type = b2_staticBody;
    bdef.position.Set(0.0f, 0.0f);
    b2Body *staticBody = physics.createBody(bdef);

    b2FixtureDef fixture;
    fixture.density = 1;
    fixture.shape = &poly;

    poly.SetAsBox(data.length / 2, data.height / 2, b2Vec2(data.pos.x, data.pos.y),
                  data.angle * (PI / 180.0f));
    staticBody→CreateFixture(&fixture);

    staticBody→SetUserData(this);
}

Worms::Girder::Girder(Worms::Girder ∧other) noexcept :
        PhysicsEntity(other.id), angle(other.angle){
    this→handlingContact = other.handlingContact;
    this→numObservers = other.numObservers;
    this→observers = std::move(other.observers);
}
```

jun 26, 18 17:16 **GameTurn.h** Page 1/1

```cpp
//
// Created by rodrigo on 10/06/18.
//

#ifndef INC_4_WORMS_GAMETURN_H
#define INC_4_WORMS_GAMETURN_H

#include <memory>
#include "GameStates/GameTurnState.h"
#include "Subject.h"

namespace Worms {
enum class GameTurnStateID { StartTurn, PlayerShot, ImpactOnCourse };
class Game;
class GameTurn : public Subject {
   public:
    GameTurn(Observer &game);
    ~GameTurn() override = default;

    void playerShot(Worm::WeaponID weaponID);
    void endTurn();
    void wormHit(uint8_t wormId);
    void explosion();
    void wormEndHit(uint8_t wormId);
    void wormDrowning(uint8_t wormId);
    void wormDrowned(uint8_t wormId);
    void restart();
    void update(float dt);
    void wormFalling(uint8_t wormId);
    void wormLanded(uint8_t wormId);
    void wormDead();
    void wormDying();
    void playerDisconnected(uint8_t wormId);
    void playerDisconnectedDead(uint8_t wormId);

private:
    std::shared_ptr<GameTurnState> state{nullptr};
    Observer &game;
    GameTurnStateID stateID;
    bool newState{false};
    uint8_t bulletFragments{1};
};
}

#endif  // INC_4_WORMS_GAMETURN_H
```

```
1  //
2  // Created by rodrigo on 10/06/18.
3  //
4
5  #include "GameTurn.h"
6  #include "Config/Config.h"
7  #include "GameStateMsg.h"
8  #include "GameStates/ImpactOnCourse.h"
9  #include "GameStates/PlayerShot.h"
10 #include "GameStates/StartTurn.h"
11
12 Worms::GameTurn::GameTurn(Observer &game) : game(game) {
13     this→state = std::shared_ptr<GameTurnState>(new StartTurn());
14     this→state→addObserver(&this→game);
15 }
16
17 void Worms::GameTurn::playerShot(Worm::WeaponID weaponID) {
18     this→stateID = GameTurnStateID::PlayerShot;
19     this→newState = true;
20
21     switch (weaponID) {
22         case Worm::WeaponID::WMortar:
23             this→bulletFragments = ::Game::Config::getInstance().getMortarFragm
   entQuantity();
24             break;
25         case Worm::WeaponID::WCluster:
26             this→bulletFragments = ::Game::Config::getInstance().getClusterFrag
   mentQuantity();
27             break;
28         case Worm::WeaponID::WAerial:
29             this→bulletFragments = ::Game::Config::getInstance().getAerialAttac
   kMissileQuantity();
30             break;
31         default:
32             break;
33     }
34 }
35
36 void Worms::GameTurn::endTurn() {
37     this→state→endTurn(*this);
38 }
39
40 void Worms::GameTurn::wormHit(uint8_t wormId) {
41     this→state→wormHit(*this, wormId);
42 }
43
44 void Worms::GameTurn::explosion() {
45     if (this→stateID ≠ GameTurnStateID::ImpactOnCourse) {
46         this→stateID = GameTurnStateID::ImpactOnCourse;
47         this→state = std::shared_ptr<GameTurnState>(new ImpactOnCourse(this→bu
   lletFragments));
48         this→state→addObserver(&this→game);
49     }
50     this→state→explosion();
51 }
52
53 void Worms::GameTurn::wormEndHit(uint8_t wormId) {
54     this→state→wormEndHit(*this, wormId);
55 }
56
57 void Worms::GameTurn::wormDrowning(uint8_t wormId) {
58     this→state→wormDrowning(*this, wormId);
59 }
60
61 void Worms::GameTurn::wormDrowned(uint8_t wormId) {
62     this→state→wormDrowned(*this, wormId);
```



```
63 }
64
65 void Worms::GameTurn::restart() {
66     this→stateID = GameTurnStateID::StartTurn;
67     this→newState = true;
68     this→bulletFragments = 1;
69 }
70
71 void Worms::GameTurn::update(float dt) {
72     if (this→newState) {
73         switch (this→stateID) {
74             case GameTurnStateID::StartTurn:
75                 this→state = std::shared_ptr<GameTurnState>(new StartTurn());
76                 break;
77             case GameTurnStateID::PlayerShot:
78                 this→state = std::shared_ptr<GameTurnState>(new PlayerShot());
79                 break;
80             case GameTurnStateID::ImpactOnCourse:
81                 break;
82         }
83         this→state→addObserver(&this→game);
84         this→newState = false;
85     }
86     this→state→update(dt);
87 }
88
89 void Worms::GameTurn::wormFalling(uint8_t wormId) {
90     this→state→wormFalling(wormId);
91 }
92
93 void Worms::GameTurn::wormLanded(uint8_t wormId) {
94     this→state→wormLanded(wormId);
95 }
96
97 void Worms::GameTurn::wormDead() {
98     this→state→wormDead();
99 }
100
101 void Worms::GameTurn::wormDying() {
102     this→state→wormDying();
103 }
104
105 void Worms::GameTurn::playerDisconnected(uint8_t wormId) {
106     this→state→wormDisconnectedDying(wormId);
107 }
108
109 void Worms::GameTurn::playerDisconnectedDead(uint8_t wormId) {
110     this→state→wormDisconnectedDead(wormId);
111 }
```

jun 26, 18 17:16 **GameTeams.h** Page 1/1

```
//
// Created by rodrigo on 3/06/18.
//

#ifndef INC_4_WORMS_TEAMS_H
#define INC_4_WORMS_TEAMS_H

#include <vector>
#include "Player.h"
#include "Team.h"

namespace Worms {
class GameTeams {
   public:
    GameTeams() = default;
    ~GameTeams(){};
    void makeTeams(std::vector<Player> &players, uint8_t numTeams,
                   const std::map<Worm::WeaponID, std::int16_t> &ammoCounter);
    void checkAlive(std::vector<Player> &players);
    bool endTurn(std::vector<Player> &players);
    uint8_t getCurrentPlayerID();
    std::uint8_t getTeamQuantity() const;
    uint8_t getCurrentTeamID();
    Team &getCurrentTeam();
    std::uint8_t getWinner();
    std::vector<std::uint32_t> getTotalHealth(std::vector<Worms::Player> &players);
    void weaponUsed(const Worm::WeaponID weaponID);
    void serialize(IO::GameStateMsg &msg) const;

    void kill(uint8_t team, std::vector<Player> &players);

private:
    std::vector<Team> teams;
    std::vector<std::uint8_t> deadTeams;
    uint8_t currentTeam{0};
};
}

#endif  // INC_4_WORMS_TEAMS_H
```

jun 26, 18 17:16 **GameTeams.cpp** Page 1/2

```
//
// Created by rodrigo on 3/06/18.
//

#include "GameTeams.h"
#include <random>

void Worms::GameTeams::makeTeams(std::vector<Worms::Player> &players, uint8_t numTeams,
                                 const std::map<Worm::WeaponID, std::int16_t> &ammoCounter) {
    uint8_t numPlayers = players.size();

    this→teams.reserve(numTeams);
    std::vector<uint8_t> playersNum(numPlayers);
    for (uint8_t i = 0; i < numPlayers; i++) {
        playersNum[i] = i;
    }

    std::random_device rnd_device;
    std::mt19937 mersenne_engine(rnd_device());

    shuffle(playersNum.begin(), playersNum.end(), mersenne_engine);

    uint8_t maxTeamPlayers =
        (numPlayers % numTeams ≡ 0) ? numPlayers / numTeams : numPlayers / numTeams + 1;
    std::vector<uint8_t> numPlayersPerTeam(numTeams);
    for (uint8_t i = 0, nP = numPlayers, nT = numTeams; i < numPlayersPerTeam.size(); i++) {
        numPlayersPerTeam[i] = nP / nT;
        nP −= numPlayersPerTeam[i];
        nT−−;
    }
    std::vector<uint8_t> playerIDs;
    for (uint8_t i = 0, currentTeam = 0; i < numPlayers; i++) {
        //        this->teams[currentTeam].players.emplace_back(players[playersNum[i]].getId());
        playerIDs.emplace_back(players[playersNum[i]].getId());
        players[playersNum[i]].setTeamID(currentTeam);
        if (numPlayersPerTeam[currentTeam] < maxTeamPlayers) {
            players[playersNum[i]].increaseHealth(25.0f);
        }
        if (playerIDs.size() ≡ numPlayersPerTeam[currentTeam]) {
            this→teams.emplace_back(playerIDs, players, ammoCounter);
            playerIDs.clear();
            currentTeam++;
        }
    }
}

void Worms::GameTeams::checkAlive(std::vector<Worms::Player> &players) {
    std::uint8_t teamID{0};
    for (auto &team : this→teams) {
        team.checkAlive(players);
        if (¬team.isAlive() ∧ std::find(this→deadTeams.begin(), this→deadTeams.end(), teamID) ≡ this→deadTeams.end()) {
            this→deadTeams.emplace_back(teamID);
        }
        teamID++;
    }
}

bool Worms::GameTeams::endTurn(std::vector<Player> &players) {
    this→checkAlive(players);

```

jun 26, 18 17:16 **GameTeams.cpp** Page 2/2

```
61      do {
62          this→currentTeam = (this→currentTeam + 1) % this→teams.size();
63      } while (¬this→teams[this→currentTeam].isAlive());
64
65      this→teams[this→currentTeam].endTurn(players);
66
67      if (this→deadTeams.size() ≥ this→teams.size() − 1) {
68          return true;
69      } else {
70          return false;
71      }
72  }
73
74  std::vector<std::uint32_t> Worms::GameTeams::getTotalHealth(std::vector<Worms::P
    layer> &players) {
75      uint8_t i{0};
76      std::vector<std::uint32_t> teamHealths;
77      for (auto &team : this→teams) {
78          teamHealths.emplace_back(team.calculateTotalHealth(players));
79          i++;
80      }
81      return std::move(teamHealths);
82  }
83
84  uint8_t Worms::GameTeams::getCurrentPlayerID() {
85      return this→teams[this→currentTeam].getCurrentPlayerID();
86  }
87
88  uint8_t Worms::GameTeams::getCurrentTeamID() {
89      return this→currentTeam;
90  }
91
92  uint8_t Worms::GameTeams::getWinner() {
93      std::uint8_t winner{0};
94      for (auto &team : this→teams) {
95          if (team.isAlive()) {
96              return winner;
97          }
98          winner++;
99      }
100     return winner;
101 }
102
103 std::uint8_t Worms::GameTeams::getTeamQuantity() const {
104     return (std::uint8_t) this→teams.size();
105 }
106
107 Worms::Team &Worms::GameTeams::getCurrentTeam() {
108     return this→teams[this→currentTeam];
109 }
110
111 void Worms::GameTeams::weaponUsed(const Worm::WeaponID weaponID) {
112     this→teams[this→currentTeam].weaponUsed(weaponID);
113 }
114
115 void Worms::GameTeams::serialize(IO::GameStateMsg &msg) const {
116     this→teams[this→currentTeam].serialize(msg);
117 }
118
119 void Worms::GameTeams::kill(uint8_t team, std::vector<Worms::Player> &players) {
120     this→teams[team].kill(players);
121 }
```

jun 26, 18 17:16 **StartTurn.h** Page 1/1

```
1   //
2   // Created by rodrigo on 10/06/18.
3   //
4
5   #ifndef INC_4_WORMS_STARTTURN_H
6   #define INC_4_WORMS_STARTTURN_H
7
8   #include "../../../libs/Observer.h"
9   #include "GameTurnState.h"
10
11  namespace Worms {
12  class StartTurn : public GameTurnState {
13     public:
14      StartTurn();
15      ~StartTurn() = default;
16
17      void endTurn(GameTurn &gt) override;
18      void update(float dt) override;
19      void wormHit(GameTurn &gt, uint8_t wormId) override;
20      void wormEndHit(GameTurn &gt, uint8_t wormId) override;
21      void wormDrowning(GameTurn &gt, uint8_t wormId) override;
22      void wormDrowned(GameTurn &gt, uint8_t wormId) override;
23      void explosion() override;
24      void wormDisconnectedDying(uint8_t wormId) override;
25      void wormDisconnectedDead(uint8_t wormId) override;
26  };
27  }
28
29  #endif  // INC_4_WORMS_STARTTURN_H
```

jun 26, 18 17:16 **StartTurn.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 10/06/18.
//

#include <algorithm>

#include "StartTurn.h"

void Worms::StartTurn::endTurn(GameTurn &gt) {
    if (this→wormsFalling.size() ≡ 0 ∧ this→wormsDrowning.size() ≡ 0 ∧ ¬this→wormsDying ∧ this→wormsDisconnectedDying.size() ≡ 0) {
        this→notify(*this, Event::TurnEnded);
    }
}

void Worms::StartTurn::wormHit(GameTurn &gt, uint8_t wormId) {}

void Worms::StartTurn::wormEndHit(Worms::GameTurn &gt, uint8_t wormId) {}

void Worms::StartTurn::wormDrowning(Worms::GameTurn &gt, uint8_t wormId) {
    this→wormsDrowning.emplace_back(wormId);
    this→wormLanded(wormId);
}

void Worms::StartTurn::wormDrowned(Worms::GameTurn &gt, uint8_t wormId) {
    this→wormsDrowning.erase(
        std::remove(this→wormsDrowning.begin(), this→wormsDrowning.end(), wormId),
        this→wormsDrowning.end());
}

Worms::StartTurn::StartTurn() {}

void Worms::StartTurn::explosion() {}

void Worms::StartTurn::update(float dt) {}

void Worms::StartTurn::wormDisconnectedDying(uint8_t wormId) {
    this→wormsDisconnectedDying.emplace_back(wormId);
    if (this→wormToFollow ≠ this→wormsDisconnectedDying[0] ∧ this→wormsFalling.size() ≡ 0 ∧ this→wormsDrowning.size() ≡ 0) {
        this→wormToFollow = this→wormsDisconnectedDying[0];
        this→notify(*this, Event::NewWormToFollow);
    }
}

void Worms::StartTurn::wormDisconnectedDead(uint8_t wormId) {
    this→wormsDisconnectedDying.erase(
            std::remove(this→wormsDisconnectedDying.begin(), this→wormsDisconnectedDying.end(), wormId),
            this→wormsDisconnectedDying.end());
    if (this→wormToFollow ≡ wormId) {
        this→wormToFollow = this→wormsDisconnectedDying[0];
        this→notify(*this, Event::NewWormToFollow);
    }
}
```

jun 26, 18 17:16 **PlayerShot.h** Page 1/1

```cpp
//
// Created by rodrigo on 10/06/18.
//

#ifndef INC_4_WORMS_PLAYERSHOT_H
#define INC_4_WORMS_PLAYERSHOT_H

#include "../../../libs/Observer.h"
#include "GameTurnState.h"

namespace Worms {
class PlayerShot : public GameTurnState {
   public:
    PlayerShot();
    ~PlayerShot() = default;

    void endTurn(GameTurn &gt) override;
    void update(float dt) override;
    void wormHit(GameTurn &gt, uint8_t wormId) override;
    void wormEndHit(GameTurn &gt, uint8_t wormId) override;
    void wormDrowning(GameTurn &gt, uint8_t wormId) override;
    void wormDrowned(GameTurn &gt, uint8_t wormId) override;
    void explosion() override;
    void wormDisconnectedDying(uint8_t wormId) override;
    void wormDisconnectedDead(uint8_t wormId) override;
};
}

#endif  // INC_4_WORMS_PLAYERSHOT_H
```

jun 26, 18 17:16 **PlayerShot.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 10/06/18.
//

#include "PlayerShot.h"

void Worms::PlayerShot::endTurn(GameTurn &gt) {}

void Worms::PlayerShot::wormHit(GameTurn &gt, uint8_t wormId) {}

void Worms::PlayerShot::wormEndHit(Worms::GameTurn &gt, uint8_t wormId) {}

void Worms::PlayerShot::wormDrowning(Worms::GameTurn &gt, uint8_t wormId) {}

void Worms::PlayerShot::wormDrowned(Worms::GameTurn &gt, uint8_t wormId) {}

Worms::PlayerShot::PlayerShot() {}

void Worms::PlayerShot::explosion() {}

void Worms::PlayerShot::update(float dt) {}

void Worms::PlayerShot::wormDisconnectedDying(uint8_t wormId) {}

void Worms::PlayerShot::wormDisconnectedDead(uint8_t wormId) {}
```

jun 26, 18 17:16 **ImpactOnCourse.h** Page 1/1

```cpp
//
// Created by rodrigo on 10/06/18.
//

#ifndef INC_4_WORMS_IMPACTONCOURSE_H
#define INC_4_WORMS_IMPACTONCOURSE_H

#include <GameStateMsg.h>
#include <vector>
#include "../../../libs/Observer.h"
#include "GameTurnState.h"

namespace Worms {
class ImpactOnCourse : public GameTurnState {
   public:
    ImpactOnCourse(uint8_t bulletFragments);
    ~ImpactOnCourse() = default;

    void endTurn(GameTurn &gt) override;
    void update(float dt) override;
    void wormHit(GameTurn &gt, uint8_t wormId) override;
    void wormEndHit(GameTurn &gt, uint8_t wormId) override;
    void wormDrowning(GameTurn &gt, uint8_t wormId) override;
    void wormDrowned(GameTurn &gt, uint8_t wormId) override;
    void explosion() override;
    void wormDisconnectedDying(uint8_t wormId) override;
    void wormDisconnectedDead(uint8_t wormId) override;
    std::vector<uint8_t> &getWormsHit();
    void impactNotEnded();

   private:
    std::vector<uint8_t> wormsStillHit;
    std::vector<uint8_t> wormsHit;
//    uint8_t wormToFollow{0};
    bool impactEnded{false};
    uint8_t bulletFragments{0};
    uint8_t fragmentExplosions{0};
};
}

#endif  // INC_4_WORMS_IMPACTONCOURSE_H
```

```
//
// Created by rodrigo on 10/06/18.
//

#include <algorithm>
#include <iostream>

#include "GameStateMsg.h"
#include "ImpactOnCourse.h"

void Worms::ImpactOnCourse::endTurn(GameTurn &gt) {
    if (this→impactEnded ∧ this→wormsFalling.size() ≡ 0 ∧ this→wormsDrowning.size() ≡ 0 ∧
        ¬this→wormsDying ∧ this→wormsDisconnectedDying.size() ≡ 0) {
        this→notify(*this, Event::TurnEnded);
    }
}

void Worms::ImpactOnCourse::wormHit(GameTurn &gt, uint8_t wormId) {
    this→wormsStillHit.emplace_back(wormId);
    this→wormsHit.emplace_back(wormId);
    if (this→wormToFollow ≠ this→wormsStillHit[0]) {
        this→wormToFollow = this→wormsStillHit[0];
        this→notify(*this, Event::NewWormToFollow);
    }
}

void Worms::ImpactOnCourse::wormEndHit(Worms::GameTurn &gt, uint8_t wormId) {
    this→wormsStillHit.erase(
        std::remove(this→wormsStillHit.begin(), this→wormsStillHit.end(), wormId),
        this→wormsStillHit.end());
    if (this→wormToFollow ≡ wormId) {
        this→wormToFollow = this→wormsStillHit[0];
        this→notify(*this, Event::NewWormToFollow);
    }
}

void Worms::ImpactOnCourse::wormDrowning(Worms::GameTurn &gt, uint8_t wormId) {
    this→wormsDrowning.emplace_back(wormId);
    this→wormLanded(wormId);
    if (this→wormsStillHit.size() ≡ 0) {
        if (this→wormToFollow ≠ this→wormsDrowning[0]) {
            this→wormToFollow = this→wormsDrowning[0];
            this→notify(*this, Event::NewWormToFollow);
        }
    }
}

void Worms::ImpactOnCourse::wormDrowned(Worms::GameTurn &gt, uint8_t wormId) {
    this→wormsDrowning.erase(
        std::remove(this→wormsDrowning.begin(), this→wormsDrowning.end(), wormId),
        this→wormsDrowning.end());
    if (this→wormsStillHit.size() ≡ 0) {
        if (this→wormToFollow ≠ this→wormsDrowning[0]) {
            this→wormToFollow = this→wormsDrowning[0];
            this→notify(*this, Event::NewWormToFollow);
        }
    }
}

std::vector<uint8_t> &Worms::ImpactOnCourse::getWormsHit() {
    return this→wormsHit;
}

```



```
void Worms::ImpactOnCourse::impactNotEnded() {
    this→impactEnded = false;
}

Worms::ImpactOnCourse::ImpactOnCourse(uint8_t bulletFragments) {
    this→bulletFragments = bulletFragments;
}

void Worms::ImpactOnCourse::explosion() {
    this→fragmentExplosions++;
}

void Worms::ImpactOnCourse::update(float dt) {
    if (¬this→impactEnded) {
        if (this→wormsStillHit.size() ≡ 0 ∧ this→wormsDrowning.size() ≡ 0 ∧
            this→fragmentExplosions ≡ this→bulletFragments) {
            this→impactEnded = true;
            this→notify(*this, Event::ImpactEnd);
        }
    }
}

void Worms::ImpactOnCourse::wormDisconnectedDying(uint8_t wormId) {
    this→wormsDisconnectedDying.emplace_back(wormId);
    if (this→wormToFollow ≠ this→wormsDisconnectedDying[0] ∧ this→wormsFalling.size() ≡ 0 ∧ this→wormsDrowning.size() ≡ 0) {
        this→wormToFollow = this→wormsDisconnectedDying[0];
        this→notify(*this, Event::NewWormToFollow);
    }
}

void Worms::ImpactOnCourse::wormDisconnectedDead(uint8_t wormId) {
    this→wormsDisconnectedDying.erase(
            std::remove(this→wormsDisconnectedDying.begin(), this→wormsDisconnectedDying.end(), wormId),
            this→wormsDisconnectedDying.end());
    if (this→wormToFollow ≡ wormId) {
        this→wormToFollow = this→wormsDisconnectedDying[0];
        this→notify(*this, Event::NewWormToFollow);
    }
}
```

jun 26, 18 17:16 **GameTurnState.h** Page 1/1

```
1  //
2  // Created by rodrigo on 10/06/18.
3  //
4
5  #ifndef INC_4_WORMS_GAMETURNSTATE_H
6  #define INC_4_WORMS_GAMETURNSTATE_H
7
8  #include <cstdint>
9  #include <vector>
10
11 #include "../../../libs/Subject.h"
12 #include "GameStateMsg.h"
13
14 namespace Worms {
15 class GameTurn;
16 class GameTurnState : public Subject {
17    public:
18     GameTurnState();
19     virtual ~GameTurnState() = default;
20
21     virtual void endTurn(GameTurn &gt) = 0;
22     virtual void update(float dt) = 0;
23     virtual void wormHit(GameTurn &gt, uint8_t wormId) = 0;
24     virtual void wormEndHit(GameTurn &gt, uint8_t wormId) = 0;
25     virtual void wormDrowning(GameTurn &gt, uint8_t wormId) = 0;
26     virtual void wormDrowned(GameTurn &gt, uint8_t wormId) = 0;
27     virtual void explosion() = 0;
28     virtual void wormFalling(uint8_t wormId);
29     virtual void wormLanded(uint8_t wormId);
30     virtual void wormDying();
31     virtual void wormDead();
32     virtual void wormDisconnectedDying(uint8_t wormId) = 0;
33     virtual void wormDisconnectedDead(uint8_t wormId) = 0;
34     virtual uint8_t getWormToFollow() const;
35
36    protected:
37     std::vector<uint8_t> wormsFalling;
38     std::vector<uint8_t> wormsDrowning;
39     uint8_t wormsDying{0};
40     std::vector<uint8_t> wormsDisconnectedDying;
41     uint8_t wormToFollow{0};
42 };
43 }
44
45 #endif  // INC_4_WORMS_GAMETURNSTATE_H
```

jun 26, 18 17:16 **GameTurnState.cpp** Page 1/1

```
1  //
2  // Created by rodrigo on 10/06/18.
3  //
4
5  #include <algorithm>
6
7  #include "GameTurnState.h"
8
9  Worms::GameTurnState::GameTurnState() {}
10
11 void Worms::GameTurnState::wormFalling(uint8_t wormId) {
12     this→wormsFalling.emplace_back(wormId);
13 }
14
15 void Worms::GameTurnState::wormLanded(uint8_t wormId) {
16     this→wormsFalling.erase(
17         std::remove(this→wormsFalling.begin(), this→wormsFalling.end(), wormId
   ),
18         this→wormsFalling.end());
19 }
20
21 void Worms::GameTurnState::wormDead() {
22     this→wormsDying−−;
23 }
24
25 void Worms::GameTurnState::wormDying() {
26     this→wormsDying++;
27 }
28
29 uint8_t Worms::GameTurnState::getWormToFollow() const {
30     return this→wormToFollow;
31 }
```

jun 26, 18 17:16 **GamesGetter.h** Page 1/1

```h
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 17/06/18
 */

#ifndef __GamesGetter_H__
#define __GamesGetter_H__

#include <list>
#include <string>

#include "GameStateMsg.h"
#include "Lobby.h"

struct GamesGetter{
public:
    void operator()(const std::list<Worms::Lobby> &lobbies);
    std::vector<IO::GameInfo> gamesInfo;
};


#endif //__GamesGetter_H__
```

jun 26, 18 17:16 **GamesGetter.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 17/06/18
 */

#include "GamesGetter.h"

void GamesGetter::operator()(const std::list<Worms::Lobby> &lobbies){
    for (auto &lobby : lobbies) {
        auto &levelInfo = lobby.getLevelInfo();
        IO::GameInfo gameInfo{lobby.getID(),
                              levelInfo.id,
                              levelInfo.name,
                              lobby.getActualPlayers(),
                              levelInfo.playersQuantity};
        this→gamesInfo.emplace_back(gameInfo);
    }
}
```

jun 26, 18 17:16 **GameLobby.h** Page 1/1

```
//
// Created by rodrigo on 15/06/18.
//

#ifndef INC_4_WORMS_GAMELOBBY_H
#define INC_4_WORMS_GAMELOBBY_H


#define RESOURCE_PATH "/var/Worms/res/"

#include <list>
#include <string>

#include <CommunicationSocket.h>
#include <thread>
#include <GameStateMsg.h>
#include <Stream.h>
#include "ServerSocket.h"
#include "GameLobbyAssistant.h"

namespace Worms {
    class GameLobby : public Observer, public Thread {
    public:
        GameLobby(std::string port);
        GameLobby(GameLobby &copy) = delete;

        void run() override;
        void onNotify(Subject &subject, Event event) override;
        void stop() override;

    private:
        ServerSocket serverSocket;
        IO::Stream<IO::ServerInternalMsg> msgToJoiner;
        std::list<GameLobbyAssistant> players;
        bool quit{false};
        /**
         * @brief check if the GameLobbyAssistant thread is over. If so, join
         * it and erase it (because the sockets was already moved to the Lobby)
         */
        void removePlayers();

        void loadLevels(std::string &path, std::vector<IO::LevelData> &levels);
        void loadLevel(std::string &path, std::vector<IO::LevelData> &levels);
        std::string splitpath(const std::string &str, const std::set<char> &deli
miters);
        void loadLevelBackground(std::string &path, IO::LevelData &level);

        void killPlayers();
    };
}


#endif //INC_4_WORMS_GAMELOBBY_H
```

jun 26, 18 17:16 **GameLobby.cpp** Page 1/4

```
//
// Created by rodrigo on 15/06/18.
//

#include <iostream>
#include <dirent.h>

#include "GameLobby.h"
#include "ServerSocket.h"
#include "Lobbies.h"
#include "Game.h"
#include "LobbyJoiner.h"
#include "Stage.h"

Worms::GameLobby::GameLobby(std::string port) :
        serverSocket(port.c_str()) {
    std::cout << "Se bindeo" << std::endl;
}

void Worms::GameLobby::run() {
    std::string path(RESOURCE_PATH);
    std::vector<IO::LevelData> levels;

    Lobbies lobbies{levels};

    LobbyJoiner lobbyJoiner{lobbies, this→msgToJoiner};
    try {
        this→loadLevels(path, levels);
        lobbies.configure();
        lobbyJoiner.start();
        int id = 0;

        while (¬quit) {
            this→players.emplace_back(this→serverSocket.accept(), lobbies, id,
 this);
            this→players.back().start();
            id++;

            this→removePlayers();

            std::cout << "hubo una conexiÃ³n" << std::endl;
        }

    } catch (std::exception &e) {
        if (¬this→quit){
            std::cerr << "In GameLobby::run()" << std::endl;
            std::cerr << e.what() << std::endl;
        }
    } catch (...) {
        std::cerr << "Unkown error in GameLobby::run()" << std::endl;
    }

    this→killPlayers();
    this→msgToJoiner << IO::ServerInternalMsg{IO::ServerInternalAction::quit};
    lobbyJoiner.join();
}

void Worms::GameLobby::stop() {
    this→quit = true;
    this→serverSocket.shutdown();
}

void Worms::GameLobby::onNotify(Subject &subject, Event event) {
    switch (event) {
        case Event::StartGame: {
            auto &lobby = dynamic_cast<Lobby &>(subject);
```

```
66
67              /** En algÃºn momento le tengo que sacar el socket al GameLobbyAssis
    tant
68               *  para crear un vector con los sockets de todos los jugadores, que
     es lo que
69               *  recibe Game, entonces pienso que es mejor que sea al momento de
    iniciar la partida
70               *  por si el jugador se arrepiente antes y quiere salir, que el Ass
    istant lo pueda
71               *  manejar.
72               */
73
74              const std::vector<int> &playerIDs = lobby.getPlayerIDs();
75              for (auto &playerID : playerIDs) {
76                  for (auto &player : this→players){
77                      if (player.getPlayerID() ≡ playerID){
78                          //TODO revisar el lugar donde se setea terminado el hilo
79                          lobby.addPlayerSocket(std::move(player.getSocket()));
80                          player.stop();
81                      };
82                  }
83              }
84              lobby.start();
85
86              break;
87          }
88          case Event::EndGame: {
89              this→msgToJoiner << IO::ServerInternalMsg{IO::ServerInternalAction:
    :lobbyFinished};
90              break;
91          }
92          default: {
93              break;
94          }
95      }
96  }
97
98  void Worms::GameLobby::removePlayers(){
99      std::list<GameLobbyAssistant>::iterator playerIt;
100     playerIt = this→players.begin();
101     while (playerIt ≠ this→players.end()){
102         if (playerIt→itsOver()){
103             playerIt→join();
104             playerIt = this→players.erase(playerIt);
105         } else {
106             playerIt++;
107         }
108     }
109 }
110
111 void Worms::GameLobby::loadLevels(std::string &path, std::vector<IO::LevelData>
    &levels) {
112     DIR *dir;
113     struct dirent *ent;
114     if ((dir = opendir(path.c_str())) ≠ NULL) {
115         /* print all the files and directories within directory */
116         while ((ent = readdir(dir)) ≠ NULL) {
117             if (std::string(ent→d_name)[0] ≠ '.') {
118                 std::string levelPath(path + ent→d_name + "/");
119                 this→loadLevel(levelPath, levels);
120             }
121         }
122         closedir (dir);
123     } else {
124         /* could not open directory */
125         throw Exception("Could not open directory: %s", path.c_str());
```



```
126     }
127 }
128
129 void Worms::GameLobby::loadLevel(std::string &path, std::vector<IO::LevelData> &
    levels) {
130     DIR *dir;
131     struct dirent *ent;
132     IO::LevelData level;
133     if ((dir = opendir(path.c_str())) ≠ NULL) {
134         /* print all the files and directories within directory */
135         while ((ent = readdir(dir)) ≠ NULL) {
136             if (std::string(ent→d_name)[0] ≠ '.') {
137                 std::string levelPath(path + ent→d_name);
138 //                if (std::string(ent->d_name) == "Background") {
139 //                    std::string backgroundsPath(levelPath + "/");
140 //                    this->loadLevelBackground(backgroundsPath, level);
141 //                } else {
142                 std::string levelName(ent→d_name);
143                 YAML::Node data = YAML::LoadFile(levelPath);
144                 std::set<char> delims{'/'};
145                 std::string closeBackgroundFile = data["background"]["closeBackground
    File"].as<std::string>();
146                 level.backgroundName.emplace_back(std::move(this→splitpath(clos
    eBackgroundFile, delims)));
147                 level.backgroundPath.emplace_back(std::move(closeBackgroundFile)
    );
148                 std::string midBackgroundFile = data["background"]["midBackgroundFile
    "].as<std::string>();
149                 level.backgroundName.emplace_back(std::move(this→splitpath(midB
    ackgroundFile, delims)));
150                 level.backgroundPath.emplace_back(std::move(midBackgroundFile));
151                 std::string fartherBackgroundFile = data["background"]["fartherBackgro
    undFile"].as<std::string>();
152                 level.backgroundName.emplace_back(std::move(this→splitpath(fart
    herBackgroundFile, delims)));
153                 level.backgroundPath.emplace_back(std::move(fartherBackgroundFil
    e));
154
155
156
157 //                    levelName = levelName.substr(0, levelName.find('.'));
158
159                 level.levelPath = std::move(levelPath);
160                 level.levelName = std::move(levelName);
161 //                }
162             }
163         }
164         closedir (dir);
165         levels.emplace_back(std::move(level));
166     } else {
167         /* could not open directory */
168         throw Exception("Could not open directory: %s", path.c_str());
169     }
170 }
171
172 std::string Worms::GameLobby::splitpath(const std::string &str, const std::set<c
    har> &delimiters) {
173     std::vector<std::string> result;
174
175     char const* pch = str.c_str();
176     char const* start = pch;
177     for(; *pch; ++pch) {
178         if (delimiters.find(*pch) ≠ delimiters.end()) {
179             if (start ≠ pch) {
180                 std::string str(start, pch);
181                 result.push_back(str);
```

jun 26, 18 17:16 **GameLobby.cpp** Page 4/4

```
182                 } else {
183                     result.emplace_back("");
184                 }
185                 start = pch + 1;
186             }
187         }
188         result.emplace_back(start);
189 
190         return result.back();
191     }
192 
193     void Worms::GameLobby::loadLevelBackground(std::string &path, IO::LevelData &lev
        el) {
194         DIR *dir;
195         struct dirent *ent;
196         std::vector<std::string> backgrounds;
197         if ((dir = opendir(path.c_str())) ≠ NULL) {
198             /* print all the files and directories within directory */
199             while ((ent = readdir(dir)) ≠ NULL) {
200                 if (std::string(ent→d_name)[0] ≠ '.') {
201                     std::string backgroundPath(path + ent→d_name);
202                     std::string backgroundName(ent→d_name);
203 
204                     level.backgroundPath.emplace_back(std::move(backgroundPath));
205                     level.backgroundName.emplace_back(std::move(backgroundName));
206                 }
207             }
208             closedir (dir);
209         } else {
210             /* could not open directory */
211             throw Exception("Could not open directory: %s", path.c_str());
212         }
213     }
214 
215     void Worms::GameLobby::killPlayers(){
216         std::list<GameLobbyAssistant>::iterator playerIt;
217         playerIt = this→players.begin();
218         while (playerIt ≠ this→players.end()){
219             playerIt→stop();
220             playerIt→join();
221             playerIt++;
222         }
223         this→players.erase(this→players.begin(), this→players.end());
224     }
```

jun 26, 18 17:16 **GameLobbyAssistant.h** Page 1/1

```
1   //
2   // Created by rodrigo on 15/06/18.
3   //
4   
5   #ifndef INC_4_WORMS_GAMELOBBYASSISTANT_H
6   #define INC_4_WORMS_GAMELOBBYASSISTANT_H
7   
8   
9   #include <Protocol.h>
10  #include <sstream>
11  
12  #include "Thread.h"
13  #include "Lobbies.h"
14  #include "Observer.h"
15  
16  namespace Worms {
17      class GameLobbyAssistant : public Thread, public Observer {
18      public:
19          explicit GameLobbyAssistant(CommunicationSocket ∧communicationSocket, L
        obbies &lobbies, int id,
20                                      Observer *lobbyObs);
21          GameLobbyAssistant(GameLobbyAssistant &copy) = delete;
22          void run() override;
23          void stop() override;
24          bool itsOver() const;
25          void onNotify(Subject &subject, Event event) override;
26          int getPlayerID() const;
27          CommunicationSocket getSocket();
28  
29      private:
30          Protocol<CommunicationSocket> protocol;
31          Lobbies &lobbies;
32          int playerID;
33          std::vector<Observer *> lobbyObservers;
34          bool finished{false};
35  
36          void getLevels();
37          void getGames();
38          void joinGame();
39  
40          void createGame();
41  
42          bool quit{false};
43  
44          void sendLevelFiles(uint8_t level);
45      };
46  }
47  
48  #endif //INC_4_WORMS_GAMELOBBYASSISTANT_H
```

```
1   //
2   // Created by rodrigo on 15/06/18.
3   //
4
5   #include <fstream>
6   #include <iostream>
7
8   #include "GameLobbyAssistant.h"
9   #include <GameStateMsg.h>
10  #include "Protocol.h"
11  #include "Lobbies.h"
12  #include "GamesGetter.h"
13
14  Worms::GameLobbyAssistant::GameLobbyAssistant(CommunicationSocket ∧communicatio
    nSocket, Lobbies &lobbies, int id,
15                                                Observer *lobbyObs) :
16          protocol(communicationSocket),
17          lobbies(lobbies),
18          playerID(id) {
19      this→lobbyObservers.emplace_back(lobbyObs);
20      this→lobbyObservers.emplace_back(this);
21  }
22
23  void Worms::GameLobbyAssistant::run() {
24      try {
25          std::uint8_t command{COMMAND_GET_LEVELS};
26          while (¬this→quit) {
27              this→protocol >> command;
28              switch (command) {
29                  case COMMAND_GET_LEVELS:
30                      this→getLevels();
31                      break;
32                  case COMMAND_CREATE_GAME:
33                      this→createGame();
34                      break;
35                  case COMMAND_GET_GAMES:
36                      this→getGames();
37                      break;
38                  case COMMAND_JOIN_GAME:
39                      this→joinGame();
40                      break;
41              }
42          }
43  //        this->createGame();
44  //        this->createGame();
45  //        this->createGame();
46  //        this->getGames();
47      } catch (std::exception &e) {
48          std::cerr << "In GameLobbyAssistant::run()" << std::endl;
49          std::cerr << e.what() << std::endl;
50      } catch (...) {
51          std::cerr << "Unkown error in GameLobbyAssistant::run()" << std::endl;
52      }
53  }
54
55  void Worms::GameLobbyAssistant::stop() {
56      this→finished = true;
57      this→protocol.stopCommunication();
58  }
59
60  void Worms::GameLobbyAssistant::getLevels() {
61  //    std::vector<IO::LevelInfo> levelsInfo;
62  //    IO::LevelInfo levelInfo{"First Stage", 2};
63  //    levelsInfo.emplace_back(levelInfo);
64  //    levelInfo = {"Second Stage", 3};
65  //    levelsInfo.emplace_back(levelInfo);
```



```
66  //    levelInfo = {"Third Stage", 4};
67  //    levelsInfo.emplace_back(levelInfo);
68
69
70      this→protocol << this→lobbies.getLevels();
71  }
72
73  void Worms::GameLobbyAssistant::createGame() {
74      uint8_t levelSelected{0};
75      this→protocol >> levelSelected;
76      this→sendLevelFiles(levelSelected);
77
78      this→lobbies.createGame(this→playerID, this→lobbyObservers, levelSelected)
    ;
79      this→quit = true;
80  }
81
82  void Worms::GameLobbyAssistant::getGames() {
83      GamesGetter getter;
84      this→lobbies.getGames(getter);
85      this→protocol << getter.gamesInfo;
86  }
87
88  void Worms::GameLobbyAssistant::joinGame() {
89      std::uint8_t gameID{0};
90      std::uint8_t levelID{0};
91      this→protocol >> gameID;
92      this→protocol >> levelID;
93      this→sendLevelFiles(levelID);
94      this→lobbies.joinGame(gameID, this→playerID, this);
95      this→quit = true;
96  }
97
98  void Worms::GameLobbyAssistant::onNotify(Subject &subject, Event event) {
99      switch (event) {
100         case Event::NewPlayer: {
101             auto &lobby = dynamic_cast<Lobby &>(subject);
102             this→protocol << lobby.getActualPlayers();
103             break;
104         }
105         default: {
106             break;
107         }
108     }
109 }
110
111 CommunicationSocket Worms::GameLobbyAssistant::getSocket() {
112     return std::move(this→protocol.getSocket());
113 }
114
115 int Worms::GameLobbyAssistant::getPlayerID() const{
116     return this→playerID;
117 }
118
119 bool Worms::GameLobbyAssistant::itsOver() const{
120     return this→finished;
121 }
122
123 void Worms::GameLobbyAssistant::sendLevelFiles(uint8_t level) {
124     const IO::LevelData &levelData = this→lobbies.getLevelData(level);
125     this→protocol << levelData.levelName;
126     std::ifstream levelFile(levelData.levelPath, std::ifstream::binary);
127     this→protocol << levelFile;
128
129     this→protocol << levelData.backgroundName;
130     for (auto &background : levelData.backgroundPath) {
```

jun 26, 18 17:16 **GameLobbyAssistant.cpp** Page 3/3

```
131         std::ifstream backgroundFile(background, std::ifstream::binary);
132         if (¬backgroundFile) {
133         }
134         this→protocol << backgroundFile;
135     }
136 }
```

jun 26, 18 17:16 **Game.h** Page 1/2

```
1  /*
2   *  Created by Federico Manuel Gomez Peter.
3   *  date: 18/05/18
4   */
5
6  #ifndef __GAME_H__
7  #define __GAME_H__
8
9  #include <atomic>
10 #include <list>
11 #include <thread>
12 #include <unordered_map>
13
14 #include "CommunicationSocket.h"
15 #include "Direction.h"
16 #include "DoubleBuffer.h"
17 #include "GameClock.h"
18 #include "GameTeams.h"
19 #include "GameTurn.h"
20 #include "Girder.h"
21 #include "Observer.h"
22 #include "Player.h"
23 #include "Stage.h"
24 #include "Weapons/Bullet.h"
25
26 namespace Worms {
27 using PlayerInput = IO::Stream<IO::PlayerMsg>;
28 using GameSnapshot = IO::DoubleBuffer<IO::GameStateMsg>;
29
30 struct Teamasd {
31     std::vector<uint8_t> players;
32     uint8_t currentPlayer;
33     bool alive;
34 };
35
36 class Game : Observer {
37    public:
38     std::atomic<bool> quit{false};
39
40     Game(Stage ∧stage, std::vector<CommunicationSocket> &sockets);
41     virtual ~Game();
42
43     Game(Game ∧other) = delete;
44
45     void start();
46     IO::GameStateMsg serialize() const;
47     void onNotify(Subject &subject, Event event) override;
48     /**
49      * @brief calculates damage for weapons that throw bullets. It gives
50      * information of the bullet to all players so them can calculate his damage
51      * and apply an impulse if this was hitted.
52      * @param bullet
53      */
54     void calculateDamage(const Bullet &bullet);
55     /**
56      * @brief calculates damage for p2p weapons (baseball). It gives
57      * information of the weapon (direction, point and damageInfo) to the
58      * players so that they can calculate his damage and apply an impulse if
59      * this was hitted.
60      * @param weapon
61      */
62     void calculateDamage(std::shared_ptr<Worms::Weapon> weapon, Math::Point<floa
    t> shooterPosition,
63                          Worm::Direction shooterDirection);
64     void calculateWind();
65     void exit();
```

jun 26, 18 17:16 **Game.h** Page 2/2

```
66      void endTurn();
67
68     private:
69      void inputWorker(std::size_t playerIndex);
70      void outputWorker(std::size_t playerIndex);
71      void calculateCurrentPlayer();
72
73      uint8_t currentWorm;
74      uint8_t currentTeam{0};
75      Physics physics;
76      Stage stage;
77      std::vector<Girder> girders;
78      std::vector<Player> players;
79      std::vector<std::uint32_t> teamHealths;
80      const double maxTurnTime;
81      bool processingClientInputs{false};
82      uint8_t currentWormToFollow{0};
83      bool currentPlayerShot{false};
84      GameTeams teams;
85      std::list<Bullet> bullets;
86      Config::Wind wind;
87
88      std::vector<uint8_t> deadTeams;
89      GameClock gameClock;
90      GameTurn gameTurn;
91
92      /* communication */
93      std::vector<std::thread> inputThreads;
94      std::vector<std::thread> outputThreads;
95      std::vector<CommunicationSocket> &sockets;
96      std::vector<PlayerInput> inputs;
97      std::vector<GameSnapshot> snapshots;
98      std::uint8_t playersConnected;
99      bool removeBullets{false};
100     bool gameEnded{false};
101     std::uint8_t winnerTeam{0};
102     bool waitingForNextTurn{true};
103
104 void playerDisconnected(uint8_t teamDisconnected);
105 };
106 }  // namespace Worms
107
108 #endif  //__GAME_H__
```

jun 26, 18 17:16 **Game.cpp** Page 1/10

```
1   /*
2    *  Created by Federico Manuel Gomez Peter.
3    *  date: 18/05/18
4    */
5
6   #include <Stage.h>
7   #include <zconf.h>
8   #include <atomic>
9   #include <cassert>
10  #include <chrono>
11  #include <iostream>
12  #include <random>
13  #include "Box2D/Box2D.h"
14  #include "Chronometer.h"
15
16  #include "Config/Config.h"
17  #include "Direction.h"
18  #include "Game.h"
19  #include "GameStates/ImpactOnCourse.h"
20  #include "Player.h"
21  #include "Stage.h"
22  #include "Weapons/BaseballBat.h"
23
24  #define CONFIG ::Game::Config::getInstance()
25  #define TIME_STEP (1.0f / 30.0f)
26
27  Worms::Game::Game(Stage &stage, std::vector<CommunicationSocket> &sockets)
28      : physics(b2Vec2{0.0f, -10.0f}, TIME_STEP),
29        stage(std::move(stage)),
30        maxTurnTime(::Game::Config::getInstance().getExtraTurnTime()),
31        gameTurn(*this),
32        sockets(sockets),
33        inputs(sockets.size()),
34        snapshots(sockets.size()),
35        playersConnected(sockets.size()) {
36      this->inputThreads.reserve(sockets.size());
37      this->outputThreads.reserve(sockets.size());
38      for (std::size_t i = 0; i < sockets.size(); i++) {
39          this->inputThreads.emplace_back([this, i] { this->inputWorker(i); });
40          this->outputThreads.emplace_back([this, i] { this->outputWorker(i); });
41      }
42      /* reserves the required space to avoid reallocations that may move the worm
     addresses */
43      this->players.reserve(this->stage.getWorms().size());
44      uint8_t id = 0;
45      for (auto &wormData : this->stage.getWorms()) {
46          /* initializes the instances */
47          this->players.emplace_back(this->physics);
48          this->players.back().setPosition(wormData.position);
49          this->players.back().health = wormData.health;
50          this->players.back().setId(id);
51          this->players.back().addObserver(this);
52          id++;
53      }
54
55      this->teams.makeTeams(this->players, (uint8_t)sockets.size(), this->stage.get
    AmmoCounter());
56      //    this->wind.range = CONFIG.getWindIntensityRange();
57      this->wind.minIntensity = CONFIG.getMinWindIntensity();
58      this->wind.maxIntensity = CONFIG.getMaxWindIntensity();
59      this->calculateWind();
60
61      /* sets the girders */
62      this->girders.reserve(this->stage.getGirders().size());
63      for (auto &girder : this->stage.getGirders()) {
64          this->girders.emplace_back(girder, this->physics);
```

```
65      }
66
67      /* calculate the initial team's healths */
68      this→teamHealths = this→teams.getTotalHealth(this→players);
69
70      this→currentWorm = this→teams.getCurrentPlayerID();
71      this→currentWormToFollow = this→currentWorm;
72
73      this→gameClock.addObserver(this);
74      this→gameClock.waitForNextTurn();
75  }
76
77  Worms::Game::~Game() {
78      this→exit();
79      for (auto &t : this→outputThreads) {
80          t.join();
81      }
82
83      for (auto &t : this→inputThreads) {
84          t.join();
85      }
86  }
87  ///**
88  // * @brief Reads player messages from a socket and pushes them into the input q
    ueue.
89  // *
90  // * @param playerIndex The index of the player.
91  // */
92  //void Worms::Game::inputWorker(std::size_t playerIndex) {
93  //    PlayerInput &input = this->inputs.at(playerIndex);
94  //    CommunicationSocket &socket = this->sockets.at(playerIndex);
95  //
96  //    /* TODO: avoid hardcoding the size */
97  //    IO::PlayerMsg msg;
98  //    char *buffer = new char[msg.getSerializedSize()];
99  //
100 //    try {
101 //        while (!this->quit) {
102 //            /* reads the raw data from the buffer */
103 //            socket.receive(buffer, msg.getSerializedSize());
104 //
105 //            /* sets the struct data from the buffer */
106 //            msg.deserialize(buffer, msg.getSerializedSize());
107 //
108 //            /* pushes the message into the player's input queue if it's the cu
    rrent player */
109 //            if (this->currentTeam == playerIndex) {
110 //                input.push(msg);
111 //            }
112 //        }
113 //    } catch (const std::exception &e) {
114 //        std::cerr << "Worms::Game::inputWorker:" << e.what() << std::endl;
115 //        msg.input = IO::PlayerInput::disconnected;
116 //        msg.position = Math::Point<float>{0, 0};
117 //        input.push(msg);
118 //    } catch (...) {
119 //        std::cerr << "Unknown error in Worms::Game::inputWorker()" << std::end
    l;
120 //    }
121 //
122 //    delete[] buffer;
123 //}
124 //
125 ///**
126 // * @brief Sends model snapshot messages to a socket.
127 // *
```



```
128 // * @param playerIndex The index of the player to send the spanshots to.
129 // */
130 //void Worms::Game::outputWorker(std::size_t playerIndex) {
131 //    CommunicationSocket &socket = this->sockets.at(playerIndex);
132 //    GameSnapshot &snapshot = this->snapshots.at(playerIndex);
133 //
134 //    IO::GameStateMsg msg;
135 //    char *buffer = new char[msg.getSerializedSize()];
136 //
137 //    try {
138 //        while (!this->quit) {
139 //            msg = snapshot.get(true);
140 //            msg.serialize(buffer, msg.getSerializedSize());
141 //            socket.send(buffer, msg.getSerializedSize());
142 //        }
143 //    } catch (const IO::Interrupted &e) {
144 //        /* this means that the game is ready to exit */
145 //    } catch (const std::exception &e) {
146 //        std::cerr << "Worms::Game::outputWorker:" << e.what() << std::endl;
147 //    } catch (...) {
148 //        std::cerr << "Unknown error in Worms::Game::outputWorker()" << std::en
    dl;
149 //    }
150 //
151 //    delete[] buffer;
152 //}
153
154
155 /**
156  * @brief Reads player messages from a socket and pushes them into the input que
    ue.
157  *
158  * @param playerIndex The index of the player.
159  */
160 void Worms::Game::inputWorker(std::size_t playerIndex) {
161     PlayerInput &input = this→inputs.at(playerIndex);
162     CommunicationSocket &socket = this→sockets.at(playerIndex);
163
164     /* TODO: avoid hardcoding the size */
165     IO::PlayerMsg msg;
166     try {
167         while (¬this→quit) {
168             /* receives the size of the msg */
169             std::uint32_t size(0);
170             socket.receive((char *)&size, sizeof(std::uint32_t));
171             size = ntohl(size);
172
173             std::vector<char> buffer(size, 0);
174             /* reads the raw data from the buffer */
175             socket.receive(buffer.data(), size);
176
177             std::string buff(buffer.data(), size);
178
179             /* sets the struct data from the buffer */
180             msg.deserialize(buff);
181
182             /* pushes the message into the player's input queue if it's the curr
    ent player */
183             if (this→currentTeam ≡ playerIndex) {
184                 input.push(msg);
185             }
186         }
187     } catch (const std::exception &e) {
188         std::cerr << "Worms::Game::inputWorker:" << e.what() << std::endl;
189         msg.input = IO::PlayerInput::disconnected;
190         msg.position = Math::Point<float>{0, 0};
```

```
191             input.push(msg);
192         } catch (...) {
193             std::cerr << "Unknown error in Worms::Game::inputWorker()" << std::endl;
194         }
195     }
196
197     /**
198      * @brief Sends model snapshot messages to a socket.
199      *
200      * @param playerIndex The index of the player to send the spanshots to.
201      */
202     void Worms::Game::outputWorker(std::size_t playerIndex) {
203         CommunicationSocket &socket = this→sockets.at(playerIndex);
204         GameSnapshot &snapshot = this→snapshots.at(playerIndex);
205
206         IO::GameStateMsg msg;
207         try {
208             while (¬this→quit) {
209                 msg = snapshot.get(true);
210                 std::string buff = msg.serialize();
211                 std::uint32_t size = buff.size();
212                 std::uint32_t netInt = htonl(size);
213
214                 socket.send((char *)&netInt, sizeof(std::uint32_t));
215                 socket.send(buff.data(), size);
216             }
217         } catch (const IO::Interrupted &e) {
218             /* this means that the game is ready to exit */
219         } catch (const std::exception &e) {
220             std::cerr << "Worms::Game::outputWorker:" << e.what() << std::endl;
221         }
222     }
223
224     void Worms::Game::start() {
225         try {
226             /* game loop */
227             Utils::Chronometer chronometer;
228             while (¬quit) {
229                 double dt = chronometer.elapsed();
230
231                 this→gameClock.update(dt);
232                 this→gameTurn.update(dt);
233
234                 IO::PlayerMsg pMsg;
235                 if (this→inputs.at(this→currentTeam).pop(pMsg, false)) {
236                     if (pMsg.input ≡ IO::PlayerInput::disconnected) {
237                         this→playerDisconnected(this→currentTeam);
238                     } else {
239                         if (this→processingClientInputs) {
240                             if (this→currentPlayerShot) {
241                                 if (pMsg.input ≠ IO::PlayerInput::startShot ∧
242                                     pMsg.input ≠ IO::PlayerInput::endShot ∧
243                                     pMsg.input ≠ IO::PlayerInput::positionSelected)
    {
244                                     this→players.at(this→currentWorm).handleState(
    pMsg);
245                                 }
246                             } else {
247                                 this→players.at(this→currentWorm).handleState(pMsg
    );
248                             }
249                         } else {
250                             this→players.at(this→currentWorm).handleState(pMsg);
251                         }
252                     }
253                 }
```



```
254
255                 /* updates the actors */
256                 for (auto &worm : this→players) {
257                     worm.update(dt);
258                 }
259
260                 for (auto &bullet : this→bullets) {
261                     bullet.update(dt, this→wind);
262                 }
263
264                 this→physics.update(dt);
265
266                 /* serializes and updates the game state */
267                 auto msg = this→serialize();
268                 for (auto &snapshot : this→snapshots) {
269                     snapshot.set(msg);
270                     snapshot.swap();
271                 }
272
273                 if (this→gameEnded) {
274                     this→quit = true;
275                 }
276
277                 if (TIME_STEP > dt) {
278                     usleep((TIME_STEP − dt) * 1000000);
279                 }
280             }
281         } catch (std::exception &e) {
282             std::cerr << e.what() << std::endl << "In Worms::Game::start" << std::endl;
283         } catch (...) {
284             std::cerr << "Unkown error in Worms::Game::start()" << std::endl;
285         }
286     }
287
288     void Worms::Game::endTurn() {
289         this→waitingForNextTurn = false;
290         this→processingClientInputs = true;
291         this→gameClock.restart();
292         this→gameTurn.restart();
293         this→calculateWind();
294     }
295
296     void Worms::Game::calculateCurrentPlayer() {
297         this→waitingForNextTurn = true;
298         this→players[this→currentWorm].reset();
299         this→gameEnded = this→teams.endTurn(this→players);
300         if (this→gameEnded) {
301             this→winnerTeam = this→teams.getWinner();
302         }
303         this→currentTeam = this→teams.getCurrentTeamID();
304         this→currentWorm = this→teams.getCurrentPlayerID();
305         this→currentWormToFollow = this→currentWorm;
306     }
307
308     IO::GameStateMsg Worms::Game::serialize() const {
309         assert(this→players.size() ≤ 20);
310
311         IO::GameStateMsg m;
312         memset(&m, 0, sizeof(m));
313
314         m.num_worms = 0;
315         m.num_teams = this→teams.getTeamQuantity();
316         for (const auto &worm : this→players) {
317             m.positions[m.num_worms * 2] = worm.getPosition().x;
318             m.positions[m.num_worms * 2 + 1] = worm.getPosition().y;
319             m.stateIDs[m.num_worms] = worm.getStateId();
```

```
320         m.wormsHealth[m.num_worms] = worm.health;
321         m.wormsTeam[m.num_worms] = worm.getTeam();
322         m.wormsDirection[m.num_worms] = worm.direction;
323         m.num_worms++;
324     }
325
326     /* sets team health*/
327     uint8_t i{0};
328     for (auto health : this→teamHealths) {
329         m.teamHealths[i++] = health;
330     }
331     /* sets wind data */
332     m.windIntensity =
333         (char)(127.0f * this→wind.instensity /
334                 (this→wind.maxIntensity − this→wind.minIntensity) * this→wind.x
    Direction);
335
336     /* sets the current player's data */
337     m.elapsedTurnSeconds = static_cast<std::uint16_t>(std::floor(this→gameClock
    .getTimeElapsed()));
338     m.currentPlayerTurnTime = static_cast<std::uint16_t>(std::floor(this→gameCl
    ock.getTurnTime()));
339     m.currentWorm = this→currentWorm;
340     m.currentWormToFollow = this→currentWormToFollow;
341     m.currentTeam = this→currentTeam;
342     m.activePlayerAngle = this→players[this→currentWorm].getWeaponAngle();
343     m.activePlayerWeapon = this→players[this→currentWorm].getWeaponID();
344
345     m.bulletsQuantity = this→bullets.size();
346     i = 0;
347     uint8_t j = 0;
348     for (auto &bullet : this→bullets) {
349         Math::Point<float> p = bullet.getPosition();
350         m.bullets[i++] = p.x;
351         m.bullets[i++] = p.y;
352         m.bulletsAngle[j] = bullet.getAngle();
353         m.bulletType[j++] = bullet.getWeaponID();
354     }
355     /*
356      * serialize the ammunition counter
357      */
358     this→teams.serialize(m);
359     m.processingInputs = this→processingClientInputs;
360     m.playerUsedTool = this→currentPlayerShot;
361     m.waitingForNextTurn = this→waitingForNextTurn;
362     m.gameEnded = this→gameEnded;
363     m.winner = this→winnerTeam;
364
365     return m;
366 }
367
368 void Worms::Game::exit() {
369     this→quit = true;
370     for (auto &snapshot : this→snapshots) {
371         snapshot.interrupt();
372     }
373     for (auto &socket : this→sockets) {
374         socket.shutdown();
375     }
376 }
377
378 void Worms::Game::onNotify(Subject &subject, Event event) {
379     switch (event) {
380         /**
381          * Because i didnt want to move all responsability of the bullets to
382          * the game (until the refactor of the start), i added this function
```



```
383          * that delegates to the player the responsability to iterate all over
384          * the bullets and add the game as an observer
385          */
386         case Event::Shot: {
387             //             this->players[this->currentWorm].addObserverToBullets
    (this);
388             this→bullets.merge(this→players[this→currentWorm].getBullets());
389             for (auto &bullet : this→bullets) {
390                 bullet.addObserver(this);
391             }
392             this→gameClock.playerShot();
393             this→gameTurn.playerShot(this→players[this→currentWorm].getWeaponI
    D());
394             this→currentPlayerShot = true;
395             break;
396         }
397         /**
398          * On explode, the game must check worms health.
399          */
400         case Event::Explode: {
401             auto &bullet = dynamic_cast<const Bullet &>(subject);
402             this→gameTurn.explosion();
403             this→calculateDamage(bullet);
404             break;
405         }
406         case Event::P2PWeaponUsed: {
407             auto &player = dynamic_cast<const Worms::Player &>(subject);
408             const std::shared_ptr<Worms::Weapon> weapon = player.getWeapon();
409             this→gameClock.playerShot();
410             this→gameTurn.playerShot(this→players[this→currentWorm].getWeaponI
    D());
411             this→currentPlayerShot = true;
412             this→gameTurn.explosion();
413             this→calculateDamage(weapon, player.getPosition(), player.direction
    );
414             break;
415         }
416         /**
417          * onExplode will create new Bullets in player's container, and we
418          * need to listen to them.
419          */
420         case Event::OnExplode: {
421             auto &bullet = dynamic_cast<const Bullet &>(subject);
422             this→calculateDamage(bullet);
423
424             this→bullets.merge(this→players[this→currentWorm].onExplode(bullet
    , this→physics));
425
426             for (auto &fragment : this→bullets) {
427                 fragment.addObserver(this);
428             }
429             //             this->players[this->currentWorm].addObserverToBullets(
    this);
430             break;
431         }
432         case Event::DyingDueToDisconnection: {
433             this→gameTurn.playerDisconnected(dynamic_cast<const Player &>(subje
    ct).getId());
434             break;
435         }
436         case Event::DeadDueToDisconnection: {
437             this→gameTurn.playerDisconnectedDead(dynamic_cast<const Player &>(s
    ubject).getId());
438             break;
439         }
440         case Event::Teleported: {
```

```
441             this→gameClock.playerShot();
442             this→currentPlayerShot = true;
443             this→teams.weaponUsed(this→players[this→currentWorm].getWeaponID()
    );
444             break;
445         }
446         case Event::WormFalling: {
447             this→gameTurn.wormFalling(dynamic_cast<const Player &>(subject).get
    Id());
448             break;
449         }
450         case Event::WormLanded: {
451             this→gameTurn.wormLanded(dynamic_cast<const Player &>(subject).getI
    d());
452             break;
453         }
454         case Event::Hit: {
455             this→gameTurn.wormHit(dynamic_cast<const Player &>(subject).getId()
    );
456             break;
457         }
458         case Event::EndHit: {
459             this→gameTurn.wormEndHit(dynamic_cast<const Player &>(subject).getI
    d());
460             break;
461         }
462         case Event::Drowning: {
463             this→gameTurn.wormDrowning(dynamic_cast<const Player &>(subject).ge
    tId());
464             break;
465         }
466         case Event::Drowned: {
467             this→gameTurn.wormDrowned(dynamic_cast<const Player &>(subject).get
    Id());
468             break;
469         }
470         case Event::Dying: {
471             this→gameTurn.wormDying();
472             break;
473         }
474         case Event::Dead: {
475             this→gameTurn.wormDead();
476             this→gameTurn.endTurn();
477             break;
478         }
479         case Event::NewWormToFollow: {
480             this→currentWormToFollow =
481                 dynamic_cast<const GameTurnState &>(subject).getWormToFollow();
482             break;
483         }
484         case Event::DamageOnLanding: {
485             this→gameClock.endTurn();
486             break;
487         }
488         case Event::ImpactEnd: {
489             auto &wormsHit = dynamic_cast<ImpactOnCourse &>(subject).getWormsHit
    ();
490             for (auto worm : wormsHit) {
491                 Worm::StateID wormState = this→players[worm].getStateId();
492                 if (this→players[worm].health ≡ 0) {
493                     if (wormState ≠ Worm::StateID::Die ∧ wormState ≠ Worm::State
    ID::Dead) {
494                         this→players[worm].notify(this→players[worm], Event::D
    ying);
495                         this→players[worm].setState(Worm::StateID::Die);
496                     }
```



```
497                 }
498             }
499             break;
500         }
501         case Event::EndTurn: {
502             this→processingClientInputs = false;
503             this→gameTurn.endTurn();
504             break;
505         }
506         case Event::TurnEnded: {
507             if (this→players[this→currentWorm].getStateId() ≠ Worm::StateID::D
    ead) {
508                 this→players[this→currentWorm].setState(Worm::StateID::Still);
509             }
510             this→bullets.erase(this→bullets.begin(), this→bullets.end());
511             this→gameClock.waitForNextTurn();
512             this→teamHealths = this→teams.getTotalHealth(this→players);
513             this→calculateCurrentPlayer();
514             break;
515         }
516         case Event::NextTurn: {
517             this→currentPlayerShot = false;
518             this→endTurn();
519             break;
520         }
521         default: {
522             break;
523         }
524     }
525 }
526
527 void Worms::Game::calculateDamage(const Worms::Bullet &bullet) {
528     Config::Bullet::DamageInfo damageInfo = bullet.getDamageInfo();
529     for (auto &worm : this→players) {
530         worm.acknowledgeDamage(damageInfo, bullet.getPosition());
531     }
532     this→removeBullets = true;
533 }
534 /**
535  * @brief calculate damage for p2p weapons. Because the only one is the
536  * baseball bat and because we are running out of time, there will be
537  * a cast to a baseballWeapon.
538  * TODO make a class between weapon and baseballBat, that represents a
539  * p2pWeapon.
540  * @param weapon
541  */
542 void Worms::Game::calculateDamage(std::shared_ptr<Worms::Weapon> weapon,
543                                   Math::Point<float> shooterPosition,
544                                   Worm::Direction shooterDirection) {
545     auto *baseball = (::Weapon::BaseballBat *)weapon.get();
546     Config::P2PWeapon &weaponInfo = baseball→getWeaponInfo();
547     for (auto &worm : this→players) {
548         worm.acknowledgeDamage(weaponInfo, shooterPosition, shooterDirection);
549     }
550     this→removeBullets = true;
551 }
552
553 void Worms::Game::calculateWind() {
554         std::random_device rnd_device;
555         std::mt19937 mersenne_engine(rnd_device());
556         std::uniform_real_distribution<> distr(this→wind.minIntensity, this→wi
    nd.maxIntensity);
557
558         this→wind.xDirection =
559             (distr(mersenne_engine) > (this→wind.maxIntensity − this→wind.minI
    ntensity) / 2.0f)
```

jun 26, 18 17:16 **Game.cpp** Page 10/10

```
560             ? 1
561
                    : -1;
562         this→wind.instensity = (float) distr(mersenne_engine);
563 }
564
565 void Worms::Game::playerDisconnected(uint8_t teamDisconnected) {
566     this→playersConnected--;
567     this→teams.kill(teamDisconnected, this→players);
568     if (this→playersConnected ≤ 1) {
569         this→winnerTeam = this→teams.getWinner();
570         this→gameEnded = true;
571     }
572 }
```

jun 26, 18 17:16 **GameClock.h** Page 1/1

```
1  //
2  // Created by rodrigo on 10/06/18.
3  //
4
5  #ifndef INC_4_WORMS_GAMECLOCK_H
6  #define INC_4_WORMS_GAMECLOCK_H
7
8  #include "Config/Config.h"
9  #include "Subject.h"
10
11 class GameClock : public Subject {
12    public:
13     GameClock();
14     ~GameClock() = default;
15     void update(float dt);
16     void playerShot();
17     double getTimeElapsed() const;
18     double getTurnTime() const;
19     void waitForNextTurn();
20     void restart();
21     void endTurn();
22
23    private:
24     float timeElapsed{0.0f};
25     float turnTime;
26     float extraTurnTime;
27     float currentTurnTime;
28     float waitForNextTurnTime;
29     bool waitingForNextTurn{false};
30 };
31
32 #endif  // INC_4_WORMS_GAMECLOCK_H
```

jun 26, 18 17:16 **GameClock.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 10/06/18.
//

#include "GameClock.h"

GameClock::GameClock()
    : turnTime(Game::Config::getInstance().getTurnTime()),
      extraTurnTime(Game::Config::getInstance().getExtraTurnTime()),
      currentTurnTime(turnTime),
      waitForNextTurnTime(Game::Config::getInstance().getWaitForNextTurnTime())
{}

void GameClock::playerShot() {
    this→currentTurnTime = this→extraTurnTime;
    this→timeElapsed = 0.0f;
}

void GameClock::update(float dt) {
    this→timeElapsed += dt;
    if (this→timeElapsed > this→currentTurnTime) {
        if (this→waitingForNextTurn) {
            this→notify(*this, Event::NextTurn);
        } else {
            this→notify(*this, Event::EndTurn);
        }
    }
}

double GameClock::getTimeElapsed() const {
    return this→timeElapsed;
}

double GameClock::getTurnTime() const {
    return this→currentTurnTime;
}

void GameClock::restart() {
    this→waitingForNextTurn = false;
    this→timeElapsed = 0.0f;
    this→currentTurnTime = this→turnTime;
}

void GameClock::endTurn() {
    this→timeElapsed = this→currentTurnTime + 1.0f;
    this→notify(*this, Event::EndTurn);
}

void GameClock::waitForNextTurn() {
    this→timeElapsed = 0.0f;
    this→currentTurnTime = this→waitForNextTurnTime;
    this→waitingForNextTurn = true;
}
```

jun 26, 18 17:16 **ContactEventListener.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 20/05/18
 */

#ifndef __ContactEventListener_H__
#define __ContactEventListener_H__

#include "Box2D/Box2D.h"

class ContactEventListener : public b2ContactListener {
   public:
    ContactEventListener() = default;
    ~ContactEventListener() = default;

    void PreSolve(b2Contact* contact, const b2Manifold* oldManifold) override;
    void BeginContact(b2Contact* contact) override;
    void EndContact(b2Contact* contact) override;
};

#endif  //__ContactEventListener_H__
```

jun 26, 18 17:16 **ContactEventListener.cpp** Page 1/2

```
1  /*
2   *  Created by Federico Manuel Gomez Peter.
3   *  date: 20/05/18
4   */
5
6  #include <iostream>
7
8  #include "ContactEventListener.h"
9  #include "Player.h"
10
11 /**
12  * @brief Pre collision solver handler for Box2D. Notifies colliding objects so
    the can act
13  * appropriately.
14  *
15  * @param contact Collision contact.
16  * @param oldManifold Manifold.
17  */
18 void ContactEventListener::PreSolve(b2Contact *contact, const b2Manifold *oldMan
    ifold) {
19     Worms::PhysicsEntity *e1 =
20         static_cast<Worms::PhysicsEntity *>(contact→GetFixtureA()→GetBody()→Ge
    tUserData());
21     Worms::PhysicsEntity *e2 =
22         static_cast<Worms::PhysicsEntity *>(contact→GetFixtureB()→GetBody()→Ge
    tUserData());
23
24     if (¬e1 ∨ ¬e2) {
25         return;
26     }
27
28     e1→contactWith(*e2, *contact);
29     e2→contactWith(*e1, *contact);
30 }
31
32 void ContactEventListener::BeginContact(b2Contact *contact) {
33     Worms::PhysicsEntity *playerA =
34         static_cast<Worms::PhysicsEntity *>(contact→GetFixtureA()→GetBody()→Ge
    tUserData());
35     Worms::PhysicsEntity *playerB =
36         static_cast<Worms::PhysicsEntity *>(contact→GetFixtureB()→GetBody()→Ge
    tUserData());
37     /*
38      * If fixture A is a Worm, then call startContact. This will delegate
39      * the action to the internal state. For example, when a worm jump,
40      * it run with a state startJump, after a few seconds (so the clients
41      * could animate the impulse the worm takes to jump), it changes its
42      * state to Jumping. The moment the state changes to endJump will be
43      * when box2d detects a collision between the worm and the girder.
44      */
45     if (playerA) {
46         playerA→startContact(playerB);
47     }
48     if (playerB) {
49         playerB→startContact(playerA);
50     }
51
52     void *fixtureData = contact→GetFixtureA()→GetUserData();
53     if (fixtureData) {
54         Worms::PhysicsEntity *sensor = static_cast<Worms::TouchSensor *>(fixture
    Data);
55         sensor→startContact(playerB, *contact);
56     }
57
58     fixtureData = contact→GetFixtureB()→GetUserData();
59     if (fixtureData) {
```

jun 26, 18 17:16 **ContactEventListener.cpp** Page 2/2

```
60         Worms::PhysicsEntity *sensor = static_cast<Worms::TouchSensor *>(fixture
    Data);
61         sensor→startContact(playerA, *contact);
62     }
63 }
64
65 void ContactEventListener::EndContact(b2Contact *contact) {
66     Worms::PhysicsEntity *playerA =
67         static_cast<Worms::PhysicsEntity *>(contact→GetFixtureA()→GetBody()→Ge
    tUserData());
68     Worms::PhysicsEntity *playerB =
69         static_cast<Worms::PhysicsEntity *>(contact→GetFixtureB()→GetBody()→Ge
    tUserData());
70
71     if (playerA) {
72         playerA→endContact(playerB);
73     }
74     if (playerB) {
75         playerB→endContact(playerA);
76     }
77
78     void *fixtureData = contact→GetFixtureA()→GetUserData();
79     if (fixtureData) {
80         Worms::PhysicsEntity *sensor = static_cast<Worms::TouchSensor *>(fixture
    Data);
81         sensor→endContact(playerB, *contact);
82     }
83
84     fixtureData = contact→GetFixtureB()→GetUserData();
85     if (fixtureData) {
86         Worms::PhysicsEntity *sensor = static_cast<Worms::TouchSensor *>(fixture
    Data);
87         sensor→endContact(playerA, *contact);
88     }
89 }
```

jun 26, 18 17:16 **WindConfig.h** Page 1/1

```
1  /*
2   *  Created by Federico Manuel Gomez Peter.
3   *  date: 22/06/18
4   */
5
6  #ifndef __WIND_CONFIG_H__
7  #define __WIND_CONFIG_H__
8
9  #include "yaml-cpp/node/node.h"
10
11 namespace Config {
12 struct Wind {
13     float minIntensity;
14     float maxIntensity;
15     int xDirection;
16     float instensity;
17 };
18 }  // namespace Config
19
20 #endif  //__WIND_CONFIG_H__
```

jun 26, 18 17:16 **WeaponConfig.h** Page 1/1

```
1  /*
2   *  Created by Federico Manuel Gomez Peter.
3   *  date: 22/06/18
4   */
5
6  #ifndef __WeaponConfig_H__
7  #define __WeaponConfig_H__
8
9  #include <cstdint>
10 #include "yaml-cpp/node/node.h"
11
12 #include "BulletConfig.h"
13
14 namespace Config {
15 struct Weapon {
16     Bullet::DamageInfo dmgInfo;
17     float minAngle;
18     float maxAngle;
19     float angleStep;
20     std::uint16_t maxShotPower;
21     float restitution;
22     float friction;
23     std::uint8_t explotionInitialTimeout;
24     bool hasAfterExplode;
25     float bulletRadius;
26     float bulletDampingRatio;
27     bool windAffected;
28
29     explicit Weapon(const YAML::Node &config);
30 };
31 }  // namespace Config
32
33 #endif  //__WeaponConfig_H__
```

jun 26, 18 17:16 **WeaponConfig.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 22/06/18
 */

#include "WeaponConfig.h"
#include "ConfigDefines.h"

Config::Weapon::Weapon(const YAML::Node &config)
    : dmgInfo(config[BULLET][DAMAGE]),
      minAngle(config[ANGLE][MIN].as<float>()),
      maxAngle(config[ANGLE][MAX].as<float>()),
      angleStep(config[ANGLE][STEP].as<float>()),
      maxShotPower((std::uint16_t)config[MAX_SHOT_POWER].as<unsigned int>()),
      restitution(config[BULLET][RESTITUTION].as<float>()),
      friction(config[BULLET][FRICTION].as<float>()),
      explotionInitialTimeout(
          (std::uint8_t)config[BULLET][EXPLOTION_INITIAL_TIMEOUT].as<unsigned int>()),
      hasAfterExplode(config[HAS_AFTER_EXPLODE].as<bool>()),
      bulletRadius(config[BULLET][RADIUS].as<float>()),
      bulletDampingRatio(config[BULLET][DAMAGE][DAMPING_RATIO].as<float>()),
      windAffected(config[BULLET][WIND_AFFECTED].as<bool>()) {}
```

jun 26, 18 17:16 **P2PWeapon.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 22/06/18
 */

#ifndef __P2PWeapon_H__
#define __P2PWeapon_H__

#include <Direction.h>
#include <Point.h>

#include "BulletConfig.h"

namespace Config {
struct P2PWeapon {
    Bullet::DamageInfo dmgInfo;
    Worm::Direction direction;
    Math::Point<float> position;
    float angle;
};
}  // namespace Config

#endif  //__P2PWeapon_H__
```

jun 26, 18 17:16 **P2PWeapon.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 22/06/18
 */

#include "P2PWeapon.h"
```

jun 26, 18 17:16 **Config.h** Page 1/3

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 01/06/18
 */

#ifndef __GAMECONFIG_H__
#define __GAMECONFIG_H__

#include <stdint.h>
#include <mutex>

#include "Direction.h"
#include "Point.h"
#include "WeaponConfig.h"
#include "WindConfig.h"

#define NUM_TEAMS 2
#define GAME_HEIGHT 30.0f
#define GAME_WIDTH 30.0f
#define WORM_HEALTH 100

namespace Math {
using Vector = Math::Point<float>;
}

namespace Game {

/**
 *  Singleton class with all the game configuration (Velocity constants,
 *  Weapons attributes, etc)
 */
class Config {
   public:
    static Config &getInstance();
    ~Config() = default;

    const Math::Vector getJumpVelocity() const;
    const Math::Vector getBackflipVelocity() const;
    const float getStartJumpTime() const;
    const float getLandTime() const;

    const float getWalkVelocity() const;
    float getSafeFallDistance() const;
    float getMaxFallDamage() const;
    float getMinWindIntensity() const;
    float getMaxWindIntensity() const;

    const uint8_t getTurnTime() const;
    const float getExtraTurnTime() const;
    const float getWaitForNextTurnTime() const;
    const float getPowerChargeTime() const;
    const float getGameWidth() const;
    const float getGameHeight() const;
    const float getDyingTime() const;
    const float getDrowningTime() const;
    const float getBattingTime() const;
    const float getTeleportTime() const;
    const int getWaterLevel() const;
    const uint16_t getWormHealth() const;

    const ::Config::Weapon &getBazookaConfig() const;
    const ::Config::Weapon &getGreenGrenadeConfig() const;
    const uint8_t getClusterFragmentQuantity() const;
    const ::Config::Weapon &getClusterConfig() const;
    const ::Config::Weapon &getMortarConfig() const;
    const ::Config::Weapon &getBananaConfig() const;
```

```
67      const ::Config::Weapon &getHolyConfig() const;
68      const ::Config::Weapon &getClusterFragmentConfig() const;
69      const ::Config::Weapon &getMortarFragmentConfig() const;
70      const uint8_t getMortarFragmentQuantity() const;
71      const ::Config::Weapon &getAerialAttackConfig() const;
72      const ::Config::Weapon &getDynamiteConfig() const;
73      const uint8_t getAerialAttackMissileQuantity() const;
74      const float getAerialAttackMissileSeparation() const;
75      const float getAerialAttackLaunchHeight() const;
76      const ::Config::Weapon &getTeleportConfig() const;
77      const ::Config::Weapon &getBaseballBatConfig() const;
78
79    private:
80      /**
81       * Constructor hidden because is a singleton.
82       * TODO change constructor so it loads information from yaml file
83       */
84      //    Config();
85      explicit Config(const YAML::Node &node);
86      Config(Config &copy) = delete;
87      Config(Config ∧other) = delete;
88      Config &operator=(Config &copy) = delete;
89      Config &operator=(Config ∧other) = delete;
90
91      // jump
92      const Math::Vector jumpVelocity;
93      const Math::Vector backflipVelocity;
94      const float startJumpTime;
95      const float landTime;
96
97      // moving
98      const float walkVelocity;
99      // game
100     const float safeFallDistance;
101     const float maxFallDamage;
102     const std::uint8_t turnTime;
103     const float extraTurnTime;
104     const float waitForNextTurnTime;
105     const float powerChargeTime;
106     uint8_t numTeams{NUM_TEAMS};
107     float gameWidth{GAME_WIDTH};
108     float gameHeight{GAME_HEIGHT};
109     uint16_t wormHealth{WORM_HEALTH};
110     const float dyingTime;
111     const float drowningTime;
112     const float battingTime;
113     const float teleportTime;
114     const int waterLevel;
115     const float minWindIntensity;
116     const float maxWindIntensity;
117     // weapons
118     const ::Config::Weapon bazooka;
119     const ::Config::Weapon greenGrenade;
120     const ::Config::Weapon cluster;
121     const ::Config::Weapon clusterFragments;
122     const uint8_t clusterFragmentQuantity;
123     const ::Config::Weapon mortar;
124     const ::Config::Weapon mortarFragments;
125     const uint8_t mortarFragmentQuantity;
126     const ::Config::Weapon banana;
127     const ::Config::Weapon holy;
128     const uint8_t aerialAttackMissileQuantity;
129     const float aerialAttackMissileSeparation;
130     const ::Config::Weapon aerialAttack;
131     const float aerialAttackLaunchHeight;
132     const ::Config::Weapon dynamite;
```



```
133     const ::Config::Weapon teleport;
134     const ::Config::Weapon baseballBat;
135 };
136
137 void endTurn();
138 }  // namespace Game
139
140 #endif  //__GAMECONFIG_H__
```

jun 26, 18 17:16 **ConfigDefines.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 22/06/18
 */

#ifndef __CONFIG_DEFINES_H__
#define __CONFIG_DEFINES_H__

#define CONFIG_PATH "/etc/Worms/serverConfig.yaml"

#define JUMP "jump"
#define VELOCITY "velocity"
#define X "x"
#define Y "y"
#define BACKFLIP "backflip"
#define START_TIME "startTime"
#define LAND_TIME "landTime"
#define WALK "walk"
#define GAME "game"
#define SAFE_FALL_DISTANCE "safeFallDistance"
#define MAX_FALL_DAMAGE "maxFallDamage"
#define TURN_TIME "turnTime"
#define EXTRA_TURN_TIME "extraTurnTime"
#define WAIT_FOR_NEXT_TURN_TIME "waitForNextTurnTime"
#define POWER_CHARGE_MAX_TIME "powerChargeMaxTime"
#define DYING_TIME "dyingTime"
#define DROWNING_TIME "drowningTime"
#define BATTING_TIME "battingTime"
#define TELEPORT_TIME "teleportTime"
#define WATER_LEVEL "waterLevel"
#define WIND_INTENSITY "windIntensity"
#define MIN "min"
#define MAX "max"
#define BAZOOKA "bazooka"
#define GRENADE "grenade"
#define CLUSTER "cluster"
#define FRAGMENT "fragment"
#define QUANTITY "quantity"
#define MORTAR "mortar"
#define BANANA "banana"
#define HOLY "holy"
#define AERIAL_ATTACK "aerialAttack"
#define BULLET "bullet"
#define SEPARATION "separation"
#define LAUNCH_HEIGHT "launchHeight"
#define DYNAMITE "dynamite"
#define TELEPORT "teleport"
#define BASEBALL_BAT "baseballBat"

#define DAMAGE "damage"
#define RADIUS "radius"
#define IMPULSE_DAMPING_RATIO "impulseDampingRatio"
#define ANGLE "angle"
#define STEP "step"
#define MAX_SHOT_POWER "maxShotPower"
#define RESTITUTION "restitution"
#define FRICTION "friction"
#define EXPLOTION_INITIAL_TIMEOUT "explotionInitialTimeout"
#define HAS_AFTER_EXPLODE "hasAfterExplode"
#define DAMPING_RATIO "dampingRatio"
#define WIND_AFFECTED "windAffected"

#endif  //__CONFIG_DEFINES_H__
```

jun 26, 18 17:16 **Config.cpp** Page 1/4

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 01/06/18
 */

#include "Config.h"
#include <iostream>
#include "ConfigDefines.h"
#include "yaml-cpp/yaml.h"

/**
 * Meyer's singleton implementation.
 * @return
 */
Game::Config &Game::Config::getInstance() {
    static Config instance(YAML::LoadFile(CONFIG_PATH));
    return instance;
}

Game::Config::Config(const YAML::Node &node)
        : jumpVelocity(node[JUMP][VELOCITY][X].as<float>(), node[JUMP][VELOCITY][Y].as<float>()),
          backflipVelocity(node[BACKFLIP][VELOCITY][X].as<float>(),
                           node[BACKFLIP][VELOCITY][Y].as<float>()),
          startJumpTime(node[JUMP][START_TIME].as<float>()),
          landTime(node[JUMP][LAND_TIME].as<float>()),
          walkVelocity(node[WALK][VELOCITY].as<float>()),
          safeFallDistance(node[GAME][SAFE_FALL_DISTANCE].as<float>()),
          maxFallDamage(node[GAME][MAX_FALL_DAMAGE].as<float>()),
          turnTime((std::uint8_t)node[GAME][TURN_TIME].as<unsigned int>()),
          extraTurnTime(node[GAME][EXTRA_TURN_TIME].as<float>()),
          waitForNextTurnTime(node[GAME][WAIT_FOR_NEXT_TURN_TIME].as<float>()),
          powerChargeTime(node[GAME][POWER_CHARGE_MAX_TIME].as<float>()),
          dyingTime(node[GAME][DYING_TIME].as<float>()),
          drowningTime(node[GAME][DROWNING_TIME].as<float>()),
          battingTime(node[GAME][BATTING_TIME].as<float>()),
          teleportTime(node[GAME][TELEPORT_TIME].as<float>()),
          waterLevel(node[GAME][WATER_LEVEL].as<int>()),
          minWindIntensity(node[WIND_INTENSITY][MIN].as<float>()),
          maxWindIntensity(node[WIND_INTENSITY][MAX].as<float>()),
          bazooka(node[BAZOOKA]),
          greenGrenade(node[GRENADE]),
          cluster(node[CLUSTER]),
          clusterFragments(node[CLUSTER][FRAGMENT]),
          clusterFragmentQuantity((std::uint8_t)node[CLUSTER][FRAGMENT][QUANTITY].as<unsigned int>()),
          mortar(node[MORTAR]),
          mortarFragments(node[MORTAR][FRAGMENT]),
          mortarFragmentQuantity((std::uint8_t)node[MORTAR][FRAGMENT][QUANTITY].as<unsigned int>()),
          banana(node[BANANA]),
          holy(node[HOLY]),
          aerialAttackMissileQuantity(
                  (std::uint8_t)node[AERIAL_ATTACK][BULLET][QUANTITY].as<unsigned int>()),
          aerialAttackMissileSeparation(node[AERIAL_ATTACK][BULLET][SEPARATION].as<float>()),
          aerialAttack(node[AERIAL_ATTACK]),
          aerialAttackLaunchHeight(node[AERIAL_ATTACK][LAUNCH_HEIGHT].as<float>()),
          dynamite(node[DYNAMITE]),
          teleport(node[TELEPORT]),
          baseballBat(node[BASEBALL_BAT]) {}

float Game::Config::getSafeFallDistance() const {
    return this→safeFallDistance;
```

```
61  }
62
63  float Game::Config::getMaxFallDamage() const {
64      return this→maxFallDamage;
65  }
66
67  const Math::Vector Game::Config::getJumpVelocity() const {
68      return this→jumpVelocity;
69  }
70
71  const float Game::Config::getStartJumpTime() const {
72      return this→startJumpTime;
73  }
74
75  const float Game::Config::getLandTime() const {
76      return this→landTime;
77  }
78
79  const Math::Vector Game::Config::getBackflipVelocity() const {
80      return this→backflipVelocity;
81  }
82
83  const uint8_t Game::Config::getTurnTime() const {
84      return this→turnTime;
85  }
86
87  const float Game::Config::getGameWidth() const {
88      return this→gameWidth;
89  }
90
91  const float Game::Config::getGameHeight() const {
92      return this→gameHeight;
93  }
94
95  const uint16_t Game::Config::getWormHealth() const {
96      return this→wormHealth;
97  }
98
99  const Config::Weapon &Game::Config::getBazookaConfig() const {
100     return this→bazooka;
101 }
102
103 const float Game::Config::getDyingTime() const {
104     return this→dyingTime;
105 }
106
107 const float Game::Config::getDrowningTime() const {
108     return this→drowningTime;
109 }
110
111 const float Game::Config::getExtraTurnTime() const {
112     return this→extraTurnTime;
113 }
114
115 const int Game::Config::getWaterLevel() const {
116     return this→waterLevel;
117 }
118
119 const float Game::Config::getWalkVelocity() const {
120     return this→walkVelocity;
121 }
122
123 const Config::Weapon &Game::Config::getGreenGrenadeConfig() const {
124     return this→greenGrenade;
125 }
126
```



```
127 const Config::Weapon &Game::Config::getClusterConfig() const {
128     return this→cluster;
129 }
130
131 const Config::Weapon &Game::Config::getMortarConfig() const {
132     return this→mortar;
133 }
134
135 const Config::Weapon &Game::Config::getBananaConfig() const {
136     return this→banana;
137 }
138
139 const Config::Weapon &Game::Config::getHolyConfig() const {
140     return this→holy;
141 }
142
143 const float Game::Config::getPowerChargeTime() const {
144     return this→powerChargeTime;
145 }
146
147 const Config::Weapon &Game::Config::getClusterFragmentConfig() const {
148     return this→clusterFragments;
149 }
150
151 const uint8_t Game::Config::getClusterFragmentQuantity() const {
152     return this→clusterFragmentQuantity;
153 }
154
155 const Config::Weapon &Game::Config::getMortarFragmentConfig() const {
156     return this→mortarFragments;
157 }
158
159 const uint8_t Game::Config::getMortarFragmentQuantity() const {
160     return this→mortarFragmentQuantity;
161 }
162
163 const float Game::Config::getWaitForNextTurnTime() const {
164     return this→waitForNextTurnTime;
165 }
166
167 const Config::Weapon &Game::Config::getAerialAttackConfig() const {
168     return this→aerialAttack;
169 }
170
171 const uint8_t Game::Config::getAerialAttackMissileQuantity() const {
172     return this→aerialAttackMissileQuantity;
173 }
174
175 const float Game::Config::getAerialAttackMissileSeparation() const {
176     return this→aerialAttackMissileSeparation;
177 }
178
179 const float Game::Config::getAerialAttackLaunchHeight() const {
180     return this→aerialAttackLaunchHeight;
181 }
182
183 const float Game::Config::getBattingTime() const {
184     return this→battingTime;
185 }
186
187 const Config::Weapon &Game::Config::getTeleportConfig() const {
188     return this→teleport;
189 }
190
191 const float Game::Config::getTeleportTime() const {
192     return this→teleportTime;
```

jun 26, 18 17:16 **Config.cpp** Page 4/4

```
193 }
194
195 const Config::Weapon &Game::Config::getDynamiteConfig() const {
196     return this→dynamite;
197 }
198
199 const Config::Weapon &Game::Config::getBaseballBatConfig() const {
200     return this→baseballBat;
201 }
202
203 float Game::Config::getMinWindIntensity() const {
204     return this→minWindIntensity;
205 }
206
207 float Game::Config::getMaxWindIntensity() const {
208     return this→maxWindIntensity;
209 }
```

jun 26, 18 17:16 **BulletConfig.h** Page 1/1

```
1  /*
2   *  Created by Federico Manuel Gomez Peter.
3   *  date: 22/06/18
4   */
5
6  #ifndef __BULLET_CONFIG_H__
7  #define __BULLET_CONFIG_H__
8
9  #include <cstdint>
10 #include "yaml−cpp/yaml.h"
11
12 namespace Config {
13 namespace Bullet {
14 struct DamageInfo {
15     std::uint16_t damage;
16     float radius;
17     float impulseDampingRatio;
18
19     explicit DamageInfo(const YAML::Node &config);
20 };
21 }  // namespace Bullet
22 }  // namespace Bullet
23
24 #endif  //__BULLET_CONFIG_H__
```

jun 26, 18 17:16 **BulletConfig.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 22/06/18
 */

#include "BulletConfig.h"
#include "ConfigDefines.h"

Config::Bullet::DamageInfo::DamageInfo(const YAML::Node &config)
    : damage((std::uint16_t)config[DAMAGE].as<unsigned int>()),
      radius(config[RADIUS].as<float>()),
      impulseDampingRatio(config[IMPULSE_DAMPING_RATIO].as<float>()) {}
```

jun 26, 18 17:16 **stageelemshortgirder.h** Page 1/1

```cpp
#ifndef STAGEELEMSHORTGIRDER_H
#define STAGEELEMSHORTGIRDER_H

#include <stageelement.h>

class StageElemShortGirder : public StageElement {
   public:
    StageElemShortGirder(qreal opacity = 1.0);

    StageElement *clone();
    virtual bool canOverlap(StageElement *other);
    void serialize(StageData &sd);
};

#endif  // STAGEELEMSHORTGIRDER_H
```

jun 26, 18 17:16 **stageelemshortgirder.cpp** Page 1/1

```cpp
#include "stageelemshortgirder.h"

StageElemShortGirder::StageElemShortGirder(qreal opacity)
    : StageElement(":/assets/stage/short_girder.png", ItemType::ShortGirder, opacity) {}

StageElement *StageElemShortGirder::clone() {
    auto *e = new StageElemShortGirder;
    e→angle = this→angle;
    e→setRotation(this→angle);
    return e;
}

void StageElemShortGirder::serialize(StageData &sd) {
    sd.addShortGirder(this→getPosition(), this→getAngle());
}

bool StageElemShortGirder::canOverlap(StageElement *other) {
    return (other→getType() ≠ ItemType::Worm);
}
```

jun 26, 18 17:16 **stageelemlonggirder.h** Page 1/1

```cpp
#ifndef STAGEELEMLONGGIRDER_H
#define STAGEELEMLONGGIRDER_H

#include "stageelement.h"

class StageElemLongGirder : public StageElement {
   public:
    StageElemLongGirder(qreal opacity = 1.0);

    virtual StageElement *clone();
    virtual bool canOverlap(StageElement *other);
    virtual void serialize(StageData &sd);
};

#endif  // STAGEELEMLONGGIRDER_H
```

jun 26, 18 17:16 **stageelemlonggirder.cpp** Page 1/1

```cpp
#include "stageelemlonggirder.h"

StageElemLongGirder::StageElemLongGirder(qreal opacity)
    : StageElement(":/assets/stage/long_girder.png", ItemType::LongGirder, opacity) {}

StageElement *StageElemLongGirder::clone() {
    auto *e = new StageElemLongGirder;
    e→angle = this→angle;
    e→setRotation(this→angle);
    return e;
}

void StageElemLongGirder::serialize(StageData &sd) {
    sd.addLongGirder(this→getPosition(), this→getAngle());
}

bool StageElemLongGirder::canOverlap(StageElement *other) {
    return (other→getType() ≠ ItemType::Worm);
}
```

jun 26, 18 17:16 **stageelementworm.h** Page 1/1

```cpp
#ifndef STAGEELEMENTWORM_H
#define STAGEELEMENTWORM_H

#include "stageelement.h"

class StageElementWorm : public StageElement {
   public:
    StageElementWorm(qreal opacity = 1.0);

    virtual StageElement *clone();

    virtual void increaseAngle() override;
    virtual void decreaseAngle() override;
    virtual void serialize(StageData &sd);
};

#endif  // STAGEELEMENTWORM_H
```

jun 26, 18 17:16 **stageelementworm.cpp** Page 1/1

```cpp
#include "stageelement.h"
#include "stageelementworm.h"

StageElementWorm::StageElementWorm(qreal opacity)
    : StageElement(":/assets/stage/worm.png", ItemType::Worm, opacity) {}

void StageElementWorm::increaseAngle() {}

void StageElementWorm::decreaseAngle() {}

StageElement *StageElementWorm::clone() {
    auto *e = new StageElementWorm;
    e→angle = this→angle;
    return e;
}

void StageElementWorm::serialize(StageData &sd) {
    sd.addWorm(this→getPosition());
}
```

jun 26, 18 17:16 **stageelement.h** Page 1/1

```cpp
#ifndef STAGEELEMENT_H
#define STAGEELEMENT_H

#include <QGraphicsItem>
#include <QGraphicsPixmapItem>
#include <QObject>
#include <QWidget>
#include <QtDebug>
#include <string>
#include "stagedata.h"

enum class ItemType {
    Worm,
    ShortGirder,
    LongGirder,
};

class StageElement : public QGraphicsPixmapItem {
   public:
    StageElement(const std::string &resource, ItemType type, qreal opacity);

    ItemType getType();

    qreal getAngle() const;
    QPointF getPosition() const;

    virtual StageElement *clone() = 0;
    virtual bool canOverlap(StageElement *other);

    virtual void increaseAngle();
    virtual void decreaseAngle();

    void setRotationEnabled(bool);

    virtual void serialize(StageData &sd) = 0;

   protected:
    QPixmap getResource(qreal opacity = 1.0);
    qreal angle{0.0f};
    ItemType type;

   private:
    std::string resource;
};

#endif  // STAGEELEMENT_H
```

jun 26, 18 17:16 **stageelement.cpp** Page 1/1

```cpp
#include "stageelement.h"
#include <QPainter>

const double PI = 3.141592653589793;

StageElement::StageElement(const std::string &resource, ItemType type, qreal opacity)
    : QGraphicsPixmapItem(nullptr), type(type), resource(resource) {
    this→setPixmap(this→getResource(opacity));
    this→setTransformOriginPoint(this→pixmap().width() / 2, this→pixmap().height() / 2);
    this→setFlag(QGraphicsItem::ItemIsMovable);
}

ItemType StageElement::getType() {
    return this→type;
}

qreal StageElement::getAngle() const {
    return this→angle;
}

QPointF StageElement::getPosition() const {
    qreal hw = this→pixmap().width() / 2.0;
    qreal hh = this→pixmap().height() / 2.0;
    return QPointF{this→pos().x() + hw, this→pos().y() + hh};
}

void StageElement::increaseAngle() {
    this→angle += 90.0f / 10.0f;

    if (this→angle > 90.0f) {
        this→angle = 90.0f;
    }

    this→setRotation(this→angle);
}

void StageElement::decreaseAngle() {
    this→angle −= 90.0f / 10.0f;

    if (this→angle < −90.0f) {
        this→angle = −90.0f;
    }

    this→setRotation(this→angle);
}

QPixmap StageElement::getResource(qreal opacity) {
    QImage image;
    image.load(this→resource.c_str());
    image = image.convertToFormat(QImage::Format_ARGB32);

    QImage image2(image.size(), QImage::Format_ARGB32);
    image2.fill(Qt::transparent);

    QPainter painter(&image2);
    painter.setOpacity(opacity);
    painter.drawImage(image.rect(), image);

    return QPixmap::fromImage(image2);
}

bool StageElement::canOverlap(StageElement *) {
    return false;
}
```

jun 26, 18 17:16 **stagedata.h** Page 1/1

```cpp
#ifndef STAGEDATA_H
#define STAGEDATA_H

#include <QColor>
#include <QDebug>
#include <QPoint>
#include <QString>
#include <Qt>
#include <iostream>
#include <vector>
#include <map>
#include "yaml−cpp/yaml.h"

struct GirderData {
    QPointF position;
    qreal angle;
    qreal length;
};

struct WormData {
    QPointF position;
};

class StageData {
   public:
    QString fartherBgFile;
    QString medianBgFile;
    QString closeBgFile;
    QColor bgColor;
    int wormsHealth;
    int numPlayers;

    StageData(qreal width, qreal height);

    void dump(std::ostream &output, std::string fileName);

    std::size_t numWorms() const;

    void addWorm(QPointF position);
    void addShortGirder(QPointF position, qreal angle);
    void addLongGirder(QPointF position, qreal angle);
    void addWeaponAmmo(QString weaponName, int ammo);

   private:
    QPointF toGameCoords(const QPointF &point) const;

    qreal width;
    qreal height;
    std::vector<GirderData> girders;
    std::vector<WormData> worms;
    std::map<std::string, int> weapons;
};

#endif  // STAGEDATA_H
```

jun 26, 18 17:16 **stagedata.cpp** Page 1/2

```cpp
#include "stagedata.h"
#include <QDebug>
#include <cassert>

const qreal scale = 13.0;

YAML::Emitter& operator<<(YAML::Emitter& out, const QColor& v) {
    out << YAML::Flow;
    out << YAML::BeginSeq << v.red() << v.green() << v.blue() << YAML::EndSeq;
    return out;
}

YAML::Emitter& operator<<(YAML::Emitter& out, const QPointF& v) {
    out << YAML::Flow;
    out << YAML::BeginSeq << v.x() << v.y() << YAML::EndSeq;
    return out;
}

YAML::Emitter& operator<<(YAML::Emitter& out, const WormData& v) {
    out << YAML::BeginMap;

    out << YAML::Key << "position";
    out << YAML::Value << v.position;

    out << YAML::EndMap;
    return out;
}

YAML::Emitter& operator<<(YAML::Emitter& out, const GirderData& v) {
    out << YAML::BeginMap;

    out << YAML::Key << "position";
    out << YAML::Value << v.position;

    out << YAML::Key << "angle";
    out << YAML::Value << v.angle;

    out << YAML::Key << "length";
    out << YAML::Value << v.length;

    out << YAML::EndMap;
    return out;
}

StageData::StageData(qreal width, qreal height) : width(width / scale), height(height / scale) {}

QPointF StageData::toGameCoords(const QPointF& point) const {
    qreal xpos = (point.x() / scale - this→width / 2.0);
    qreal ypos = this→height - point.y() / scale;
    return QPointF(xpos, ypos);
}

std::size_t StageData::numWorms() const {
    return this→worms.size();
}

void StageData::dump(std::ostream& output, std::string fileName) {
    YAML::Emitter emitter;

    emitter << YAML::BeginMap;

    emitter << YAML::Key << "name";
    emitter << YAML::Value << fileName;

    emitter << YAML::Key << "numPlayers";
```

jun 26, 18 17:16 **stagedata.cpp** Page 2/2

```cpp
    emitter << YAML::Value << this→numPlayers;

    emitter << YAML::Key << "weaponsAmmo";
    emitter << YAML::Value << this→weapons;

    emitter << YAML::Key << "width";
    emitter << YAML::Value << this→width;

    emitter << YAML::Key << "height";
    emitter << YAML::Value << this→height;

    emitter << YAML::Key << "wormsHealth";
    emitter << YAML::Value << this→wormsHealth;

    emitter << YAML::Key << "worms";
    emitter << YAML::Value << this→worms;

    emitter << YAML::Key << "girders";
    emitter << YAML::Value << this→girders;

    emitter << YAML::Key << "background";
    emitter << YAML::Value;
    {
        emitter << YAML::BeginMap;
        emitter << YAML::Key << "closeBackgroundFile";
        emitter << YAML::Value << this→closeBgFile.toStdString();

        emitter << YAML::Key << "midBackgroundFile";
        emitter << YAML::Value << this→medianBgFile.toStdString();

        emitter << YAML::Key << "fartherBackgroundFile";
        emitter << YAML::Value << this→fartherBgFile.toStdString();

        emitter << YAML::Key << "color";
        emitter << YAML::Value << this→bgColor;

        emitter << YAML::EndMap;
    }

    emitter << YAML::EndMap;

    assert(emitter.good());

    output << emitter.c_str();
}

void StageData::addWorm(QPointF position) {
    this→worms.push_back(WormData{this→toGameCoords(position)});
}

void StageData::addShortGirder(QPointF position, qreal angle) {
    this→girders.push_back(GirderData{this→toGameCoords(position), -angle, 5.3845});
}

void StageData::addLongGirder(QPointF position, qreal angle) {
    this→girders.push_back(GirderData{this→toGameCoords(position), -angle, 10.769});
}

void StageData::addWeaponAmmo(QString weaponName, int ammo) {
    this→weapons[weaponName.toStdString()] = ammo;
}
```

jun 26, 18 17:16 **qgraphicsitemlayer.h** Page 1/1

```
1  #ifndef QGRAPHICSITEMLAYER_H
2  #define QGRAPHICSITEMLAYER_H
3
4  #include <QGraphicsItem>
5  #include <QObject>
6
7  class QGraphicsItemLayer : public QGraphicsItem {
8     public:
9      QGraphicsItemLayer();
10
11     virtual QRectF boundingRect() const;
12     virtual void paint(QPainter *, const QStyleOptionGraphicsItem *, QWidget *);
13 };
14
15 #endif  // QGRAPHICSITEMLAYER_H
```

jun 26, 18 17:16 **qgraphicsitemlayer.cpp** Page 1/1

```
1  #include "qgraphicsitemlayer.h"
2
3  QGraphicsItemLayer::QGraphicsItemLayer() : QGraphicsItem(nullptr) {}
4
5  QRectF QGraphicsItemLayer::boundingRect() const {
6      return QRectF(0, 0, 0, 0);
7  }
8
9  void QGraphicsItemLayer::paint(QPainter *, const QStyleOptionGraphicsItem *, QWi
   dget *) {}
```

jun 26, 18 17:16 **mainwindow.h** Page 1/1

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QGraphicsScene>
#include <QGraphicsView>
#include <QMainWindow>
#include <QString>
#include "editorscene.h"

namespace Ui {

class MainWindow;
}

class MainWindow : public QMainWindow {
    Q_OBJECT

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

   private slots:
    void on_actionLejano_triggered();

    void on_actionMedio_triggered();

    void on_actionCercano_triggered();

    void on_bgColorButton_clicked();

    void on_actionOpen_triggered();

   private:
    Ui::MainWindow *ui;
    QRectF stageSize{0, 0, 13 * 250, 13 * 250};
    EditorScene *scene;
    QString closeBgFile;
    QString midBgFile;
    QString fartherBgFile;
};

#endif  // MAINWINDOW_H
```

jun 26, 18 17:16 **mainwindow.cpp** Page 1/2

```
#include "mainwindow.h"
#include <QColor>
#include <QColorDialog>
#include <QErrorMessage>
#include <QFileDialog>
#include <fstream>
#include "stagedata.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow) {
    ui→setupUi(this);

    this→scene = new EditorScene{this→stageSize};
    this→ui→editorView→setScene(this→scene);

    this→ui→colorPreview→setScene(new QGraphicsScene);

    /* toolbar */
    connect(this→ui→actionAdd_Worm, SIGNAL(triggered(bool)), this→ui→editorView,
            SLOT(setWorm()));
    connect(this→ui→actionAdd_Long_Girder, SIGNAL(triggered(bool)), this→ui→editorView,
            SLOT(setLongGirder()));
    connect(this→ui→actionShort_Girder, SIGNAL(triggered(bool)), this→ui→editorView,
            SLOT(setShortGirder()));

    QColor defaultColor{0xba, 0x8d, 0xc6};
    this→scene→setBgColor(defaultColor);
    this→ui→colorPreview→setBackgroundBrush(QBrush(defaultColor));

    this→showMaximized();
}

MainWindow::~MainWindow() {
    delete ui;
    delete scene;
}

void MainWindow::on_actionLejano_triggered() {
    QString fileName = QFileDialog::getOpenFileName(
        this, tr("Seleccione una imagen para el fondo lejano"), "/home", tr("Image Files (*.png)"));
    if (¬fileName.isEmpty()) {
        this→fartherBgFile = fileName;
        this→scene→setFartherBg(QImage(fileName));
    }
}

void MainWindow::on_actionMedio_triggered() {
    QString fileName = QFileDialog::getOpenFileName(
        this, tr("Seleccione una imagen para el fondo medio"), "/home", tr("Image Files (*.png)"));
    if (¬fileName.isEmpty()) {
        this→midBgFile = fileName;
        this→scene→setMedianBg(QImage(fileName));
    }
}

void MainWindow::on_actionCercano_triggered() {
    QString fileName =
        QFileDialog::getOpenFileName(this, tr("Seleccione una imagen para el fondo cercano"),
                                     "/home", tr("Image Files (*.png)"));
```

jun 26, 18 17:16 **mainwindow.cpp** Page 2/2

```
60      if (¬fileName.isEmpty()) {
61          this→closeBgFile = fileName;
62          this→scene→setCloserBg(QImage(fileName));
63      }
64  }
65
66  void MainWindow::on_bgColorButton_clicked() {
67      QColor color = QColorDialog::getColor(Qt::white, this);
68      if (color.isValid()) {
69          this→scene→setBgColor(color);
70          this→ui→colorPreview→setBackgroundBrush(QBrush(color));
71      }
72  }
73
74  void MainWindow::on_actionOpen_triggered() {
75      /* serializes the stage */
76      StageData sd{this→stageSize.width(), stageSize.height()};
77
78      sd.closeBgFile = this→closeBgFile;
79      sd.medianBgFile = this→midBgFile;
80      sd.fartherBgFile = this→fartherBgFile;
81      sd.wormsHealth = this→ui→wormsHP→value();
82      sd.numPlayers= this→ui→numPlayers→value();
83
84      /* weapon ammo */
85      const QString WEAPON_PREFIX = "wpn_";
86      for(auto *child : this→ui→stageParams→children()) {
87          if(child→objectName().startsWith(WEAPON_PREFIX)) {
88              QString weaponName = child→objectName().remove(0, WEAPON_PREFIX.siz
    e());
89
90              QSpinBox *widget = dynamic_cast<QSpinBox *>(child);
91              sd.addWeaponAmmo(weaponName, widget→value());
92          }
93      }
94
95      this→ui→editorView→serialize(sd);
96
97      if (static_cast<int>(sd.numWorms()) < sd.numPlayers) {
98          QErrorMessage::qtHandler()→showMessage("Se necesita al menos 1 worm por jugador");
99          return;
100     }
101
102     /* gets the output file name */
103     QString fileName = QFileDialog::getSaveFileName(this, tr("Nombre de archivo de sali
    da"),
104                                                     "/home", tr("YAML (*.yml)"));
105
106     /* checks if a file was selected */
107     if (fileName.isEmpty()) {
108         return;
109     }
110
111     std::ofstream file;
112     file.open(fileName.toStdString(), std::ios::out | std::ios::trunc);
113     if (¬file) {
114         QErrorMessage::qtHandler()→showMessage("Error al abrir el archivo");
115         return;
116     }
117     /* gets the base name of the file */
118     QStringList list = fileName.split('/');
119     QList<QString>::Iterator it = list.end();
120     it--;
121     QStringList list2 = it→split('.');
122     sd.dump(file,list2[0].toStdString());
123 }
```

jun 26, 18 17:16 **main.cpp** Page 1/1

```
1   #include <QApplication>
2   #include "mainwindow.h"
3
4   int main(int argc, char *argv[]) {
5       QApplication a(argc, argv);
6       MainWindow w;
7       w.show();
8
9       return a.exec();
10  }
```

jun 26, 18 17:16 **editorview.h** Page 1/1

```
#ifndef EDITORVIEW_H
#define EDITORVIEW_H

#include <QEvent>
#include <QGraphicsView>
#include <QObject>
#include <QWheelEvent>
#include <QWidget>
#include "editorscene.h"
#include "stagedata.h"
#include "stageelement.h"

class EditorView : public QGraphicsView {
    Q_OBJECT

   public:
    EditorView(QWidget *parent);
    virtual void setScene(EditorScene *scene);

    void drawCloseBg(QString &fileName);

   public slots:
    void setWorm();
    void setShortGirder();
    void setLongGirder();

    void serialize(StageData &sd) const;

    // QWidget interface
   protected:
    void mousePressEvent(QMouseEvent *event);
    void mouseReleaseEvent(QMouseEvent *);
    void mouseMoveEvent(QMouseEvent *);
    void wheelEvent(QWheelEvent *);
    bool event(QEvent *event);
    void hoverEvent(QHoverEvent *event);
    void keyPressEvent(QKeyEvent *event);

    bool collides();
    void deleteAt(QPoint pos);
    void createAt(QPoint pos);

   private:
    StageElement *stageElem{nullptr};
    EditorScene *escene{nullptr};
};

#endif  // EDITORVIEW_H
```

jun 26, 18 17:16 **editorview.cpp** Page 1/3

```
#include "editorview.h"
#include <QGraphicsPixmapItem>
#include <QImage>
#include <QtDebug>
#include <QScrollBar>
#include <cmath>
#include "stageelementworm.h"
#include "stageelemlonggirder.h"
#include "stageelemshortgirder.h"

const qreal cursorOpacity = 0.7;

EditorView::EditorView(QWidget *parent) : QGraphicsView(parent) {
    this→setAttribute(Qt::WA_Hover, true);
    this→setTransformationAnchor(QGraphicsView::AnchorUnderMouse);
    this→setLongGirder();
}

void EditorView::drawCloseBg(QString &) {}

void EditorView::setScene(EditorScene *scene) {
    QGraphicsView::setScene(scene);
    this→escene = scene;
    this→horizontalScrollBar()→setValue(this→horizontalScrollBar()→maximum()
/ 2);
    this→verticalScrollBar()→setValue(this→verticalScrollBar()→maximum());
}

void EditorView::setWorm() {
    if (this→stageElem) {
        delete this→stageElem;
    }

    this→stageElem = new StageElementWorm{cursorOpacity};
}

void EditorView::setShortGirder() {
    if (this→stageElem) {
        delete this→stageElem;
    }

    this→stageElem = new StageElemShortGirder{cursorOpacity};
}

void EditorView::setLongGirder() {
    if (this→stageElem) {
        delete this→stageElem;
    }

    this→stageElem = new StageElemLongGirder{cursorOpacity};
}

void EditorView::mousePressEvent(QMouseEvent *) {}

void EditorView::deleteAt(QPoint pos) {
    this→scene()→removeItem(this→stageElem);
    QGraphicsItem *item = this→itemAt(pos);
    if (item) {
        this→escene→removeItem(static_cast<StageElement *>(item));
    }
    this→escene→addItem(this→stageElem);
}

void EditorView::keyPressEvent(QKeyEvent *event) {
    if (event→key() ≡ Qt::Key_Plus) {
        this→stageElem→increaseAngle();
```

jun 26, 18 17:16 **editorview.cpp** Page 2/3

```
66      } else if (event→key() ≡ Qt::Key_Minus) {
67          this→stageElem→decreaseAngle();
68      }
69  }
70
71  void EditorView::createAt(QPoint pos) {
72      if (¬this→stageElem) {
73          return;
74      }
75
76      if (this→collides()) {
77          return;
78      }
79
80      StageElement *newElem = this→stageElem→clone();
81
82      QPointF lpos = this→mapToScene(pos);
83      lpos.rx() −= newElem→pixmap().width() / 2;
84      lpos.ry() −= newElem→pixmap().height() / 2;
85
86      newElem→setPos(lpos);
87      this→escene→addItem(newElem);
88  }
89
90  void EditorView::mouseReleaseEvent(QMouseEvent *event) {
91      event→accept();
92
93      if (event→button() & Qt::RightButton) {
94          this→deleteAt(event→pos());
95      } else {
96          this→createAt(event→pos());
97          this→stageElem→setZValue(1);
98      }
99  }
100
101 void EditorView::mouseMoveEvent(QMouseEvent *event) {
102     if (¬this→stageElem) {
103         return;
104     }
105
106     /* set the position of the hint image under the mouse */
107     QPointF pos = this→mapToScene(event→pos());
108     pos.rx() −= this→stageElem→pixmap().width() / 2;
109     pos.ry() −= this→stageElem→pixmap().height() / 2;
110
111     this→stageElem→setPos(pos);
112     event→accept();
113 }
114
115 bool EditorView::event(QEvent *event) {
116     switch (event→type()) {
117         case QEvent::HoverEnter:
118             if (this→stageElem) {
119                 this→setFocus();
120                 this→escene→addItem(dynamic_cast<QGraphicsItem *>(this→stageEl
    em));
121             }
122             return true;
123         case QEvent::HoverLeave:
124             if (this→stageElem) {
125                 this→escene→removeItem(dynamic_cast<QGraphicsItem *>(this→stag
    eElem));
126             }
127             return true;
128         default:
129             break;
```

jun 26, 18 17:16 **editorview.cpp** Page 3/3

```
130     }
131
132     return QGraphicsView::event(event);
133 }
134
135 void EditorView::wheelEvent(QWheelEvent *event) {
136     static qreal factor = 1.1;
137
138     if (event→delta() > 0) {
139         this→scale(factor, factor);
140     } else {
141         this→scale(1.0 / factor, 1.0 / factor);
142     }
143
144     /* set the position of the hint image under the mouse */
145     QPointF pos = this→mapToScene(event→pos());
146     pos.rx() −= this→stageElem→pixmap().width() / 2;
147     pos.ry() −= this→stageElem→pixmap().height() / 2;
148
149     this→stageElem→setPos(pos);
150     event→accept();
151 }
152
153 bool EditorView::collides() {
154     for (StageElement *other : this→escene→collidingItems(this→stageElem)) {
155         if (¬this→stageElem→canOverlap(other)) {
156             return true;
157         }
158     }
159
160     return false;
161 }
162
163 void EditorView::serialize(StageData &sd) const {
164     if (this→stageElem) {
165         this→escene→removeItem(this→stageElem);
166     }
167
168     this→escene→serialize(sd);
169
170     if (this→stageElem) {
171         this→escene→addItem(this→stageElem);
172     }
173 }
```

jun 26, 18 17:16 **editorscene.h** Page 1/1

```
#ifndef EDITORSCENE_H
#define EDITORSCENE_H

#include <QColor>
#include <QGraphicsScene>
#include <QImage>
#include <QObject>
#include <QWidget>
#include <set>
#include <string>
#include "qgraphicsitemlayer.h"
#include "stageelement.h"

class EditorScene : public QGraphicsScene {
    Q_OBJECT

   public:
    EditorScene(QRectF rect);

    void setCursor(StageElement *newCursor);
    void hideCursor();
    void showCursor();

    void addItem(QGraphicsItem *elem);
    void addItem(StageElement *elem);
    void removeItem(QGraphicsItem *elem);
    void removeItem(StageElement *elem);

    virtual QList<StageElement *> collidingItems(StageElement *elem);
    bool contains(StageElement *elem);

    void serialize(StageData &sd);

    /* background */
    void setBgColor(QColor color);
    void setFartherBg(QImage image);
    void setMedianBg(QImage image);
    void setCloserBg(QImage image);

   private:
    void setBackground(QImage image, QGraphicsItemLayer **layerPtr, qreal zValue
);

    QRectF rect;
    QColor bgColor{Qt::white};
    QGraphicsItemLayer *closeBg{nullptr};
    QGraphicsItemLayer *medianBg{nullptr};
    QGraphicsItemLayer *fartherBg{nullptr};
    QGraphicsItemLayer *bgColorLayer{nullptr};
    StageElement *cursor{nullptr};
    std::string resource;
    std::set<StageElement *> elements;
};

#endif  // EDITORSCENE_H
```

jun 26, 18 17:16 **editorscene.cpp** Page 1/3

```
#include "editorscene.h"
#include <QDebug>
#include <QGraphicsPixmapItem>
#include <QImage>
#include <QMouseEvent>
#include <QPainter>

EditorScene::EditorScene(QRectF rect) : QGraphicsScene(nullptr), rect(rect) {
    this→setSceneRect(rect);
}

void EditorScene::setCursor(StageElement *newCursor) {
    if (this→cursor) {
        delete this→cursor;
    }
    this→cursor = newCursor;
    QGraphicsScene::addItem(this→cursor);
}

void EditorScene::hideCursor() {
    if (this→cursor) {
        QGraphicsScene::removeItem(this→cursor);
    }
}

void EditorScene::showCursor() {
    if (this→cursor) {
        QGraphicsScene::addItem(this→cursor);
    }
}

void EditorScene::addItem(QGraphicsItem *elem) {
    if (elem→scene() ≠ this) {
        QGraphicsScene::addItem(elem);
    }
}

void EditorScene::addItem(StageElement *elem) {
    if (elem→scene() ≠ this) {
        if (¬this→rect.contains(elem→getPosition())) {
            return;
        }
        QGraphicsScene::addItem(elem);
        this→elements.insert(elem);
    }
}

void EditorScene::removeItem(StageElement *elem) {
    if (this→contains(elem)) {
        this→elements.erase(this→elements.find(elem));

        if (elem→scene()) {
            QGraphicsScene::removeItem(elem);
        }
    }
}

void EditorScene::removeItem(QGraphicsItem *elem) {
    if (elem→scene()) {
        QGraphicsScene::removeItem(elem);
    }
}

void EditorScene::serialize(StageData &sd) {
    sd.bgColor = this→bgColor;
    for (auto *elem : this→elements) {
```

```
67          elem→serialize(sd);
68      }
69  }
70
71  QList<StageElement *> EditorScene::collidingItems(StageElement *elem) {
72      QList<StageElement *> rv;
73      for (QGraphicsItem *other : QGraphicsScene::collidingItems(elem)) {
74          if (other ≡ elem) {
75              continue;
76          }
77
78          if (this→contains(dynamic_cast<StageElement *>(other))) {
79              rv.append(dynamic_cast<StageElement *>(other));
80          }
81      }
82      return rv;
83  }
84
85  bool EditorScene::contains(StageElement *elem) {
86      auto it = this→elements.find(elem);
87      return (it ≠ this→elements.end());
88  }
89
90  void EditorScene::setBgColor(QColor color) {
91      this→bgColor = color;
92      if (this→bgColorLayer) {
93          this→removeItem(this→bgColorLayer);
94          delete this→bgColorLayer;
95      }
96
97      this→bgColorLayer = new QGraphicsItemLayer;
98      this→bgColorLayer→setZValue(−4);
99      this→addItem(this→bgColorLayer);
100     QGraphicsRectItem *bg = new QGraphicsRectItem(this→rect, this→bgColorLayer
    );
101     bg→setBrush(QBrush{color});
102 }
103
104 void EditorScene::setFartherBg(QImage image) {
105     this→setBackground(image, &this→fartherBg, −3);
106 }
107
108 void EditorScene::setMedianBg(QImage image) {
109     this→setBackground(image, &this→medianBg, −2);
110 }
111
112 void EditorScene::setCloserBg(QImage image) {
113     this→setBackground(image, &this→closeBg, −1);
114 }
115
116 void EditorScene::setBackground(QImage image, QGraphicsItemLayer **layerPtr, qre
    al zValue) {
117     if (*layerPtr) {
118         this→removeItem(*layerPtr);
119         delete *layerPtr;
120     }
121
122     *layerPtr = new QGraphicsItemLayer;
123     QGraphicsItemLayer *layer = *layerPtr;
124
125     layer→setZValue(zValue);
126     this→addItem(layer);
127
128     for (int i = 0; i < this→rect.width() / image.width() + 1; i++) {
129         QGraphicsPixmapItem *pix = new QGraphicsPixmapItem{layer};
130         pix→setPixmap(QPixmap::fromImage(image));
```



```
131         pix→setPos(image.width() * i, this→rect.height() − image.height());
132     }
133 }
```

jun 26, 18 17:16 **editor.h** Page 1/1

```
#ifndef EDITOR_H
#define EDITOR_H

#include <QGraphicsView>
#include <QWheelEvent>

namespace Editor {
class Editor : public QGraphicsView {
   public:
    Editor(QWidget *parent);
    ~Editor();

    void wheelEvent(QWheelEvent *event);
    void mousePressEvent(QMouseEvent *event);
};
}

// Q_DECLARE_METATYPE(Editor::Editor);

#endif  // EDITOR_H
```

jun 26, 18 17:16 **editor.cpp** Page 1/1

```
#include "editor.h"

Editor::Editor::Editor(QWidget *parent) : QGraphicsView(parent) {}

Editor::Editor::~Editor() {}

void Editor::Editor::wheelEvent(QWheelEvent *event) {
    QGraphicsView::wheelEvent(event);
    if (event→isAccepted()) {
        return;
    }

    static qreal factor = 1.1;

    if (event→angleDelta().y() > 0) {
        scale(factor, factor);
    } else {
        scale(1 / factor, 1 / factor);
    }

    event→accept();
}

void Editor::Editor::mousePressEvent(QMouseEvent *event) {
    QGraphicsView::mousePressEvent(event);
    if (event→isAccepted()) {
        return;
    }

    switch (event→button()) {
        case Qt::LeftButton:
            break;

        case Qt::RightButton:
            break;

        default:
            break;
    }

    event→accept();
}
```

jun 26, 18 17:16 **WormWalk.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 18/05/18
 */

#ifndef __WORM_WALK_H__
#define __WORM_WALK_H__

#include <SDL2/SDL_system.h>

#include "../Worm.h"
#include "GameStateMsg.h"
#include "WormState.h"

namespace Worm {
class Walk : public State {
   public:
    explicit Walk();
    virtual ~Walk();

    virtual void update(float dt) override;

    virtual IO::PlayerInput moveRight(Worm &w) override;
    virtual IO::PlayerInput moveLeft(Worm &w) override;
    virtual IO::PlayerInput stopMove(Worm &w) override;
    virtual IO::PlayerInput jump(Worm &w) override;
    virtual IO::PlayerInput backFlip(Worm &w) override;
    virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;

    virtual IO::PlayerInput bazooka(Worm &w) override;
    virtual IO::PlayerInput grenade(Worm &w) override;
    virtual IO::PlayerInput cluster(Worm &w) override;
    virtual IO::PlayerInput mortar(Worm &w) override;
    virtual IO::PlayerInput banana(Worm &w) override;
    virtual IO::PlayerInput holy(Worm &w) override;
    virtual IO::PlayerInput aerialAttack(Worm &w) override;
    virtual IO::PlayerInput dynamite(Worm &w) override;
    virtual IO::PlayerInput baseballBat(Worm &w) override;
    virtual IO::PlayerInput teleport(Worm &w) override;
    virtual IO::PlayerInput positionSelected(Worm &w) override;

    virtual IO::PlayerInput startShot(Worm &w) override;
    virtual IO::PlayerInput endShot(Worm &w) override;
    virtual IO::PlayerInput pointUp(Worm &w) override;
    virtual IO::PlayerInput pointDown(Worm &w) override;
};
}  // namespace Worm

#endif  //__WORM_WALK_H__
```

jun 26, 18 17:16 **WormWalk.cpp** Page 1/2

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 18/05/18
 */

#include "WormWalk.h"
#include <iostream>

Worm::Walk::Walk() : State(StateID::Walk) {}

Worm::Walk::~Walk() {}

void Worm::Walk::update(float dt) {}

IO::PlayerInput Worm::Walk::moveLeft(Worm &w) {
    if (w.direction ≡ Direction::left) {
        return IO::PlayerInput::moveNone;
    }
    return IO::PlayerInput::moveLeft;
}

IO::PlayerInput Worm::Walk::moveRight(Worm &w) {
    if (w.direction ≡ Direction::right) {
        return IO::PlayerInput::moveNone;
    }
    return IO::PlayerInput::moveRight;
}

IO::PlayerInput Worm::Walk::stopMove(Worm &w) {
    return IO::PlayerInput::stopMove;
}

IO::PlayerInput Worm::Walk::jump(Worm &w) {
    return IO::PlayerInput::startJump;
}

IO::PlayerInput Worm::Walk::backFlip(Worm &w) {
    return IO::PlayerInput::startBackFlip;
}

IO::PlayerInput Worm::Walk::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Walk::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Walk::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Walk::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Walk::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Walk::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Walk::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
```

jun 26, 18 17:16 **WormWalk.cpp** Page 2/2

```
67  }
68
69  IO::PlayerInput Worm::Walk::mortar(Worm &w) {
70      return IO::PlayerInput::moveNone;
71  }
72
73  IO::PlayerInput Worm::Walk::banana(Worm &w) {
74      return IO::PlayerInput::moveNone;
75  }
76
77  IO::PlayerInput Worm::Walk::holy(Worm &w) {
78      return IO::PlayerInput::moveNone;
79  }
80
81  IO::PlayerInput Worm::Walk::setTimeoutTo(Worm &w, int t) {
82      return IO::PlayerInput::moveNone;
83  }
84
85  IO::PlayerInput Worm::Walk::aerialAttack(Worm &w) {
86      return IO::PlayerInput::moveNone;
87  }
88
89  IO::PlayerInput Worm::Walk::positionSelected(Worm &w) {
90      return IO::PlayerInput::moveNone;
91  }
92
93  IO::PlayerInput Worm::Walk::dynamite(Worm &w) {
94      return IO::PlayerInput::moveNone;
95  }
96
97  IO::PlayerInput Worm::Walk::teleport(Worm &w) {
98      return IO::PlayerInput::moveNone;
99  }
100
101 IO::PlayerInput Worm::Walk::baseballBat(Worm &w) {
102     return IO::PlayerInput::moveNone;
103 }
```

jun 26, 18 17:16 **WormStill.h** Page 1/1

```
1   /*
2    *  Created by Federico Manuel Gomez Peter.
3    *  date: 18/05/18
4    */
5
6   #ifndef __WORM_QUIET_H__
7   #define __WORM_QUIET_H__
8
9   #include <SDL2/SDL_system.h>
10
11  #include "../Worm.h"
12  #include "GameStateMsg.h"
13  #include "WormState.h"
14
15  namespace Worm {
16  class Still : public State {
17     public:
18      Still();
19      ~Still();
20      virtual void update(float dt) override;
21      virtual IO::PlayerInput moveRight(Worm &w) override;
22      virtual IO::PlayerInput moveLeft(Worm &w) override;
23      virtual IO::PlayerInput stopMove(Worm &w) override;
24      virtual IO::PlayerInput jump(Worm &w) override;
25      virtual IO::PlayerInput backFlip(Worm &w) override;
26      virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;
27
28      virtual IO::PlayerInput bazooka(Worm &w) override;
29      virtual IO::PlayerInput grenade(Worm &w) override;
30      virtual IO::PlayerInput cluster(Worm &w) override;
31      virtual IO::PlayerInput mortar(Worm &w) override;
32      virtual IO::PlayerInput banana(Worm &w) override;
33      virtual IO::PlayerInput holy(Worm &w) override;
34      virtual IO::PlayerInput aerialAttack(Worm &w) override;
35      virtual IO::PlayerInput dynamite(Worm &w) override;
36      virtual IO::PlayerInput baseballBat(Worm &w) override;
37      virtual IO::PlayerInput teleport(Worm &w) override;
38      virtual IO::PlayerInput positionSelected(Worm &w) override;
39
40      virtual IO::PlayerInput startShot(Worm &w) override;
41      virtual IO::PlayerInput endShot(Worm &w) override;
42      virtual IO::PlayerInput pointUp(Worm &w) override;
43      virtual IO::PlayerInput pointDown(Worm &w) override;
44  };
45  }  // namespace Worm
46
47  #endif  //__WORM_QUIET_H__
```

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 18/05/18
 */

#include "WormStill.h"
#include <iostream>
#include "Texture.h"

Worm::Still::Still() : State(StateID::Still) {}

Worm::Still::~Still() {}

void Worm::Still::update(float dt) {}

IO::PlayerInput Worm::Still::moveRight(Worm &w) {
    return IO::PlayerInput::moveRight;
}

IO::PlayerInput Worm::Still::moveLeft(Worm &w) {
    return IO::PlayerInput::moveLeft;
}

IO::PlayerInput Worm::Still::stopMove(Worm &w) {
    return IO::PlayerInput::stopMove;
}

IO::PlayerInput Worm::Still::jump(Worm &w) {
    return IO::PlayerInput::startJump;
}

IO::PlayerInput Worm::Still::backFlip(Worm &w) {
    return IO::PlayerInput::startBackFlip;
}

IO::PlayerInput Worm::Still::bazooka(Worm &w) {
    return IO::PlayerInput::bazooka;
}

IO::PlayerInput Worm::Still::pointUp(Worm &w) {
    return IO::PlayerInput::pointUp;
}

IO::PlayerInput Worm::Still::pointDown(Worm &w) {
    return IO::PlayerInput::pointDown;
}

IO::PlayerInput Worm::Still::startShot(Worm &w) {
    w.startShot();
    return IO::PlayerInput::startShot;
}

IO::PlayerInput Worm::Still::endShot(Worm &w) {
    w.endShot();
    return IO::PlayerInput::endShot;
}

IO::PlayerInput Worm::Still::grenade(Worm &w) {
    return IO::PlayerInput::grenade;
}

IO::PlayerInput Worm::Still::cluster(Worm &w) {
    return IO::PlayerInput::cluster;
}

IO::PlayerInput Worm::Still::mortar(Worm &w) {
```



```
    return IO::PlayerInput::mortar;
}

IO::PlayerInput Worm::Still::banana(Worm &w) {
    return IO::PlayerInput::banana;
}

IO::PlayerInput Worm::Still::holy(Worm &w) {
    return IO::PlayerInput::holy;
}

IO::PlayerInput Worm::Still::setTimeoutTo(Worm &w, int time) {
    switch (time) {
        case 1:
            return IO::PlayerInput::timeout1;
        case 2:
            return IO::PlayerInput::timeout2;
        case 3:
            return IO::PlayerInput::timeout3;
        case 4:
            return IO::PlayerInput::timeout4;
        case 5:
            return IO::PlayerInput::timeout5;
        default:
            return IO::PlayerInput::moveNone;
    }
}

IO::PlayerInput Worm::Still::aerialAttack(Worm &w) {
    return IO::PlayerInput::aerialAttack;
}

IO::PlayerInput Worm::Still::positionSelected(Worm &w) {
    return IO::PlayerInput::positionSelected;
}

IO::PlayerInput Worm::Still::dynamite(Worm &w) {
    return IO::PlayerInput::dynamite;
}

IO::PlayerInput Worm::Still::teleport(Worm &w) {
    return IO::PlayerInput::teleport;
}

IO::PlayerInput Worm::Still::baseballBat(Worm &w) {
    return IO::PlayerInput::baseballBat;
}
```

jun 26, 18 17:16 **WormState.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 18/05/18
 */

#ifndef __WORM_STATE_H__
#define __WORM_STATE_H__

#include "Animation.h"
#include "GameStateMsg.h"

namespace Worm {

class Worm;
/**
 * Worm status interface. It is used to implement the state pattern and
 * thus obtain a polymorphic behavior and at the same time treat the
 * animation as a state machine
 */
class State {
   public:
    State(StateID stateID) : stateID(stateID){};
    virtual ~State() = default;

    virtual void update(float dt) = 0;

    virtual IO::PlayerInput moveRight(Worm &w) = 0;
    virtual IO::PlayerInput moveLeft(Worm &w) = 0;
    virtual IO::PlayerInput stopMove(Worm &w) = 0;
    virtual IO::PlayerInput pointUp(Worm &w) = 0;
    virtual IO::PlayerInput pointDown(Worm &w) = 0;
    virtual IO::PlayerInput jump(Worm &w) = 0;
    virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) = 0;

    virtual IO::PlayerInput bazooka(Worm &w) = 0;
    virtual IO::PlayerInput grenade(Worm &w) = 0;
    virtual IO::PlayerInput cluster(Worm &w) = 0;
    virtual IO::PlayerInput mortar(Worm &w) = 0;
    virtual IO::PlayerInput banana(Worm &w) = 0;
    virtual IO::PlayerInput holy(Worm &w) = 0;
    virtual IO::PlayerInput aerialAttack(Worm &w) = 0;
    virtual IO::PlayerInput dynamite(Worm &w) = 0;
    virtual IO::PlayerInput baseballBat(Worm &w) = 0;
    virtual IO::PlayerInput teleport(Worm &w) = 0;

    virtual IO::PlayerInput startShot(Worm &w) = 0;
    virtual IO::PlayerInput endShot(Worm &w) = 0;
    virtual IO::PlayerInput backFlip(Worm &w) = 0;
    virtual IO::PlayerInput positionSelected(Worm &w) = 0;

    virtual StateID &getState() {
        return this→stateID;
    };

   protected:
    StateID stateID;
};
}  // namespace Worm

#endif  //__WORM_STATE_H__
```

jun 26, 18 17:16 **WormStartJump.h** Page 1/1

```
/*
 *  Created by Rodrigo.
 *  date: 19/05/18
 */

#ifndef __WORM_START_JUMP_H__
#define __WORM_START_JUMP_H__

#include "../Worm.h"
#include "GameStateMsg.h"
#include "WormState.h"

namespace Worm {
class StartJump : public State {
   public:
    StartJump();
    ~StartJump();

    virtual void update(float dt) override;

    virtual IO::PlayerInput moveRight(Worm &w) override;
    virtual IO::PlayerInput moveLeft(Worm &w) override;
    virtual IO::PlayerInput stopMove(Worm &w) override;
    virtual IO::PlayerInput jump(Worm &w) override;
    virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;

    virtual IO::PlayerInput bazooka(Worm &w) override;
    virtual IO::PlayerInput grenade(Worm &w) override;
    virtual IO::PlayerInput cluster(Worm &w) override;
    virtual IO::PlayerInput mortar(Worm &w) override;
    virtual IO::PlayerInput banana(Worm &w) override;
    virtual IO::PlayerInput holy(Worm &w) override;
    virtual IO::PlayerInput aerialAttack(Worm &w) override;
    virtual IO::PlayerInput dynamite(Worm &w) override;
    virtual IO::PlayerInput baseballBat(Worm &w) override;
    virtual IO::PlayerInput teleport(Worm &w) override;
    virtual IO::PlayerInput positionSelected(Worm &w) override;

    virtual IO::PlayerInput startShot(Worm &w) override;
    virtual IO::PlayerInput endShot(Worm &w) override;
    virtual IO::PlayerInput pointUp(Worm &w) override;
    virtual IO::PlayerInput pointDown(Worm &w) override;
    virtual IO::PlayerInput backFlip(Worm &w) override;
};
}  // namespace Worm

#endif  //__WORM_START_JUMP_H__
```

```cpp
/*
 *  Created by Rodrigo.
 *  date: 19/05/18
 */

#include "WormStartJump.h"

Worm::StartJump::StartJump() : State(StateID::StartJump) {}

Worm::StartJump::~StartJump() {}

void Worm::StartJump::update(float dt) {}

IO::PlayerInput Worm::StartJump::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::banana(Worm &w) {
```



```cpp
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::holy(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::setTimeoutTo(Worm &w, int t) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::aerialAttack(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::positionSelected(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::dynamite(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::teleport(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::StartJump::baseballBat(Worm &w) {
    return IO::PlayerInput::moveNone;
}
```

jun 26, 18 17:16 **WormJumping.h** Page 1/1

```
/*
 *  Created by Rodrigo.
 *  date: 21/05/18
 */

#ifndef __JUMPING_H__
#define __JUMPING_H__

#include "../Worm.h"
#include "GameStateMsg.h"

namespace Worm {
class Jumping : public State {
   public:
    explicit Jumping();
    virtual ~Jumping();

    virtual void update(float dt) override;

    virtual IO::PlayerInput moveRight(Worm &w) override;
    virtual IO::PlayerInput moveLeft(Worm &w) override;
    virtual IO::PlayerInput stopMove(Worm &w) override;
    virtual IO::PlayerInput jump(Worm &w) override;
    virtual IO::PlayerInput backFlip(Worm &w) override;
    virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;

    virtual IO::PlayerInput bazooka(Worm &w) override;
    virtual IO::PlayerInput grenade(Worm &w) override;
    virtual IO::PlayerInput cluster(Worm &w) override;
    virtual IO::PlayerInput mortar(Worm &w) override;
    virtual IO::PlayerInput banana(Worm &w) override;
    virtual IO::PlayerInput holy(Worm &w) override;
    virtual IO::PlayerInput aerialAttack(Worm &w) override;
    virtual IO::PlayerInput dynamite(Worm &w) override;
    virtual IO::PlayerInput baseballBat(Worm &w) override;
    virtual IO::PlayerInput teleport(Worm &w) override;
    virtual IO::PlayerInput positionSelected(Worm &w) override;

    virtual IO::PlayerInput startShot(Worm &w) override;
    virtual IO::PlayerInput endShot(Worm &w) override;
    virtual IO::PlayerInput pointUp(Worm &w) override;
    virtual IO::PlayerInput pointDown(Worm &w) override;
};
}  // namespace Worm

#endif  //__JUMPING_H__
```

jun 26, 18 17:16 **WormJumping.cpp** Page 1/2

```
/*
 *  Created by Rodrigo.
 *  date: 21/05/18
 */

#include "WormJumping.h"

Worm::Jumping::Jumping() : State(StateID::Jumping) {}

Worm::Jumping::~Jumping() {}

void Worm::Jumping::update(float dt) {}

IO::PlayerInput Worm::Jumping::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Jumping::banana(Worm &w) {
```

jun 26, 18 17:16 **WormJumping.cpp** Page 2/2

```
67      return IO::PlayerInput::moveNone;
68  }
69
70  IO::PlayerInput Worm::Jumping::holy(Worm &w) {
71      return IO::PlayerInput::moveNone;
72  }
73
74  IO::PlayerInput Worm::Jumping::setTimeoutTo(Worm &w, int t) {
75      return IO::PlayerInput::moveNone;
76  }
77
78  IO::PlayerInput Worm::Jumping::aerialAttack(Worm &w) {
79      return IO::PlayerInput::moveNone;
80  }
81
82  IO::PlayerInput Worm::Jumping::positionSelected(Worm &w) {
83      return IO::PlayerInput::moveNone;
84  }
85
86  IO::PlayerInput Worm::Jumping::dynamite(Worm &w) {
87      return IO::PlayerInput::moveNone;
88  }
89
90  IO::PlayerInput Worm::Jumping::teleport(Worm &w) {
91      return IO::PlayerInput::moveNone;
92  }
93
94  IO::PlayerInput Worm::Jumping::baseballBat(Worm &w) {
95      return IO::PlayerInput::moveNone;
96  }
```

jun 26, 18 17:16 **WormEndJump.h** Page 1/1

```
1   /*
2    *  Created by Rodrigo.
3    *  date: 21/05/18
4    */
5
6   #ifndef __END_JUMP_H__
7   #define __END_JUMP_H__
8
9   #include "GameStateMsg.h"
10  #include "WormState.h"
11
12  namespace Worm {
13  class EndJump : public State {
14     public:
15      EndJump();
16      ~EndJump();
17
18      virtual void update(float dt) override;
19
20      virtual IO::PlayerInput moveRight(Worm &w) override;
21      virtual IO::PlayerInput moveLeft(Worm &w) override;
22      virtual IO::PlayerInput stopMove(Worm &w) override;
23      virtual IO::PlayerInput jump(Worm &w) override;
24      virtual IO::PlayerInput backFlip(Worm &w) override;
25      virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;
26
27      virtual IO::PlayerInput bazooka(Worm &w) override;
28      virtual IO::PlayerInput grenade(Worm &w) override;
29      virtual IO::PlayerInput cluster(Worm &w) override;
30      virtual IO::PlayerInput mortar(Worm &w) override;
31      virtual IO::PlayerInput banana(Worm &w) override;
32      virtual IO::PlayerInput holy(Worm &w) override;
33      virtual IO::PlayerInput aerialAttack(Worm &w) override;
34      virtual IO::PlayerInput dynamite(Worm &w) override;
35      virtual IO::PlayerInput baseballBat(Worm &w) override;
36      virtual IO::PlayerInput teleport(Worm &w) override;
37      virtual IO::PlayerInput positionSelected(Worm &w) override;
38
39      virtual IO::PlayerInput startShot(Worm &w) override;
40      virtual IO::PlayerInput endShot(Worm &w) override;
41      virtual IO::PlayerInput pointUp(Worm &w) override;
42      virtual IO::PlayerInput pointDown(Worm &w) override;
43  };
44  }  // namespace Worm
45
46  #endif  //__END_JUMP_H__
```

jun 26, 18 17:16 **WormEndJump.cpp** Page 1/2

```cpp
/*
 *  Created by Rodrigo.
 *  date: 21/05/18
 */

#include "WormEndJump.h"

Worm::EndJump::EndJump() : State(StateID::EndJump) {}

Worm::EndJump::~EndJump() {}

void Worm::EndJump::update(float dt) {}

IO::PlayerInput Worm::EndJump::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::banana(Worm &w) {
```

jun 26, 18 17:16 **WormEndJump.cpp** Page 2/2

```cpp
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::holy(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::setTimeoutTo(Worm &w, int t) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::aerialAttack(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::positionSelected(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::dynamite(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::teleport(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndJump::baseballBat(Worm &w) {
    return IO::PlayerInput::moveNone;
}
```

jun 26, 18 17:16 **WormEndBackFlip.h** Page 1/1

```
/*
 *  Created by Rodrigo.
 *  date: 21/05/18
 */

#ifndef __WORM_END_BACKFLIP_H__
#define __WORM_END_BACKFLIP_H__

#include "GameStateMsg.h"
#include "WormState.h"

namespace Worm {
class EndBackFlip : public State {
   public:
    EndBackFlip();
    ~EndBackFlip();

    virtual void update(float dt) override;

    virtual IO::PlayerInput moveRight(Worm &w) override;
    virtual IO::PlayerInput moveLeft(Worm &w) override;
    virtual IO::PlayerInput stopMove(Worm &w) override;
    virtual IO::PlayerInput jump(Worm &w) override;
    virtual IO::PlayerInput backFlip(Worm &w) override;
    virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;

    virtual IO::PlayerInput bazooka(Worm &w) override;
    virtual IO::PlayerInput grenade(Worm &w) override;
    virtual IO::PlayerInput cluster(Worm &w) override;
    virtual IO::PlayerInput mortar(Worm &w) override;
    virtual IO::PlayerInput banana(Worm &w) override;
    virtual IO::PlayerInput holy(Worm &w) override;
    virtual IO::PlayerInput aerialAttack(Worm &w) override;
    virtual IO::PlayerInput dynamite(Worm &w) override;
    virtual IO::PlayerInput baseballBat(Worm &w) override;
    virtual IO::PlayerInput teleport(Worm &w) override;
    virtual IO::PlayerInput positionSelected(Worm &w) override;

    virtual IO::PlayerInput endShot(Worm &w) override;
    virtual IO::PlayerInput startShot(Worm &w) override;
    virtual IO::PlayerInput pointUp(Worm &w) override;
    virtual IO::PlayerInput pointDown(Worm &w) override;
};
}  // namespace Worm

#endif  //__WORM_END_BACKFLIP_H__
```

jun 26, 18 17:16 **WormEndBackFlip.cpp** Page 1/2

```
/*
 *  Created by Rodrigo.
 *  date: 21/05/18
 */

#include "WormEndBackFlip.h"

Worm::EndBackFlip::EndBackFlip() : State(StateID::EndBackFlip) {}

Worm::EndBackFlip::~EndBackFlip() {}

void Worm::EndBackFlip::update(float dt) {}

IO::PlayerInput Worm::EndBackFlip::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::EndBackFlip::banana(Worm &w) {
```

jun 26, 18 17:16 **WormEndBackFlip.cpp** Page 2/2

```
67      return IO::PlayerInput::moveNone;
68  }
69
70  IO::PlayerInput Worm::EndBackFlip::holy(Worm &w) {
71      return IO::PlayerInput::moveNone;
72  }
73
74  IO::PlayerInput Worm::EndBackFlip::setTimeoutTo(Worm &w, int t) {
75      return IO::PlayerInput::moveNone;
76  }
77
78  IO::PlayerInput Worm::EndBackFlip::aerialAttack(Worm &w) {
79      return IO::PlayerInput::moveNone;
80  }
81
82  IO::PlayerInput Worm::EndBackFlip::positionSelected(Worm &w) {
83      return IO::PlayerInput::moveNone;
84  }
85
86  IO::PlayerInput Worm::EndBackFlip::dynamite(Worm &w) {
87      return IO::PlayerInput::moveNone;
88  }
89
90  IO::PlayerInput Worm::EndBackFlip::teleport(Worm &w) {
91      return IO::PlayerInput::moveNone;
92  }
93
94  IO::PlayerInput Worm::EndBackFlip::baseballBat(Worm &w) {
95      return IO::PlayerInput::moveNone;
96  }
```

jun 26, 18 17:16 **WormBackFlipping.h** Page 1/1

```
1   /*
2    *  Created by Rodrigo.
3    *  date: 21/05/18
4    */
5
6   #ifndef __WORM_BACK_FLIPPING_H__
7   #define __WORM_BACK_FLIPPING_H__
8
9   #include "GameStateMsg.h"
10  #include "WormState.h"
11
12  namespace Worm {
13  class BackFlipping : public State {
14     public:
15      BackFlipping();
16      ~BackFlipping();
17
18      virtual void update(float dt) override;
19
20      virtual IO::PlayerInput moveRight(Worm &w) override;
21      virtual IO::PlayerInput moveLeft(Worm &w) override;
22      virtual IO::PlayerInput stopMove(Worm &w) override;
23      virtual IO::PlayerInput jump(Worm &w) override;
24      virtual IO::PlayerInput backFlip(Worm &w) override;
25      virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;
26
27      virtual IO::PlayerInput bazooka(Worm &w) override;
28      virtual IO::PlayerInput grenade(Worm &w) override;
29      virtual IO::PlayerInput cluster(Worm &w) override;
30      virtual IO::PlayerInput mortar(Worm &w) override;
31      virtual IO::PlayerInput banana(Worm &w) override;
32      virtual IO::PlayerInput holy(Worm &w) override;
33      virtual IO::PlayerInput aerialAttack(Worm &w) override;
34      virtual IO::PlayerInput dynamite(Worm &w) override;
35      virtual IO::PlayerInput baseballBat(Worm &w) override;
36
37      virtual IO::PlayerInput teleport(Worm &w) override;
38      virtual IO::PlayerInput positionSelected(Worm &w) override;
39      virtual IO::PlayerInput startShot(Worm &w) override;
40      virtual IO::PlayerInput endShot(Worm &w) override;
41      virtual IO::PlayerInput pointUp(Worm &w) override;
42      virtual IO::PlayerInput pointDown(Worm &w) override;
43  };
44  }  // namespace Worm
45
46  #endif  //__WORM_BACK_FLIPPING_H__
```

```cpp
/*
 *  Created by Rodrigo.
 *  date: 21/05/18
 */

#include "WormBackFlipping.h"

Worm::BackFlipping::BackFlipping() : State(StateID::BackFlipping) {}

Worm::BackFlipping::~BackFlipping() {}

void Worm::BackFlipping::update(float dt) {}

IO::PlayerInput Worm::BackFlipping::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::banana(Worm &w) {
```



```cpp
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::holy(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::setTimeoutTo(Worm &w, int t) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::aerialAttack(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::positionSelected(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::dynamite(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::teleport(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlipping::baseballBat(Worm &w) {
    return IO::PlayerInput::moveNone;
}
```

jun 26, 18 17:16 **Teleporting.h** Page 1/1

```
1  //
2  // Created by rodrigo on 16/06/18.
3  //
4
5  #ifndef INC_4_WORMS_TELEPORTING_H
6  #define INC_4_WORMS_TELEPORTING_H
7
8  #include "WormState.h"
9
10 namespace Worm {
11 class Teleporting : public State {
12    public:
13     Teleporting();
14     ~Teleporting();
15
16     virtual void update(float dt) override;
17
18     virtual IO::PlayerInput moveRight(Worm &w) override;
19     virtual IO::PlayerInput moveLeft(Worm &w) override;
20     virtual IO::PlayerInput stopMove(Worm &w) override;
21     virtual IO::PlayerInput jump(Worm &w) override;
22     virtual IO::PlayerInput backFlip(Worm &w) override;
23     virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;
24
25     virtual IO::PlayerInput bazooka(Worm &w) override;
26     virtual IO::PlayerInput grenade(Worm &w) override;
27     virtual IO::PlayerInput cluster(Worm &w) override;
28     virtual IO::PlayerInput mortar(Worm &w) override;
29     virtual IO::PlayerInput banana(Worm &w) override;
30     virtual IO::PlayerInput holy(Worm &w) override;
31     virtual IO::PlayerInput aerialAttack(Worm &w) override;
32     virtual IO::PlayerInput dynamite(Worm &w) override;
33     virtual IO::PlayerInput baseballBat(Worm &w) override;
34     virtual IO::PlayerInput teleport(Worm &w) override;
35     virtual IO::PlayerInput positionSelected(Worm &w) override;
36
37     virtual IO::PlayerInput startShot(Worm &w) override;
38     virtual IO::PlayerInput endShot(Worm &w) override;
39     virtual IO::PlayerInput pointUp(Worm &w) override;
40     virtual IO::PlayerInput pointDown(Worm &w) override;
41 };
42 }  // namespace Worm
43
44 #endif  // INC_4_WORMS_TELEPORTING_H
```

jun 26, 18 17:16 **Teleporting.cpp** Page 1/2

```
1  //
2  // Created by rodrigo on 16/06/18.
3  //
4
5  #include "Teleporting.h"
6
7  Worm::Teleporting::Teleporting() : State(StateID::Teleporting) {}
8
9  Worm::Teleporting::~Teleporting() {}
10
11 void Worm::Teleporting::update(float dt) {}
12
13 IO::PlayerInput Worm::Teleporting::moveRight(Worm &w) {
14     return IO::PlayerInput::moveNone;
15 }
16
17 IO::PlayerInput Worm::Teleporting::moveLeft(Worm &w) {
18     return IO::PlayerInput::moveNone;
19 }
20
21 IO::PlayerInput Worm::Teleporting::stopMove(Worm &w) {
22     return IO::PlayerInput::moveNone;
23 }
24
25 IO::PlayerInput Worm::Teleporting::jump(Worm &w) {
26     return IO::PlayerInput::moveNone;
27 }
28
29 IO::PlayerInput Worm::Teleporting::backFlip(Worm &w) {
30     return IO::PlayerInput::moveNone;
31 }
32
33 IO::PlayerInput Worm::Teleporting::bazooka(Worm &w) {
34     return IO::PlayerInput::moveNone;
35 }
36
37 IO::PlayerInput Worm::Teleporting::pointUp(Worm &w) {
38     return IO::PlayerInput::moveNone;
39 }
40
41 IO::PlayerInput Worm::Teleporting::pointDown(Worm &w) {
42     return IO::PlayerInput::moveNone;
43 }
44
45 IO::PlayerInput Worm::Teleporting::startShot(Worm &w) {
46     return IO::PlayerInput::moveNone;
47 }
48
49 IO::PlayerInput Worm::Teleporting::endShot(Worm &w) {
50     return IO::PlayerInput::moveNone;
51 }
52
53 IO::PlayerInput Worm::Teleporting::grenade(Worm &w) {
54     return IO::PlayerInput::moveNone;
55 }
56
57 IO::PlayerInput Worm::Teleporting::cluster(Worm &w) {
58     return IO::PlayerInput::moveNone;
59 }
60
61 IO::PlayerInput Worm::Teleporting::mortar(Worm &w) {
62     return IO::PlayerInput::moveNone;
63 }
64
65 IO::PlayerInput Worm::Teleporting::banana(Worm &w) {
66     return IO::PlayerInput::moveNone;
```

jun 26, 18 17:16 **Teleporting.cpp** Page 2/2

```
67  }
68
69  IO::PlayerInput Worm::Teleporting::holy(Worm &w) {
70      return IO::PlayerInput::moveNone;
71  }
72
73  IO::PlayerInput Worm::Teleporting::setTimeoutTo(Worm &w, int t) {
74      return IO::PlayerInput::moveNone;
75  }
76
77  IO::PlayerInput Worm::Teleporting::aerialAttack(Worm &w) {
78      return IO::PlayerInput::moveNone;
79  }
80
81  IO::PlayerInput Worm::Teleporting::dynamite(Worm &w) {
82      return IO::PlayerInput::moveNone;
83  }
84
85  IO::PlayerInput Worm::Teleporting::positionSelected(Worm &w) {
86      return IO::PlayerInput::moveNone;
87  }
88
89  IO::PlayerInput Worm::Teleporting::teleport(Worm &w) {
90      return IO::PlayerInput::moveNone;
91  }
92
93  IO::PlayerInput Worm::Teleporting::baseballBat(Worm &w) {
94      return IO::PlayerInput::moveNone;
95  }
```

jun 26, 18 17:16 **Teleported.h** Page 1/1

```
1   //
2   // Created by rodrigo on 16/06/18.
3   //
4
5   #ifndef INC_4_WORMS_TELEPORTED_H
6   #define INC_4_WORMS_TELEPORTED_H
7
8   #include "WormState.h"
9
10  namespace Worm {
11  class Teleported : public State {
12     public:
13      Teleported();
14      ~Teleported();
15
16      virtual void update(float dt) override;
17
18      virtual IO::PlayerInput moveRight(Worm &w) override;
19      virtual IO::PlayerInput moveLeft(Worm &w) override;
20      virtual IO::PlayerInput stopMove(Worm &w) override;
21      virtual IO::PlayerInput jump(Worm &w) override;
22      virtual IO::PlayerInput backFlip(Worm &w) override;
23      virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;
24
25      virtual IO::PlayerInput bazooka(Worm &w) override;
26      virtual IO::PlayerInput grenade(Worm &w) override;
27      virtual IO::PlayerInput cluster(Worm &w) override;
28      virtual IO::PlayerInput mortar(Worm &w) override;
29      virtual IO::PlayerInput banana(Worm &w) override;
30      virtual IO::PlayerInput holy(Worm &w) override;
31      virtual IO::PlayerInput aerialAttack(Worm &w) override;
32      virtual IO::PlayerInput dynamite(Worm &w) override;
33      virtual IO::PlayerInput baseballBat(Worm &w) override;
34      virtual IO::PlayerInput teleport(Worm &w) override;
35      virtual IO::PlayerInput positionSelected(Worm &w) override;
36
37      virtual IO::PlayerInput startShot(Worm &w) override;
38      virtual IO::PlayerInput endShot(Worm &w) override;
39      virtual IO::PlayerInput pointUp(Worm &w) override;
40      virtual IO::PlayerInput pointDown(Worm &w) override;
41  };
42  }  // namespace Worm
43
44  #endif  // INC_4_WORMS_TELEPORTED_H
```

jun 26, 18 17:16 **Teleported.cpp** Page 1/2

```
//
// Created by rodrigo on 16/06/18.
//

#include "Teleported.h"

Worm::Teleported::Teleported() : State(StateID::Teleported) {}

Worm::Teleported::~Teleported() {}

void Worm::Teleported::update(float dt) {}

IO::PlayerInput Worm::Teleported::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::banana(Worm &w) {
    return IO::PlayerInput::moveNone;
```

jun 26, 18 17:16 **Teleported.cpp** Page 2/2

```
}

IO::PlayerInput Worm::Teleported::holy(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::setTimeoutTo(Worm &w, int t) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::aerialAttack(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::dynamite(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::positionSelected(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::teleport(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Teleported::baseballBat(Worm &w) {
    return IO::PlayerInput::moveNone;
}
```

jun 26, 18 17:16 **Sliding.h** Page 1/1

```
#ifndef PLAYER_SLIDING_H_
#define PLAYER_SLIDING_H_

#include "WormState.h"

namespace Worm {
class Sliding : public State {
   public:
    Sliding();
    ~Sliding();

    virtual void update(float dt) override;

    virtual IO::PlayerInput moveRight(Worm &w) override;
    virtual IO::PlayerInput moveLeft(Worm &w) override;
    virtual IO::PlayerInput stopMove(Worm &w) override;
    virtual IO::PlayerInput jump(Worm &w) override;
    virtual IO::PlayerInput backFlip(Worm &w) override;
    virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;

    virtual IO::PlayerInput bazooka(Worm &w) override;
    virtual IO::PlayerInput grenade(Worm &w) override;
    virtual IO::PlayerInput cluster(Worm &w) override;
    virtual IO::PlayerInput mortar(Worm &w) override;
    virtual IO::PlayerInput banana(Worm &w) override;
    virtual IO::PlayerInput holy(Worm &w) override;
    virtual IO::PlayerInput aerialAttack(Worm &w) override;
    virtual IO::PlayerInput dynamite(Worm &w) override;
    virtual IO::PlayerInput baseballBat(Worm &w) override;
    virtual IO::PlayerInput teleport(Worm &w) override;

    virtual IO::PlayerInput endShot(Worm &w) override;
    virtual IO::PlayerInput startShot(Worm &w) override;
    virtual IO::PlayerInput pointUp(Worm &w) override;
    virtual IO::PlayerInput pointDown(Worm &w) override;
    virtual IO::PlayerInput positionSelected(Worm &w) override;
};
}  // namespace Worm

#endif  // INC_4_WORMS_FALLING_H
```

jun 26, 18 17:16 **Sliding.cpp** Page 1/2

```
#include "Sliding.h"

Worm::Sliding::Sliding() : State(StateID::Sliding) {}

Worm::Sliding::~Sliding() {}

void Worm::Sliding::update(float dt) {}

IO::PlayerInput Worm::Sliding::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::banana(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Sliding::holy(Worm &w) {
    return IO::PlayerInput::moveNone;
```

jun 26, 18 17:16 **Sliding.cpp** Page 2/2

```
67  }
68
69  IO::PlayerInput Worm::Sliding::setTimeoutTo(Worm &w, int t) {
70      return IO::PlayerInput::moveNone;
71  }
72
73  IO::PlayerInput Worm::Sliding::aerialAttack(Worm &w) {
74      return IO::PlayerInput::moveNone;
75  }
76
77  IO::PlayerInput Worm::Sliding::dynamite(Worm &w) {
78      return IO::PlayerInput::moveNone;
79  }
80
81  IO::PlayerInput Worm::Sliding::teleport(Worm &w) {
82      return IO::PlayerInput::moveNone;
83  }
84
85  IO::PlayerInput Worm::Sliding::positionSelected(Worm &w) {
86      return IO::PlayerInput::moveNone;
87  }
88
89  IO::PlayerInput Worm::Sliding::baseballBat(Worm &w) {
90      return IO::PlayerInput::moveNone;
91  }
```

jun 26, 18 17:16 **Land.h** Page 1/1

```
1   //
2   // Created by rodrigo on 3/06/18.
3   //
4
5   #ifndef INC_4_WORMS_LAND_H
6   #define INC_4_WORMS_LAND_H
7
8   #include "GameStateMsg.h"
9   #include "WormState.h"
10
11  namespace Worm {
12  class Land : public State {
13     public:
14      Land();
15      ~Land();
16
17      virtual void update(float dt) override;
18
19      virtual IO::PlayerInput moveRight(Worm &w) override;
20      virtual IO::PlayerInput moveLeft(Worm &w) override;
21      virtual IO::PlayerInput stopMove(Worm &w) override;
22      virtual IO::PlayerInput jump(Worm &w) override;
23      virtual IO::PlayerInput backFlip(Worm &w) override;
24      virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;
25      virtual IO::PlayerInput bazooka(Worm &w) override;
26      virtual IO::PlayerInput grenade(Worm &w) override;
27      virtual IO::PlayerInput cluster(Worm &w) override;
28      virtual IO::PlayerInput mortar(Worm &w) override;
29      virtual IO::PlayerInput banana(Worm &w) override;
30      virtual IO::PlayerInput holy(Worm &w) override;
31      virtual IO::PlayerInput aerialAttack(Worm &w) override;
32      virtual IO::PlayerInput dynamite(Worm &w) override;
33      virtual IO::PlayerInput baseballBat(Worm &w) override;
34
35      virtual IO::PlayerInput teleport(Worm &w) override;
36      virtual IO::PlayerInput positionSelected(Worm &w) override;
37      virtual IO::PlayerInput endShot(Worm &w) override;
38      virtual IO::PlayerInput startShot(Worm &w) override;
39      virtual IO::PlayerInput pointUp(Worm &w) override;
40      virtual IO::PlayerInput pointDown(Worm &w) override;
41  };
42  }  // namespace Worm
43
44  #endif  // INC_4_WORMS_LAND_H
```

```cpp
//
// Created by rodrigo on 3/06/18.
//

#include "Land.h"

Worm::Land::Land() : State(StateID::Land) {}

Worm::Land::~Land() {}

void Worm::Land::update(float dt) {}

IO::PlayerInput Worm::Land::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::banana(Worm &w) {
    return IO::PlayerInput::moveNone;
```



```cpp
}

IO::PlayerInput Worm::Land::holy(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::setTimeoutTo(Worm &w, int t) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::aerialAttack(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::positionSelected(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::dynamite(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::teleport(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Land::baseballBat(Worm &w) {
    return IO::PlayerInput::moveNone;
}
```

jun 26, 18 17:16 **Hit.h** Page 1/1

```
/*
 *  Created by Rodrigo.
 *  date: 28/05/18
 */

#ifndef __Hit_H__
#define __Hit_H__

#include "WormState.h"

namespace Worm {
class Hit : public State {
   public:
    explicit Hit();
    virtual ~Hit();

    virtual void update(float dt) override;

    virtual IO::PlayerInput moveRight(Worm &w) override;
    virtual IO::PlayerInput moveLeft(Worm &w) override;
    virtual IO::PlayerInput stopMove(Worm &w) override;
    virtual IO::PlayerInput jump(Worm &w) override;
    virtual IO::PlayerInput backFlip(Worm &w) override;
    virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;

    virtual IO::PlayerInput bazooka(Worm &w) override;
    virtual IO::PlayerInput grenade(Worm &w) override;
    virtual IO::PlayerInput cluster(Worm &w) override;
    virtual IO::PlayerInput mortar(Worm &w) override;
    virtual IO::PlayerInput banana(Worm &w) override;
    virtual IO::PlayerInput holy(Worm &w) override;
    virtual IO::PlayerInput aerialAttack(Worm &w) override;
    virtual IO::PlayerInput dynamite(Worm &w) override;
    virtual IO::PlayerInput baseballBat(Worm &w) override;

    virtual IO::PlayerInput teleport(Worm &w) override;
    virtual IO::PlayerInput positionSelected(Worm &w) override;
    virtual IO::PlayerInput startShot(Worm &w) override;
    virtual IO::PlayerInput endShot(Worm &w) override;
    virtual IO::PlayerInput pointUp(Worm &w) override;
    virtual IO::PlayerInput pointDown(Worm &w) override;
};
}  // namespace Worm

#endif  //__Hit_H__
```

jun 26, 18 17:16 **Hit.cpp** Page 1/2

```
/*
 *  Created by Rodrigo.
 *  date: 28/05/18
 */

#include "Hit.h"

Worm::Hit::Hit() : State(StateID::Hit) {}

Worm::Hit::~Hit() {}

void Worm::Hit::update(float dt) {}

IO::PlayerInput Worm::Hit::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Hit::banana(Worm &w) {
```

jun 26, 18 17:16 **Hit.cpp** Page 2/2

```
67      return IO::PlayerInput::moveNone;
68  }
69
70  IO::PlayerInput Worm::Hit::holy(Worm &w) {
71      return IO::PlayerInput::moveNone;
72  }
73
74  IO::PlayerInput Worm::Hit::setTimeoutTo(Worm &w, int t) {
75      return IO::PlayerInput::moveNone;
76  }
77
78  IO::PlayerInput Worm::Hit::aerialAttack(Worm &w) {
79      return IO::PlayerInput::moveNone;
80  }
81
82  IO::PlayerInput Worm::Hit::positionSelected(Worm &w) {
83      return IO::PlayerInput::moveNone;
84  }
85
86  IO::PlayerInput Worm::Hit::dynamite(Worm &w) {
87      return IO::PlayerInput::moveNone;
88  }
89
90  IO::PlayerInput Worm::Hit::teleport(Worm &w) {
91      return IO::PlayerInput::moveNone;
92  }
93
94  IO::PlayerInput Worm::Hit::baseballBat(Worm &w) {
95      return IO::PlayerInput::moveNone;
96  }
```

jun 26, 18 17:16 **Falling.h** Page 1/1

```
1   //
2   // Created by rodrigo on 3/06/18.
3   //
4
5   #ifndef INC_4_WORMS_FALLING_H
6   #define INC_4_WORMS_FALLING_H
7
8   #include "GameStateMsg.h"
9   #include "WormState.h"
10
11  namespace Worm {
12  class Falling : public State {
13     public:
14      Falling();
15      ~Falling();
16
17      virtual void update(float dt) override;
18
19      virtual IO::PlayerInput moveRight(Worm &w) override;
20      virtual IO::PlayerInput moveLeft(Worm &w) override;
21      virtual IO::PlayerInput stopMove(Worm &w) override;
22      virtual IO::PlayerInput jump(Worm &w) override;
23      virtual IO::PlayerInput backFlip(Worm &w) override;
24      virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;
25
26      virtual IO::PlayerInput bazooka(Worm &w) override;
27      virtual IO::PlayerInput grenade(Worm &w) override;
28      virtual IO::PlayerInput cluster(Worm &w) override;
29      virtual IO::PlayerInput mortar(Worm &w) override;
30      virtual IO::PlayerInput banana(Worm &w) override;
31      virtual IO::PlayerInput holy(Worm &w) override;
32      virtual IO::PlayerInput aerialAttack(Worm &w) override;
33      virtual IO::PlayerInput dynamite(Worm &w) override;
34      virtual IO::PlayerInput baseballBat(Worm &w) override;
35
36      virtual IO::PlayerInput teleport(Worm &w) override;
37      virtual IO::PlayerInput positionSelected(Worm &w) override;
38      virtual IO::PlayerInput endShot(Worm &w) override;
39      virtual IO::PlayerInput startShot(Worm &w) override;
40      virtual IO::PlayerInput pointUp(Worm &w) override;
41      virtual IO::PlayerInput pointDown(Worm &w) override;
42  };
43  }  // namespace Worm
44
45  #endif  // INC_4_WORMS_FALLING_H
```

jun 26, 18 17:16 **Falling.cpp** Page 1/2

```cpp
//
// Created by rodrigo on 3/06/18.
//

#include "Falling.h"

Worm::Falling::Falling() : State(StateID::Falling) {}

Worm::Falling::~Falling() {}

void Worm::Falling::update(float dt) {}

IO::PlayerInput Worm::Falling::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::banana(Worm &w) {
    return IO::PlayerInput::moveNone;
```

jun 26, 18 17:16 **Falling.cpp** Page 2/2

```cpp
}

IO::PlayerInput Worm::Falling::holy(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::setTimeoutTo(Worm &w, int t) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::positionSelected(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::aerialAttack(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::dynamite(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::teleport(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Falling::baseballBat(Worm &w) {
    return IO::PlayerInput::moveNone;
}
```

jun 26, 18 17:16 **Drowning.h** Page 1/1

```
/*
 *  Created by Rodrigo.
 *  date: 29/05/18
 */

#ifndef __Drown_H__
#define __Drown_H__

#include "WormState.h"

namespace Worm {
class Drowning : public State {
   public:
    Drowning();
    ~Drowning();

    virtual void update(float dt) override;

    virtual IO::PlayerInput moveRight(Worm &w) override;
    virtual IO::PlayerInput moveLeft(Worm &w) override;
    virtual IO::PlayerInput stopMove(Worm &w) override;
    virtual IO::PlayerInput jump(Worm &w) override;
    virtual IO::PlayerInput backFlip(Worm &w) override;
    virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;

    virtual IO::PlayerInput bazooka(Worm &w) override;
    virtual IO::PlayerInput grenade(Worm &w) override;
    virtual IO::PlayerInput cluster(Worm &w) override;
    virtual IO::PlayerInput mortar(Worm &w) override;
    virtual IO::PlayerInput banana(Worm &w) override;
    virtual IO::PlayerInput holy(Worm &w) override;
    virtual IO::PlayerInput aerialAttack(Worm &w) override;
    virtual IO::PlayerInput dynamite(Worm &w) override;
    virtual IO::PlayerInput baseballBat(Worm &w) override;
    virtual IO::PlayerInput teleport(Worm &w) override;
    virtual IO::PlayerInput positionSelected(Worm &w) override;

    virtual IO::PlayerInput startShot(Worm &w) override;
    virtual IO::PlayerInput endShot(Worm &w) override;
    virtual IO::PlayerInput pointUp(Worm &w) override;
    virtual IO::PlayerInput pointDown(Worm &w) override;
};
}  // namespace Worm

#endif  //__Drown_H__
```

jun 26, 18 17:16 **Drowning.cpp** Page 1/2

```
/*
 *  Created by Rodrigo.
 *  date: 29/05/18
 */

#include "Drowning.h"

Worm::Drowning::Drowning() : State(StateID::Drowning) {}

Worm::Drowning::~Drowning() {}

void Worm::Drowning::update(float dt) {}

IO::PlayerInput Worm::Drowning::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Drowning::banana(Worm &w) {
```

jun 26, 18 17:16 **Drowning.cpp** Page 2/2

```
67      return IO::PlayerInput::moveNone;
68  }
69
70  IO::PlayerInput Worm::Drowning::holy(Worm &w) {
71      return IO::PlayerInput::moveNone;
72  }
73
74  IO::PlayerInput Worm::Drowning::setTimeoutTo(Worm &w, int t) {
75      return IO::PlayerInput::moveNone;
76  }
77
78  IO::PlayerInput Worm::Drowning::aerialAttack(Worm &w) {
79      return IO::PlayerInput::moveNone;
80  }
81
82  IO::PlayerInput Worm::Drowning::positionSelected(Worm &w) {
83      return IO::PlayerInput::moveNone;
84  }
85
86  IO::PlayerInput Worm::Drowning::dynamite(Worm &w) {
87      return IO::PlayerInput::moveNone;
88  }
89
90  IO::PlayerInput Worm::Drowning::teleport(Worm &w) {
91      return IO::PlayerInput::moveNone;
92  }
93
94  IO::PlayerInput Worm::Drowning::baseballBat(Worm &w) {
95      return IO::PlayerInput::moveNone;
96  }
```

jun 26, 18 17:16 **Die.h** Page 1/1

```
1   /*
2    *  Created by Rodrigo.
3    *  date: 28/05/18
4    */
5
6   #ifndef __Die_H__
7   #define __Die_H__
8
9   #include "WormState.h"
10
11  namespace Worm {
12  class Die : public State {
13     public:
14      Die();
15      ~Die();
16
17      virtual void update(float dt) override;
18
19      virtual IO::PlayerInput moveRight(Worm &w) override;
20      virtual IO::PlayerInput moveLeft(Worm &w) override;
21      virtual IO::PlayerInput stopMove(Worm &w) override;
22      virtual IO::PlayerInput jump(Worm &w) override;
23      virtual IO::PlayerInput backFlip(Worm &w) override;
24      virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;
25
26      virtual IO::PlayerInput bazooka(Worm &w) override;
27      virtual IO::PlayerInput grenade(Worm &w) override;
28      virtual IO::PlayerInput cluster(Worm &w) override;
29      virtual IO::PlayerInput mortar(Worm &w) override;
30      virtual IO::PlayerInput banana(Worm &w) override;
31      virtual IO::PlayerInput holy(Worm &w) override;
32      virtual IO::PlayerInput aerialAttack(Worm &w) override;
33      virtual IO::PlayerInput teleport(Worm &w) override;
34      virtual IO::PlayerInput positionSelected(Worm &w) override;
35      virtual IO::PlayerInput dynamite(Worm &w) override;
36      virtual IO::PlayerInput baseballBat(Worm &w) override;
37
38      virtual IO::PlayerInput startShot(Worm &w) override;
39      virtual IO::PlayerInput endShot(Worm &w) override;
40      virtual IO::PlayerInput pointUp(Worm &w) override;
41      virtual IO::PlayerInput pointDown(Worm &w) override;
42  };
43  }  // namespace Worm
44
45  #endif  //__Die_H__
```

jun 26, 18 17:16 **Die.cpp** Page 1/2

```
/*
 *  Created by Rodrigo.
 *  date: 28/05/18
 */

#include "Die.h"

Worm::Die::Die() : State(StateID::Die) {}

Worm::Die::~Die() {}

void Worm::Die::update(float dt) {}

IO::PlayerInput Worm::Die::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::banana(Worm &w) {
```

jun 26, 18 17:16 **Die.cpp** Page 2/2

```
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::holy(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::setTimeoutTo(Worm &w, int t) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::aerialAttack(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::positionSelected(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::dynamite(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::teleport(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Die::baseballBat(Worm &w) {
    return IO::PlayerInput::moveNone;
}
```

jun 26, 18 17:16 **Dead.h** Page 1/1

```
/*
 *  Created by Rodrigo.
 *  date: 28/05/18
 */

#ifndef __Dead_H__
#define __Dead_H__

#include "WormState.h"

namespace Worm {
class Dead : public State {
   public:
    Dead();
    ~Dead();

    virtual void update(float dt) override;

    virtual IO::PlayerInput moveRight(Worm &w) override;
    virtual IO::PlayerInput moveLeft(Worm &w) override;
    virtual IO::PlayerInput stopMove(Worm &w) override;
    virtual IO::PlayerInput jump(Worm &w) override;
    virtual IO::PlayerInput backFlip(Worm &w) override;
    virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;

    virtual IO::PlayerInput bazooka(Worm &w) override;
    virtual IO::PlayerInput grenade(Worm &w) override;
    virtual IO::PlayerInput cluster(Worm &w) override;
    virtual IO::PlayerInput mortar(Worm &w) override;
    virtual IO::PlayerInput banana(Worm &w) override;
    virtual IO::PlayerInput holy(Worm &w) override;
    virtual IO::PlayerInput aerialAttack(Worm &w) override;
    virtual IO::PlayerInput teleport(Worm &w) override;
    virtual IO::PlayerInput positionSelected(Worm &w) override;
    virtual IO::PlayerInput dynamite(Worm &w) override;
    virtual IO::PlayerInput baseballBat(Worm &w) override;

    virtual IO::PlayerInput startShot(Worm &w) override;
    virtual IO::PlayerInput endShot(Worm &w) override;
    virtual IO::PlayerInput pointUp(Worm &w) override;
    virtual IO::PlayerInput pointDown(Worm &w) override;
};
}  // namespace Worm

#endif  //__Dead_H__
```

jun 26, 18 17:16 **Dead.cpp** Page 1/2

```
/*
 *  Created by Rodrigo.
 *  date: 28/05/18
 */

#include <iostream>

#include "Dead.h"

Worm::Dead::Dead() : State(StateID::Dead) {}

Worm::Dead::~Dead() {}

void Worm::Dead::update(float dt) {
}

IO::PlayerInput Worm::Dead::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Dead::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
```

jun 26, 18 17:16 **Dead.cpp** Page 2/2

```
67  }
68
69  IO::PlayerInput Worm::Dead::banana(Worm &w) {
70      return IO::PlayerInput::moveNone;
71  }
72
73  IO::PlayerInput Worm::Dead::holy(Worm &w) {
74      return IO::PlayerInput::moveNone;
75  }
76
77  IO::PlayerInput Worm::Dead::setTimeoutTo(Worm &w, int t) {
78      return IO::PlayerInput::moveNone;
79  }
80
81  IO::PlayerInput Worm::Dead::aerialAttack(Worm &w) {
82      return IO::PlayerInput::moveNone;
83  }
84
85  IO::PlayerInput Worm::Dead::positionSelected(Worm &w) {
86      return IO::PlayerInput::moveNone;
87  }
88
89  IO::PlayerInput Worm::Dead::dynamite(Worm &w) {
90      return IO::PlayerInput::moveNone;
91  }
92
93  IO::PlayerInput Worm::Dead::teleport(Worm &w) {
94      return IO::PlayerInput::moveNone;
95  }
96
97  IO::PlayerInput Worm::Dead::baseballBat(Worm &w) {
98      return IO::PlayerInput::moveNone;
99  }
```

jun 26, 18 17:16 **Batting.h** Page 1/1

```
1   //
2   // Created by rodrigo on 23/06/18.
3   //
4
5   #ifndef INC_4_WORMS_BATTING_H
6   #define INC_4_WORMS_BATTING_H
7
8   #include "../Worm.h"
9   #include "GameStateMsg.h"
10  #include "WormState.h"
11
12  namespace Worm {
13  class Batting : public State {
14     public:
15      Batting();
16
17      ~Batting();
18
19      virtual void update(float dt) override;
20
21      virtual IO::PlayerInput moveRight(Worm &w) override;
22
23      virtual IO::PlayerInput moveLeft(Worm &w) override;
24
25      virtual IO::PlayerInput stopMove(Worm &w) override;
26
27      virtual IO::PlayerInput jump(Worm &w) override;
28
29      virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;
30
31      virtual IO::PlayerInput bazooka(Worm &w) override;
32
33      virtual IO::PlayerInput grenade(Worm &w) override;
34
35      virtual IO::PlayerInput cluster(Worm &w) override;
36
37      virtual IO::PlayerInput mortar(Worm &w) override;
38
39      virtual IO::PlayerInput banana(Worm &w) override;
40
41      virtual IO::PlayerInput holy(Worm &w) override;
42
43      virtual IO::PlayerInput aerialAttack(Worm &w) override;
44
45      virtual IO::PlayerInput dynamite(Worm &w) override;
46
47      virtual IO::PlayerInput baseballBat(Worm &w) override;
48
49      virtual IO::PlayerInput teleport(Worm &w) override;
50
51      virtual IO::PlayerInput positionSelected(Worm &w) override;
52
53      virtual IO::PlayerInput startShot(Worm &w) override;
54
55      virtual IO::PlayerInput endShot(Worm &w) override;
56
57      virtual IO::PlayerInput pointUp(Worm &w) override;
58
59      virtual IO::PlayerInput pointDown(Worm &w) override;
60
61      virtual IO::PlayerInput backFlip(Worm &w) override;
62  };
63  }
64
65  #endif  // INC_4_WORMS_BATTING_H
```

```cpp
//
// Created by rodrigo on 23/06/18.
//

#include "Batting.h"

Worm::Batting::Batting() : State(StateID::Batting) {}

Worm::Batting::~Batting() {}

void Worm::Batting::update(float dt) {}

IO::PlayerInput Worm::Batting::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::banana(Worm &w) {
    return IO::PlayerInput::moveNone;
}
```



```cpp
}

IO::PlayerInput Worm::Batting::holy(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::setTimeoutTo(Worm &w, int t) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::aerialAttack(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::positionSelected(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::dynamite(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::teleport(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::Batting::baseballBat(Worm &w) {
    return IO::PlayerInput::moveNone;
}
```

jun 26, 18 17:16 **BackFlip.h** Page 1/1

```
/*
 *  Created by Rodrigo.
 *  date: 20/05/18
 */

#ifndef __WORM_BACK_FLIP_H__
#define __WORM_BACK_FLIP_H__

#include "GameStateMsg.h"
#include "WormState.h"

namespace Worm {
class BackFlip : public State {
   public:
    explicit BackFlip();
    virtual ~BackFlip();

    virtual void update(float dt) override;

    virtual IO::PlayerInput moveRight(Worm &w) override;
    virtual IO::PlayerInput moveLeft(Worm &w) override;
    virtual IO::PlayerInput stopMove(Worm &w) override;
    virtual IO::PlayerInput jump(Worm &w) override;
    virtual IO::PlayerInput backFlip(Worm &w) override;
    virtual IO::PlayerInput setTimeoutTo(Worm &w, int t) override;

    virtual IO::PlayerInput bazooka(Worm &w) override;
    virtual IO::PlayerInput grenade(Worm &w) override;
    virtual IO::PlayerInput cluster(Worm &w) override;
    virtual IO::PlayerInput mortar(Worm &w) override;
    virtual IO::PlayerInput banana(Worm &w) override;
    virtual IO::PlayerInput holy(Worm &w) override;
    virtual IO::PlayerInput aerialAttack(Worm &w) override;
    virtual IO::PlayerInput dynamite(Worm &w) override;
    virtual IO::PlayerInput baseballBat(Worm &w) override;

    virtual IO::PlayerInput teleport(Worm &w) override;
    virtual IO::PlayerInput positionSelected(Worm &w) override;
    virtual IO::PlayerInput startShot(Worm &w) override;
    virtual IO::PlayerInput endShot(Worm &w) override;
    virtual IO::PlayerInput pointUp(Worm &w) override;
    virtual IO::PlayerInput pointDown(Worm &w) override;
};
}  // namespace Worm

#endif  //__WORM_BACK_FLIP_H__
```

jun 26, 18 17:16 **BackFlip.cpp** Page 1/2

```
/*
 *  Created by Rodrigo.
 *  date: 20/05/18
 */

#include "BackFlip.h"

Worm::BackFlip::BackFlip() : State(StateID::StartBackFlip) {}

Worm::BackFlip::~BackFlip() {}

void Worm::BackFlip::update(float dt) {}

IO::PlayerInput Worm::BackFlip::moveRight(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::moveLeft(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::stopMove(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::jump(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::backFlip(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::bazooka(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::pointUp(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::pointDown(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::startShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::endShot(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::grenade(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::cluster(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::mortar(Worm &w) {
    return IO::PlayerInput::moveNone;
}

IO::PlayerInput Worm::BackFlip::banana(Worm &w) {
```

jun 26, 18 17:16 **BackFlip.cpp** Page 2/2

```
67      return IO::PlayerInput::moveNone;
68  }
69
70  IO::PlayerInput Worm::BackFlip::holy(Worm &w) {
71      return IO::PlayerInput::moveNone;
72  }
73
74  IO::PlayerInput Worm::BackFlip::setTimeoutTo(Worm &w, int t) {
75      return IO::PlayerInput::moveNone;
76  }
77
78  IO::PlayerInput Worm::BackFlip::aerialAttack(Worm &w) {
79      return IO::PlayerInput::moveNone;
80  }
81
82  IO::PlayerInput Worm::BackFlip::positionSelected(Worm &w) {
83      return IO::PlayerInput::moveNone;
84  }
85
86  IO::PlayerInput Worm::BackFlip::dynamite(Worm &w) {
87      return IO::PlayerInput::moveNone;
88  }
89
90  IO::PlayerInput Worm::BackFlip::teleport(Worm &w) {
91      return IO::PlayerInput::moveNone;
92  }
93
94  IO::PlayerInput Worm::BackFlip::baseballBat(Worm &w) {
95      return IO::PlayerInput::moveNone;
96  }
```

jun 26, 18 17:16 **Worm.h** Page 1/2

```
1   /*
2    *  Created by Federico Manuel Gomez Peter.
3    *  date: 18/05/18
4    */
5
6   #ifndef __Worm_H__
7   #define __Worm_H__
8
9   #define FLY_CENTER_FRAME 16
10  #define DROWN_CENTER_FRAME 0
11  #define ANGLE_STEP 5.625f
12
13  #include <SDL2/SDL.h>
14  #include <memory>
15
16  #include "Animation.h"
17  #include "Camera.h"
18  #include "Direction.h"
19  #include "GameSoundEffects.h"
20  #include "GameStateMsg.h"
21  #include "GameTextures.h"
22  #include "SoundEffectPlayer.h"
23  #include "Stream.h"
24  #include "Weapons/Explosion.h"
25  #include "Weapons/Weapon.h"
26  #include "WormState/WormState.h"
27  #include "utils.h"
28
29  namespace Worm {
30  using ID = char;
31
32  class Worm {
33      /**
34       * Fundamental class of the game, it is in charge of handling the user's
35       * entries, and delegate in their attributes the rendering and animation
36       */
37     public:
38      Direction direction{Direction::left};
39      std::uint8_t health{0};
40      const ID id;
41
42      explicit Worm(ID id, const GUI::GameTextureManager &texture_mgr,
43                    const GUI::GameSoundEffectManager &sound_effect_mgr);
44      ~Worm() {}
45      /**
46       * @brief Calls State::update to change frame of animation
47       * @param dt
48       */
49      void update(float dt);
50      /**
51       * Render worm in position (x,y)
52       * @param x
53       * @param y
54       */
55      void render(GUI::Position &p, GUI::Camera &cam);
56      /**
57       * @brief Using a state pattern, change its state depending on the input, an
    d
58       * sends it to the server
59       * @param key
60       * @param out
61       */
62      void handleKeyDown(SDL_Keycode key, IO::Stream<IO::PlayerMsg> *out);
63      /**
64       * @brief Same as handleKeyDown, but stops its current status.
65       * @param key
```

```
66       * @param out
67       */
68      void handleKeyUp(SDL_Keycode key, IO::Stream<IO::PlayerMsg> *out);
69      /**
70       * @brief Receives a position in global coordinates and sends it to the stat
    e
71       * so it can handle it.
72       * @param position
73       */
74      void mouseButtonDown(GUI::Position position, IO::Stream<IO::PlayerMsg> *pStr
    eam);
75      GUI::Animation getAnimation(StateID state) const;
76      /**
77       * @brief Attributte that implements state pattern to change the behavior
78       * of the class polymorphically.
79       */
80      void setState(StateID state);
81      StateID &getState() const;
82      /**
83       * @brief Update the animation with weapons, depending on the
84       * worm's angle.
85       * @param angle
86       */
87      void setWeaponAngle(float angle);
88      /**
89       * @brief Update the used weapon
90       * @param id
91       */
92      void setWeapon(const WeaponID &id);
93      const WeaponID &getWeaponID() const;
94      void setPosition(GUI::Position p);
95      /**
96       * @brief Starts the PowerBar's rendering, adding animations in its containe
    r
97       */
98      void startShot();
99      /**
100      * @brief End PowerBar's rendering, freeing its container
101      */
102     void endShot();
103     /**
104      * @brief resets some attributes when the turn ends
105      */
106     void reset();
107
108    private:
109     const GUI::GameTextureManager &texture_mgr;
110     const GUI::GameSoundEffectManager &sound_effect_mgr;
111     std::shared_ptr<State> state{nullptr};
112     GUI::Animation animation;
113     std::shared_ptr<Weapon> weapon{nullptr};
114     bool active{false};
115     GUI::Position position{0, 0};
116     std::shared_ptr<Explosion> explosion{nullptr};
117     bool hasFired{false};
118     std::shared_ptr<GUI::SoundEffectPlayer> soundEffectPlayer{nullptr};
119     void playSoundEffect(StateID state);
120     void playWeaponSoundEffect(const WeaponID &id);
121 };
122 }  // namespace Worm
123
124 #endif  //__Worm__H__
```



```
1   /*
2    *  Created by Federico Manuel Gomez Peter.
3    *  date: 18/05/18
4    */
5
6   #include <SDL2/SDL_system.h>
7   #include <cmath>
8   #include <iostream>
9
10  #include "GameStateMsg.h"
11  #include "Text.h"
12  #include "Weapons/AerialAttack.h"
13  #include "Weapons/Banana.h"
14  #include "Weapons/BaseballBat.h"
15  #include "Weapons/Bazooka.h"
16  #include "Weapons/Cluster.h"
17  #include "Weapons/Dynamite.h"
18  #include "Weapons/Grenade.h"
19  #include "Weapons/Holy.h"
20  #include "Weapons/Mortar.h"
21  #include "Weapons/Teleport.h"
22  #include "Weapons/WeaponNone.h"
23  #include "Worm.h"
24  #include "WormState/BackFlip.h"
25  #include "WormState/Batting.h"
26  #include "WormState/Dead.h"
27  #include "WormState/Die.h"
28  #include "WormState/Drowning.h"
29  #include "WormState/Falling.h"
30  #include "WormState/Hit.h"
31  #include "WormState/Land.h"
32  #include "WormState/Sliding.h"
33  #include "WormState/Teleported.h"
34  #include "WormState/Teleporting.h"
35  #include "WormState/WormBackFlipping.h"
36  #include "WormState/WormEndBackFlip.h"
37  #include "WormState/WormEndJump.h"
38  #include "WormState/WormJumping.h"
39  #include "WormState/WormStartJump.h"
40  #include "WormState/WormStill.h"
41  #include "WormState/WormWalk.h"
42
43  Worm::Worm::Worm(ID id, const GUI::GameTextureManager &texture_mgr,
44                   const GUI::GameSoundEffectManager &sound_effect_mgr)
45      : id(id),
46        texture_mgr(texture_mgr),
47        sound_effect_mgr(sound_effect_mgr),
48        animation(texture_mgr.get(GUI::GameTextures::WormIdle)) {
49      this→setState(::Worm::StateID::Still);
50      this→weapon = std::shared_ptr<Weapon>(new Bazooka(texture_mgr));
51  }
52
53  void Worm::Worm::handleKeyDown(SDL_Keycode key, IO::Stream<IO::PlayerMsg> *out)
    {
54      IO::PlayerInput i = IO::PlayerInput::moveNone;
55      switch (key) {
56          case SDLK_RIGHT:
57              i = this→state→moveRight(*this);
58              break;
59          case SDLK_LEFT:
60              i = this→state→moveLeft(*this);
61              break;
62          case SDLK_UP:
63              i = this→state→pointUp(*this);
64              break;
65          case SDLK_DOWN:
```

```
66              i = this→state→pointDown(*this);
67              break;
68          case SDLK_RETURN:
69              i = this→state→jump(*this);
70              break;
71          case SDLK_BACKSPACE:
72              i = this→state→backFlip(*this);
73              break;
74          case SDLK_1:
75              i = this→state→setTimeoutTo(*this, 1);
76              break;
77          case SDLK_2:
78              i = this→state→setTimeoutTo(*this, 2);
79              break;
80          case SDLK_3:
81              i = this→state→setTimeoutTo(*this, 3);
82              break;
83          case SDLK_4:
84              i = this→state→setTimeoutTo(*this, 4);
85              break;
86          case SDLK_5:
87              i = this→state→setTimeoutTo(*this, 5);
88              break;
89          case SDLK_F1:
90              i = this→state→bazooka(*this);
91              break;
92          case SDLK_F2:
93              i = this→state→grenade(*this);
94              break;
95          case SDLK_F3:
96              i = this→state→cluster(*this);
97              break;
98          case SDLK_F4:
99              i = this→state→mortar(*this);
100             break;
101         case SDLK_F5:
102             i = this→state→banana(*this);
103             break;
104         case SDLK_F6:
105             i = this→state→holy(*this);
106             break;
107         case SDLK_F7:
108             i = this→state→aerialAttack(*this);
109             break;
110         case SDLK_F8:
111             i = this→state→dynamite(*this);
112             break;
113         case SDLK_F9:
114             i = this→state→baseballBat(*this);
115             break;
116         case SDLK_F10:
117             i = this→state→teleport(*this);
118             break;
119         case SDLK_SPACE:
120             i = this→state→startShot(*this);
121             break;
122     }
123     if (i ≠ IO::PlayerInput::moveNone) {
124         IO::PlayerMsg msg;
125         msg.input = i;
126         *out << msg;
127     }
128 }
129
130 void Worm::Worm::handleKeyUp(SDL_Keycode key, IO::Stream<IO::PlayerMsg> *out) {
131     IO::PlayerInput i = IO::PlayerInput::moveNone;
```



```
132     switch (key) {
133         case SDLK_RIGHT:
134             i = this→state→stopMove(*this);
135             break;
136         case SDLK_LEFT:
137             i = this→state→stopMove(*this);
138             break;
139         case SDLK_SPACE:
140             i = this→state→endShot(*this);
141             break;
142     }
143     if (i ≠ IO::PlayerInput::moveNone) {
144         IO::PlayerMsg msg;
145         msg.input = i;
146         *out << msg;
147     }
148 }
149
150 void Worm::Worm::render(GUI::Position &p, GUI::Camera &cam) {
151     SDL_RendererFlip flipType =
152         this→direction ≡ Direction::left ? SDL_FLIP_NONE : SDL_FLIP_HORIZONTAL;
153     if (this→state→getState() ≠ StateID::Still ∨
154         this→weapon→getWeaponID() ≡ WeaponID::WNone) {
155         this→animation.render(p, cam, flipType);
156     } else {
157         this→weapon→render(p, cam, flipType);
158     }
159     if (this→explosion ≠ nullptr) {
160         this→explosion→render(cam);
161         if (this→explosion→finished()) {
162             this→explosion = nullptr;
163         }
164     }
165 }
166
167 void Worm::Worm::update(float dt) {
168     this→state→update(dt);
169     this→animation.update(dt);
170     this→weapon→update(dt);
171     if (this→explosion ≠ nullptr) {
172         this→explosion→update(dt);
173     }
174     if (this→soundEffectPlayer ≠ nullptr) {
175         this→soundEffectPlayer→update(dt);
176     }
177 }
178
179 GUI::Animation Worm::Worm::getAnimation(StateID state) const {
180     switch (state) {
181         case StateID::Still:
182             break;
183         case StateID::Walk:
184             return GUI::Animation{this→texture_mgr.get(GUI::GameTextures::WormW
    alk)};
185         case StateID::StartBackFlip:
186         case StateID::StartJump:
187             return GUI::Animation{this→texture_mgr.get(GUI::GameTextures::Start
    Jump), true};
188         case StateID::Jumping:
189             return GUI::Animation{this→texture_mgr.get(GUI::GameTextures::Jumpi
    ng)};
190         case StateID::Land:
191         case StateID::EndBackFlip:
192         case StateID::EndJump:
193             return GUI::Animation{this→texture_mgr.get(GUI::GameTextures::EndJu
    mp), true};
```

```
194         case StateID::BackFlipping: {
195             GUI::Animation animation{this→texture_mgr.get(GUI::GameTextures::Ba
    ckFlipping)};
196             animation.setAnimateOnce();
197             return animation;
198         }
199         case StateID::Falling: {
200             GUI::Animation animation{this→texture_mgr.get(GUI::GameTextures::Fa
    lling), true};
201             animation.setAnimateOnce();
202             return animation;
203         }
204         case StateID::Batting: {
205             GUI::Animation animation{this→texture_mgr.get(GUI::GameTextures::Wo
    rmBaseballBatting),
206                                      false, 25, false};
207             //            animation.setAnimateOnce();
208             return animation;
209         }
210         case StateID::Teleporting: {
211             GUI::Animation animation{this→texture_mgr.get(GUI::GameTextures::Wo
    rmTeleporting),
212                                      true};
213             animation.setAnimateOnce();
214             return animation;
215         }
216         case StateID::Teleported: {
217             GUI::Animation animation{this→texture_mgr.get(GUI::GameTextures::Wo
    rmTeleporting),
218                                      true};
219             animation.setPlayInverse();
220             return animation;
221         }
222         case StateID::Hit:
223             return GUI::Animation{this→texture_mgr.get(GUI::GameTextures::Fly),
     true,
224                                   FLY_CENTER_FRAME, false};
225         case StateID::Die: {
226             GUI::Animation animation{this→texture_mgr.get(GUI::GameTextures::Di
    e)};
227             animation.setAnimateOnce();
228             return animation;
229         }
230         case StateID::Drowning:
231             return GUI::Animation{this→texture_mgr.get(GUI::GameTextures::Fly),
     true,
232                                   DROWN_CENTER_FRAME, false};
233         case StateID::Dead:
234             return GUI::Animation{this→texture_mgr.get(GUI::GameTextures::Dead)
    , true};
235         case StateID::Sliding:
236             return GUI::Animation{this→texture_mgr.get(GUI::GameTextures::Slidi
    ng), true};
237     }
238     return GUI::Animation{this→texture_mgr.get(GUI::GameTextures::WormIdle), tr
    ue};
239 }
240
241 void Worm::Worm::playSoundEffect(StateID state) {
242     this→soundEffectPlayer = nullptr;
243     switch (state) {
244         case StateID::Still:
245             break;
246         case StateID::Walk:
247             this→soundEffectPlayer =
248                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
```



```
    er{
249                     this→sound_effect_mgr.get(GUI::GameSoundEffects::WalkCompre
    ss), 0.7f});
250             this→soundEffectPlayer→update(0.3f);
251             break;
252         case StateID::StartBackFlip:
253             this→soundEffectPlayer =
254                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
255                     this→sound_effect_mgr.get(GUI::GameSoundEffects::WormBackFl
    ip), true});
256             this→soundEffectPlayer→play();
257             break;
258         case StateID::StartJump:
259             this→soundEffectPlayer =
260                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
261                     this→sound_effect_mgr.get(GUI::GameSoundEffects::WormJump),
     true});
262             this→soundEffectPlayer→play();
263             break;
264         case StateID::Jumping:
265             break;
266         case StateID::EndBackFlip:
267         case StateID::EndJump:
268         case StateID::Land:
269             this→soundEffectPlayer =
270                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
271                     this→sound_effect_mgr.get(GUI::GameSoundEffects::WormLandin
    g), true});
272             this→soundEffectPlayer→play();
273             break;
274         case StateID::BackFlipping:
275             break;
276         case StateID::Falling:
277             break;
278         case StateID::Batting:
279             break;
280         case StateID::Teleporting:
281             break;
282         case StateID::Teleported:
283             break;
284         case StateID::Hit:
285             this→soundEffectPlayer =
286                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
287                     this→sound_effect_mgr.get(GUI::GameSoundEffects::WormHit),
    true});
288             this→soundEffectPlayer→play();
289             break;
290         case StateID::Die:
291             this→soundEffectPlayer =
292                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
293                     this→sound_effect_mgr.get(GUI::GameSoundEffects::WormDie),
    true});
294             this→soundEffectPlayer→play();
295             break;
296         case StateID::Drowning:
297             this→soundEffectPlayer =
298                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
299                     this→sound_effect_mgr.get(GUI::GameSoundEffects::WormDrowni
    ng)});
300             break;
```

```
301         case StateID::Dead:
302             this→soundEffectPlayer =
303                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
304                     this→sound_effect_mgr.get(GUI::GameSoundEffects::Explosion)
    , true});
305             this→soundEffectPlayer→play();
306             break;
307         case StateID::Sliding:
308             break;
309     }
310 }
311
312 void Worm::Worm::setState(StateID state) {
313     if (this→state ≡ nullptr ∨ this→state→getState() ≠ state) {
314         this→animation = this→getAnimation(state);
315         this→playSoundEffect(state);
316
317         /* creates the right state type */
318         switch (state) {
319             case StateID::Still:
320                 this→state = std::shared_ptr<State>(new Still());
321                 break;
322             case StateID::Walk:
323                 this→state = std::shared_ptr<State>(new Walk());
324                 break;
325             case StateID::StartJump:
326                 this→state = std::shared_ptr<State>(new StartJump());
327                 break;
328             case StateID::Jumping:
329                 this→state = std::shared_ptr<State>(new Jumping());
330                 break;
331             case StateID::EndJump:
332                 this→state = std::shared_ptr<State>(new EndJump());
333                 break;
334             case StateID::StartBackFlip:
335                 this→state = std::shared_ptr<State>(new BackFlip());
336                 break;
337             case StateID::BackFlipping:
338                 this→state = std::shared_ptr<State>(new BackFlipping());
339                 break;
340             case StateID::EndBackFlip:
341                 this→state = std::shared_ptr<State>(new EndBackFlip());
342                 break;
343             case StateID::Falling:
344                 this→state = std::shared_ptr<State>(new Falling());
345                 break;
346             case StateID::Land:
347                 this→state = std::shared_ptr<State>(new Land());
348                 break;
349             case StateID::Batting:
350                 this→state = std::shared_ptr<State>(new Batting());
351                 break;
352             case StateID::Teleporting:
353                 this→state = std::shared_ptr<State>(new Teleporting());
354                 break;
355             case StateID::Teleported:
356                 this→state = std::shared_ptr<State>(new Teleported());
357                 break;
358             case StateID::Hit:
359                 this→state = std::shared_ptr<State>(new Hit());
360                 break;
361             case StateID::Die:
362                 this→state = std::shared_ptr<State>(new Die());
363                 break;
364             case StateID::Drowning:
```



```
365                 this→state = std::shared_ptr<State>(new Drowning());
366                 break;
367             case StateID::Dead:
368                 this→state = std::shared_ptr<State>(new Dead());
369                 this→explosion = std::shared_ptr<Explosion>(new Explosion(this
    →texture_mgr));
370                 this→explosion→position = this→position;
371                 break;
372             case StateID::Sliding:
373                 this→state = std::shared_ptr<State>(new Sliding());
374                 break;
375         }
376     }
377 }
378
379 Worm::StateID &Worm::Worm::getState() const {
380     return this→state→getState();
381 }
382
383 void Worm::Worm::setWeapon(const WeaponID &id) {
384     //    this->weapon.setWeapon(id);
385     if (this→weapon→getWeaponID() ≠ id) {
386         switch (id) {
387             case WeaponID::WBazooka:
388                 this→weapon = std::shared_ptr<Weapon>(new Bazooka(this→texture
    _mgr));
389                 break;
390             case WeaponID::WGrenade:
391                 this→weapon = std::shared_ptr<Weapon>(new Grenade(this→texture
    _mgr));
392                 break;
393             case WeaponID::WCluster:
394                 this→weapon = std::shared_ptr<Weapon>(new Cluster(this→texture
    _mgr));
395                 break;
396             case WeaponID::WMortar:
397                 this→weapon = std::shared_ptr<Weapon>(new Mortar(this→texture_
    mgr));
398                 break;
399             case WeaponID::WBanana:
400                 this→weapon = std::shared_ptr<Weapon>(new Banana(this→texture_
    mgr));
401                 break;
402             case WeaponID::WHoly:
403                 this→weapon = std::shared_ptr<Weapon>(new Holy(this→texture_mg
    r));
404                 break;
405             case WeaponID::WAerial:
406                 this→weapon = std::shared_ptr<Weapon>(new AerialAttack(this→te
    xture_mgr));
407                 break;
408             case WeaponID::WDynamite:
409                 this→weapon = std::shared_ptr<Weapon>(new Dynamite(this→textur
    e_mgr));
410                 break;
411             case WeaponID::WBaseballBat:
412                 this→weapon = std::shared_ptr<Weapon>(new BaseballBat(this→tex
    ture_mgr));
413                 break;
414             case WeaponID::WTeleport:
415                 this→weapon = std::shared_ptr<Weapon>(new Teleport(this→textur
    e_mgr));
416                 break;
417             case WeaponID::WNone:
418                 this→weapon = std::shared_ptr<Weapon>(new WeaponNone(this→text
    ure_mgr));
```

```
419                 break;
420             case WeaponID::WExplode:
421                 break;
422             case WeaponID::WFragment:
423                 break;
424         }
425     }
426 }
427
428 const Worm::WeaponID &Worm::Worm::getWeaponID() const {
429     return this→weapon→getWeaponID();
430 }
431
432 void Worm::Worm::setWeaponAngle(float angle) {
433     this→weapon→setAngle(angle, this→direction);
434 }
435
436 void Worm::Worm::setPosition(GUI::Position p) {
437     this→position = p;
438 }
439
440 void Worm::Worm::startShot() {
441     if (¬this→hasFired) {
442         this→weapon→startShot();
443     }
444 }
445
446 void Worm::Worm::endShot() {
447     if (this→weapon→getWeaponID() ≠ WeaponID::WAerial ∧
448         this→weapon→getWeaponID() ≠ WeaponID::WTeleport ∧
449         this→weapon→getWeaponID() ≠ WeaponID::WNone) {
450         if (¬this→hasFired) {
451             this→weapon→endShot();
452             this→playWeaponSoundEffect(this→getWeaponID());
453             this→hasFired = true;
454         }
455     }
456 }
457
458 void Worm::Worm::mouseButtonDown(GUI::Position position, IO::Stream<IO::PlayerMs
    g> *out) {
459     IO::PlayerInput i = this→state→positionSelected(*this);
460     if (i ≠ IO::PlayerInput::moveNone ∧ ¬this→hasFired ∧ this→weapon→position
    Selected()) {
461         this→playWeaponSoundEffect(this→weapon→getWeaponID());
462         IO::PlayerMsg msg;
463         msg.input = i;
464         msg.position = position;
465         *out << msg;
466     }
467 }
468
469 void Worm::Worm::playWeaponSoundEffect(const WeaponID &id) {
470     this→soundEffectPlayer = nullptr;
471
472     switch (id) {
473         case WeaponID::WBazooka:
474             this→soundEffectPlayer =
475                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
476                     this→sound_effect_mgr.get(GUI::GameSoundEffects::Shot), tru
    e});
477             this→soundEffectPlayer→play();
478             break;
479         case WeaponID::WGrenade:
480             this→soundEffectPlayer =
```



```
481                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
482                     this→sound_effect_mgr.get(GUI::GameSoundEffects::Shot), tru
    e});
483             this→soundEffectPlayer→play();
484             break;
485         case WeaponID::WCluster:
486             this→soundEffectPlayer =
487                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
488                     this→sound_effect_mgr.get(GUI::GameSoundEffects::Shot), tru
    e});
489             this→soundEffectPlayer→play();
490             break;
491         case WeaponID::WMortar:
492             this→soundEffectPlayer =
493                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
494                     this→sound_effect_mgr.get(GUI::GameSoundEffects::Shot), tru
    e});
495             this→soundEffectPlayer→play();
496             break;
497         case WeaponID::WBanana:
498             this→soundEffectPlayer =
499                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
500                     this→sound_effect_mgr.get(GUI::GameSoundEffects::Shot), tru
    e});
501             this→soundEffectPlayer→play();
502             break;
503         case WeaponID::WHoly:
504             this→soundEffectPlayer =
505                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
506                     this→sound_effect_mgr.get(GUI::GameSoundEffects::Holy), tru
    e});
507             this→soundEffectPlayer→play();
508             break;
509         case WeaponID::WAerial:
510             this→soundEffectPlayer =
511                 std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlay
    er{
512                     this→sound_effect_mgr.get(GUI::GameSoundEffects::AirStrike)
    , true});
513             this→soundEffectPlayer→play();
514             break;
515         case WeaponID::WDynamite:
516             break;
517         case WeaponID::WBaseballBat:
518             break;
519         case WeaponID::WTeleport:
520             break;
521         case WeaponID::WNone:
522             break;
523         case WeaponID::WExplode:
524             break;
525         case WeaponID::WFragment:
526             break;
527     }
528 }
529
530 void Worm::Worm::reset() {
531     this→hasFired = false;
532 }
```

jun 26, 18 17:16 **Wind.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 20/06/18
 */

#ifndef __WIND_H__
#define __WIND_H__

#include <Camera.h>
#include "GameTextures.h"

namespace GUI {
/**
 *  @brief receives the snapshot's intensity and draws the help interface
 *  to show the wind's intensity.
 */
class Wind {
   public:
    Wind(const GameTextureManager &textureManager, Camera &cam);
    ~Wind() = default;
    void render(std::int8_t intensity, int windowWidth);

   private:
    const GameTextureManager &tex;
    Camera &cam;
};
}

#endif  //__WIND_H__
```

jun 26, 18 17:16 **Wind.cpp** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 20/06/18
 */

#include "Wind.h"
#include <WrapTexture.h>
#include "Texture.h"

GUI::Wind::Wind(const GUI::GameTextureManager &tex, GUI::Camera &cam) : tex(tex)
, cam(cam) {}

void GUI::Wind::render(std::int8_t intensity, int windowWidth) {
    const GUI::Texture &toUse = (intensity > 0) ? this→tex.get(GameTextures::Wi
ndRight)
                                                : this→tex.get(GameTextures::Wi
ndLeft);
    float scaledIntensity = (float)std::abs(intensity) / 127 * this→cam.getScal
e();
    GUI::WrapTexture wt{toUse, scaledIntensity, (float)toUse.getHeight() / this
→cam.getScale()};
    GUI::ScreenPosition p{windowWidth, toUse.getHeight()};
    wt.renderFixed(p, this→cam);
}
```

jun 26, 18 17:16 **WeaponNone.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#ifndef __WEAPON_NONE_H__
#define __WEAPON_NONE_H__

#include <vector>

#include "Weapon.h"

namespace Worm {
class WeaponNone : public Weapon {
   public:
    explicit WeaponNone(const GUI::GameTextureManager &textureManager);
    ~WeaponNone() = default;
    void update(float dt) override;
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) override;
    void setAngle(float angle, Direction d) override;
    void startShot() override;
    void endShot() override;
    bool positionSelected() override;
};
}  // namespace Worm

#endif  //__WEAPON_NONE_H__
```

jun 26, 18 17:16 **WeaponNone.cpp** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#include "WeaponNone.h"

Worm::WeaponNone::WeaponNone(const GUI::GameTextureManager &textureManager)
    : Weapon(textureManager, GUI::GameTextures::WormIdle, 0, WeaponID::WNone) {}

void Worm::WeaponNone::update(float dt) {}

void Worm::WeaponNone::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) {}

void Worm::WeaponNone::setAngle(float angle, Worm::Direction d) {}

void Worm::WeaponNone::startShot() {}

void Worm::WeaponNone::endShot() {}

bool Worm::WeaponNone::positionSelected() {
    return false;
}
```

```
1  /*
2   *  Created by Federico Manuel Gomez Peter.
3   *  date: 27/05/18
4   */
5
6  #ifndef __Weapon_H__
7  #define __Weapon_H__
8
9  #include "../GameTextures.h"
10 #include "Animation.h"
11 #include "Camera.h"
12 #include "Direction.h"
13 #include "GameStateMsg.h"
14 #include "TextureManager.h"
15
16 #define ANGLE_STEP 5.625f
17 #define SCOPE_DISTANCE 4
18
19 namespace Worm {
20 class Weapon {
21    public:
22     explicit Weapon(const GUI::GameTextureManager &texMgr, GUI::GameTextures tex
   ,
23                     uint16_t centerFrame, WeaponID id);
24     virtual ~Weapon() = default;
25     /**
26      * updates all its animations.
27      * @param dt
28      */
29     virtual void update(float dt) = 0;
30     /**
31      * renders all its animations.
32      * @param p
33      * @param cam
34      * @param flip
35      */
36     virtual void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &fl
   ip) = 0;
37     const WeaponID &getWeaponID() const;
38     /**
39      * updates animations' frame depending on the angle.
40      * @param angle
41      */
42     virtual void setAngle(float angle, Direction d) = 0;
43     /**
44      * Starts the PowerBar's rendering, adding animations in its container
45      */
46     virtual void startShot() = 0;
47     /**
48      * End PowerBar's rendering, freeing its container
49      */
50     virtual void endShot() = 0;
51     /**
52      * When using remoteControl weapons, starts the animation of the worm
53      * and return
54      */
55     virtual bool positionSelected() = 0;
56
57    protected:
58     const GUI::GameTextureManager &textureMgr;
59     WeaponID current;
60     uint16_t centerFrame;
61     GUI::Animation weaponAnimation;
62     float angle{0.0f};
63 };
64 }  // namespace Weapon
```



```
65
66 #endif  //__Weapon_H__
```

jun 26, 18 17:16 **Weapon.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 27/05/18
 */

#include <iostream>

#include "GameStateMsg.h"
#include "Weapon.h"

Worm::Weapon::Weapon(const GUI::GameTextureManager &texMgr, GUI::GameTextures tex,
                     uint16_t centerFrame, WeaponID id)
    : textureMgr(texMgr),
      current(id),
      centerFrame(centerFrame),
      weaponAnimation(texMgr.get(tex), false, centerFrame, false) {}

const Worm::WeaponID &Worm::Weapon::getWeaponID() const {
    return this→current;
}
```

jun 26, 18 17:16 **Teleport.h** Page 1/1

```cpp
//
// Created by rodrigo on 16/06/18.
//

#ifndef INC_4_WORMS_TELEPORT_H
#define INC_4_WORMS_TELEPORT_H

#define TELEPORT_CENTER_FRAME 0

#include "Weapon.h"

namespace Worm {
class Teleport : public Weapon {
   public:
    explicit Teleport(const GUI::GameTextureManager &textureManager);
    ~Teleport() = default;
    void update(float dt) override;
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) override;
    void setAngle(float angle, Direction d) override;
    void startShot() override;
    void endShot() override;
    bool positionSelected() override;

   private:
    void endAnimation();
};
}  // namespace Worm

#endif  // INC_4_WORMS_TELEPORT_H
```

jun 26, 18 17:16 **Teleport.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 16/06/18.
//

#include "Teleport.h"

Worm::Teleport::Teleport(const GUI::GameTextureManager &tex)
    : Weapon(tex, GUI::GameTextures::WormTeleport, TELEPORT_CENTER_FRAME, WeaponID::WTeleport) {
    this→weaponAnimation.setAnimateOnce();
}

void Worm::Teleport::update(float dt) {
    if (¬this→weaponAnimation.finished()) {
        this→weaponAnimation.update(dt);
    } else {
        this→endAnimation();
    }
}

void Worm::Teleport::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) {
    this→weaponAnimation.render(p, cam, flip);
}

void Worm::Teleport::setAngle(float angle, Worm::Direction d) {}

void Worm::Teleport::startShot() {}

void Worm::Teleport::endShot() {}

bool Worm::Teleport::positionSelected() {
    this→weaponAnimation.setAutoUpdate(true);
    return true;
}

void Worm::Teleport::endAnimation() {
    this→weaponAnimation.setFrame(TELEPORT_CENTER_FRAME);
    this→weaponAnimation.setAutoUpdate(false);
}
```

jun 26, 18 17:16 **Scope.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#ifndef __Scope_H__
#define __Scope_H__

#include <Animation.h>
#include <Camera.h>
#include "../GameTextures.h"
#include "Direction.h"

namespace Weapon {
class Scope {
   public:
    Scope(const GUI::GameTextureManager &tex);
    ~Scope() = default;
    void setAngle(float angle, Worm::Direction d);
    void update(float dt);
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip);

   private:
    float angle{0.0f};
    GUI::Animation animation;
};
}

#endif  //__Scope_H__
```

jun 26, 18 17:16 **Scope.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#include "Scope.h"
#include "Direction.h"
#include "Weapon.h"

Weapon::Scope::Scope(const GUI::GameTextureManager &tex)
    : animation(tex.get(GUI::GameTextures::Scope), false, 0, false) {}

void Weapon::Scope::setAngle(float angle, Worm::Direction d) {
    this→angle = d ≡ Worm::Direction::right ? angle : 180 − angle;
}

void Weapon::Scope::update(float dt) {
    this→animation.update(dt);
}

void Weapon::Scope::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip
&flip) {
    GUI::Position scopePos = GUI::Position(SCOPE_DISTANCE * cos(this→angle * PI
 / 180),
                                           SCOPE_DISTANCE * sin(this→angle * PI
 / 180)) +
                             p;
    this→animation.render(scopePos, cam, flip);
}
```

jun 26, 18 17:16 **PowerBar.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#ifndef __PowerBar_H__
#define __PowerBar_H__

#include <Animation.h>
#include <Camera.h>
#include <vector>

#include "../GameTextures.h"
#include "Direction.h"

#define POWER_FRAMES_QUANTITY 16

namespace Weapon {
class PowerBar {
   public:
    explicit PowerBar(const GUI::GameTextureManager &tex);
    ~PowerBar() = default;
    void setAngle(float angle, Worm::Direction d);
    void update(float dt);
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip);
    void startShot();
    void endShot();

   private:
    bool shotStarted{false};
    float angle{0.0f};
    float elapsedTime{0.0f};
    uint16_t power{0};
    std::vector<GUI::Animation> animations;
    const GUI::GameTextureManager &textureManager;
};
}

#endif  //__PowerBar_H__
```

jun 26, 18 17:16 **PowerBar.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#include "PowerBar.h"
#include "Weapon.h"

Weapon::PowerBar::PowerBar(const GUI::GameTextureManager &tex) : textureManager(tex) {
    this→animations.reserve(POWER_FRAMES_QUANTITY);
}

void Weapon::PowerBar::setAngle(float angle, Worm::Direction d) {
    this→angle = d ≡ Worm::Direction::right ? angle : 180 − angle;
}

void Weapon::PowerBar::update(float dt) {
    if (this→shotStarted) {
        this→elapsedTime += dt;
        if (this→power < POWER_FRAMES_QUANTITY ∧ this→elapsedTime < POWER_CHARGE_TIME) {
            this→animations.emplace_back(this→textureManager.get(GUI::GameTextures::PowerBar),
                                         false, this→power, false);
            this→power++;
        }
    }
}

void Weapon::PowerBar::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) {
    for (int i = 0; i < this→power; i++) {
        GUI::Position powerPos =
            GUI::Position((SCOPE_DISTANCE * (log10(10 * i / 17))) * cos(this→angle * PI / 180),
                          (SCOPE_DISTANCE * (log10(10 * i / 17))) * sin(this→angle * PI / 180)) +
            p;
        this→animations[i].render(powerPos, cam, flip);
    }
}

void Weapon::PowerBar::startShot() {
    this→shotStarted = true;
}

void Weapon::PowerBar::endShot() {
    this→shotStarted = false;
    this→animations.erase(this→animations.begin(), this→animations.end());
    this→power = 0;
    this→elapsedTime = 0.0f;
}
```

jun 26, 18 17:16 **Mortar.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#ifndef __MORTAR_H__
#define __MORTAR_H__

#include <vector>

#include "PowerBar.h"
#include "Scope.h"
#include "Weapon.h"

#define MORTAR_CENTER_FRAME 16

namespace Worm {
class Mortar : public Weapon {
   public:
    explicit Mortar(const GUI::GameTextureManager &textureManager);
    ~Mortar() = default;
    void update(float dt) override;
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) override;
    void setAngle(float angle, Direction d) override;
    void startShot() override;
    void endShot() override;
    bool positionSelected() override;

   private:
    ::Weapon::Scope scope;
    ::Weapon::PowerBar powerBar;
};
}  // namespace Worm

#endif  //__MORTAR_H__
```

jun 26, 18 17:16 **Mortar.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#include <cmath>

#include "Mortar.h"

Worm::Mortar::Mortar(const GUI::GameTextureManager &tex)
    : Weapon(tex, GUI::GameTextures::Bazooka2, MORTAR_CENTER_FRAME, WeaponID::WMortar),
      scope(this→textureMgr),
      powerBar(this→textureMgr) {}

void Worm::Mortar::update(float dt) {
    this→weaponAnimation.update(dt);
    this→scope.update(dt);
    this→powerBar.update(dt);
}

void Worm::Mortar::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) {
    this→weaponAnimation.render(p, cam, flip);
    this→scope.render(p, cam, flip);
    this→powerBar.render(p, cam, flip);
}

void Worm::Mortar::setAngle(float angle, Worm::Direction d) {
    this→weaponAnimation.setFrame((int)std::ceil(angle / ANGLE_STEP) + this→centerFrame);
    this→scope.setAngle(angle, d);
    this→powerBar.setAngle(angle, d);
}

void Worm::Mortar::startShot() {
    this→powerBar.startShot();
}

void Worm::Mortar::endShot() {
    this→powerBar.endShot();
}

bool Worm::Mortar::positionSelected() {
    return false;
}
```

jun 26, 18 17:16 **Holy.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#ifndef __HOLY_H__
#define __HOLY_H__

#include <vector>

#include "PowerBar.h"
#include "Scope.h"
#include "Weapon.h"

#define HOLY_CENTER_FRAME 15

namespace Worm {
class Holy : public Weapon {
   public:
    explicit Holy(const GUI::GameTextureManager &textureManager);
    ~Holy() = default;
    void update(float dt) override;
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) override;
    void setAngle(float angle, Direction d) override;
    void startShot() override;
    void endShot() override;
    bool positionSelected() override;

   private:
    ::Weapon::Scope scope;
    ::Weapon::PowerBar powerBar;
};
}  // namespace Worm

#endif  //__HOLY_H__
```

jun 26, 18 17:16 **Holy.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#include <cmath>

#include "Holy.h"

Worm::Holy::Holy(const GUI::GameTextureManager &tex)
    : Weapon(tex, GUI::GameTextures::WormHoly, HOLY_CENTER_FRAME, WeaponID::WHoly),
      scope(this→textureMgr),
      powerBar(this→textureMgr) {}

void Worm::Holy::update(float dt) {
    this→weaponAnimation.update(dt);
    this→scope.update(dt);
    this→powerBar.update(dt);
}

void Worm::Holy::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) {
    this→weaponAnimation.render(p, cam, flip);
    this→scope.render(p, cam, flip);
    this→powerBar.render(p, cam, flip);
}

void Worm::Holy::setAngle(float angle, Worm::Direction d) {
    this→weaponAnimation.setFrame((int)std::ceil(angle / ANGLE_STEP) + this→centerFrame);
    this→scope.setAngle(angle, d);
    this→powerBar.setAngle(angle, d);
}

void Worm::Holy::startShot() {
    this→powerBar.startShot();
}

void Worm::Holy::endShot() {
    this→powerBar.endShot();
}

bool Worm::Holy::positionSelected() {
    return false;
}
```

jun 26, 18 17:16 **Grenade.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#ifndef __GRENADE_H__
#define __GRENADE_H__

#include <Camera.h>

#include "../GameTextures.h"
#include "Direction.h"
#include "PowerBar.h"
#include "Scope.h"
#include "Weapon.h"

#define GRENADE_CENTER_FRAME 15

namespace Worm {
class Grenade : public Weapon {
   public:
    explicit Grenade(const GUI::GameTextureManager &textureManager);
    ~Grenade() = default;
    void update(float dt) override;
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) override;
    void setAngle(float angle, Direction d) override;
    void startShot() override;
    void endShot() override;
    bool positionSelected() override;

   private:
    ::Weapon::Scope scope;
    ::Weapon::PowerBar powerBar;
};
}  // namespace Worm

#endif  //__GRENADE_H__
```

jun 26, 18 17:16 **Grenade.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#include <cmath>

#include "Grenade.h"

Worm::Grenade::Grenade(const GUI::GameTextureManager &tex)
    : Weapon(tex, GUI::GameTextures::WormGrenade, GRENADE_CENTER_FRAME, WeaponID::WGrenade),
      scope(this→textureMgr),
      powerBar(this→textureMgr) {}

void Worm::Grenade::update(float dt) {
    this→weaponAnimation.update(dt);
    this→scope.update(dt);
    this→powerBar.update(dt);
}

void Worm::Grenade::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) {
    this→weaponAnimation.render(p, cam, flip);
    this→scope.render(p, cam, flip);
    this→powerBar.render(p, cam, flip);
}

void Worm::Grenade::setAngle(float angle, Worm::Direction d) {
    this→weaponAnimation.setFrame((int)std::ceil(angle / ANGLE_STEP) + this→centerFrame);
    this→scope.setAngle(angle, d);
    this→powerBar.setAngle(angle, d);
}

void Worm::Grenade::startShot() {
    this→powerBar.startShot();
}

void Worm::Grenade::endShot() {
    this→powerBar.endShot();
}

bool Worm::Grenade::positionSelected() {
    return false;
}
```

jun 26, 18 17:16 **Explosion.h** Page 1/1

```cpp
//
// Created by rodrigo on 2/06/18.
//

#ifndef INC_4_WORMS_EXPLOSION_H
#define INC_4_WORMS_EXPLOSION_H

#include <Animation.h>
#include <vector>
#include "../GameTextures.h"

namespace Worm {
class Explosion {
   public:
    explicit Explosion(const GUI::GameTextureManager &texture_mgr);
    ~Explosion() = default;
    void update(float dt);
    void render(GUI::Camera &cam);
    GUI::Position position{0, 0};
    bool finished();

   private:
    const GUI::GameTextureManager &texture_mgr;
    std::vector<GUI::Animation> animations;
    bool explosionFinished{false};
};
}

#endif  // INC_4_WORMS_EXPLOSION_H
```

jun 26, 18 17:16 **Explosion.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 2/06/18.
//

#include "Explosion.h"

Worm::Explosion::Explosion(const GUI::GameTextureManager &texture_mgr) : texture_mgr(texture_mgr) {
    this→animations.emplace_back(this→texture_mgr.get(GUI::GameTextures::Explosion));
    this→animations.back().setAnimateOnce();
}
// TODO make observer in client side to clean exploded bullet
void Worm::Explosion::update(float dt) {
    for (auto &animation : this→animations) {
        animation.update(dt);
        this→explosionFinished = animation.finished();
    }
}

void Worm::Explosion::render(GUI::Camera &cam) {
    for (auto &animation : this→animations) {
        animation.render(this→position, cam, SDL_FLIP_HORIZONTAL);
    }
}

bool Worm::Explosion::finished() {
    return this→explosionFinished;
}
```

jun 26, 18 17:16 **Dynamite.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 16/06/18
 */

#ifndef __DYNAMITE_H__
#define __DYNAMITE_H__

#include "Weapon.h"

#define DYNAMITE_CENTER_FRAME 0

namespace Worm {
class Dynamite : public Weapon {
   public:
    explicit Dynamite(const GUI::GameTextureManager &textureManager);
    ~Dynamite() = default;
    void update(float dt) override;
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) override;
    void setAngle(float angle, Direction d) override;
    void startShot() override;
    void endShot() override;
    bool positionSelected() override;
};
}  // namespace Worm

#endif  //__DYNAMITE_H__
```

jun 26, 18 17:16 **Dynamite.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 16/06/18
 */

#include "Dynamite.h"

Worm::Dynamite::Dynamite(const GUI::GameTextureManager &tex)
    : Weapon(tex, GUI::GameTextures::WormDynamite, DYNAMITE_CENTER_FRAME, WeaponID::WDynamite) {}

void Worm::Dynamite::update(float dt) {
    this→weaponAnimation.update(dt);
}

void Worm::Dynamite::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) {
    this→weaponAnimation.render(p, cam, flip);
}

void Worm::Dynamite::setAngle(float angle, Worm::Direction d) {}

void Worm::Dynamite::startShot() {}

void Worm::Dynamite::endShot() {}

bool Worm::Dynamite::positionSelected() {
    return false;
}
```

jun 26, 18 17:16 **Cluster.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#ifndef __CLUSTER_H__
#define __CLUSTER_H__

#include <vector>

#include "PowerBar.h"
#include "Scope.h"
#include "Weapon.h"

#define CLUSTER_CENTER_FRAME 15

namespace Worm {
class Cluster : public Weapon {
   public:
    explicit Cluster(const GUI::GameTextureManager &textureManager);
    ~Cluster() = default;
    void update(float dt) override;
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) override;
    void setAngle(float angle, Direction d) override;
    void startShot() override;
    void endShot() override;
    bool positionSelected() override;

   private:
    ::Weapon::Scope scope;
    ::Weapon::PowerBar powerBar;
};
}  // namespace Worm

#endif  //__CLUSTER_H__
```

jun 26, 18 17:16 **Cluster.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#include <cmath>

#include "Cluster.h"

Worm::Cluster::Cluster(const GUI::GameTextureManager &tex)
    : Weapon(tex, GUI::GameTextures::WormCluster, CLUSTER_CENTER_FRAME, WeaponID::WCluster),
      scope(this→textureMgr),
      powerBar(this→textureMgr) {}

void Worm::Cluster::update(float dt) {
    this→weaponAnimation.update(dt);
    this→scope.update(dt);
    this→powerBar.update(dt);
}

void Worm::Cluster::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) {
    this→weaponAnimation.render(p, cam, flip);
    this→scope.render(p, cam, flip);
    this→powerBar.render(p, cam, flip);
}

void Worm::Cluster::setAngle(float angle, Worm::Direction d) {
    this→weaponAnimation.setFrame((int)std::ceil(angle / ANGLE_STEP) + this→centerFrame);
    this→scope.setAngle(angle, d);
    this→powerBar.setAngle(angle, d);
}

void Worm::Cluster::startShot() {
    this→powerBar.startShot();
}

void Worm::Cluster::endShot() {
    this→powerBar.endShot();
}

bool Worm::Cluster::positionSelected() {
    return false;
}
```

jun 26, 18 17:16 **Bullet.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 26/05/18
 */

#ifndef __Bullet_h__
#define __Bullet_h__

#include <GameStateMsg.h>
#include <memory>

#include "../GameSoundEffects.h"
#include "../GameTextures.h"
#include "../SoundEffectPlayer.h"
#include "Animation.h"
#include "Explosion.h"

#define MISSILE_0_DEG_FRAME 8
#define MISSILE_ANGLE_STEP 11.25f

namespace Ammo {
class Bullet {
   public:
    explicit Bullet(const GUI::GameTextureManager &texture_mgr,
                    const GUI::GameSoundEffectManager &sound_effect_mgr, Worm::WeaponID id);
    ~Bullet() = default;
    void update(float dt);
    void render(GUI::Position p, GUI::Camera &cam);
    void setAngle(float angle);
    void setPosition(GUI::Position p);
    GUI::Position getPosition();
    void madeImpact();
    bool exploding();
    bool exploded();

   private:
    float angle{0};
    bool updateManually{true};
    const GUI::GameTextureManager &texture_mgr;
    const GUI::GameSoundEffectManager &sound_effect_mgr;
    GUI::Animation animation;
    GUI::Position position{0, 0};
    Worm::Explosion explosion;
    bool explode{false};
    Worm::WeaponID wid;
    std::shared_ptr<GUI::SoundEffectPlayer> soundEffectPlayer{nullptr};
};
}

#endif  //__Bullet_H__
```

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 26/05/18
 */

#include <cmath>
#include <iostream>

#include "../GameSoundEffects.h"
#include "Bullet.h"

Ammo::Bullet::Bullet(const GUI::GameTextureManager &texture_mgr,
                     const GUI::GameSoundEffectManager &sound_effect_mgr, Worm::WeaponID id)
    : texture_mgr(texture_mgr),
      sound_effect_mgr(sound_effect_mgr),
      animation(this→texture_mgr.get(GUI::GameTextures::Missile), true, MISSILE_0_DEG_FRAME,
                false),
      explosion(this→texture_mgr) {
    switch (id) {
        case Worm::WeaponID::WBazooka:
            this→animation = GUI::Animation(this→texture_mgr.get(GUI::GameTextures::Missile),
                                            true, MISSILE_0_DEG_FRAME, false);
            break;
        case Worm::WeaponID::WGrenade:
            this→animation = GUI::Animation(this→texture_mgr.get(GUI::GameTextures::Grenade),
                                            false, MISSILE_0_DEG_FRAME, false);
            break;
        case Worm::WeaponID::WCluster:
            this→animation = GUI::Animation(this→texture_mgr.get(GUI::GameTextures::Cluster),
                                            false, MISSILE_0_DEG_FRAME, false);
            break;
        case Worm::WeaponID::WMortar:
            this→animation = GUI::Animation(this→texture_mgr.get(GUI::GameTextures::Mortar),
                                            false, MISSILE_0_DEG_FRAME, false);
            break;
        case Worm::WeaponID::WBanana:
            this→animation = GUI::Animation(this→texture_mgr.get(GUI::GameTextures::Banana),
                                            false, MISSILE_0_DEG_FRAME, false);
            break;
        case Worm::WeaponID::WHoly:
            this→animation = GUI::Animation(this→texture_mgr.get(GUI::GameTextures::Holy), false,
                                            MISSILE_0_DEG_FRAME, false);
            break;
        case Worm::WeaponID::WAerial:
            this→animation = GUI::Animation(this→texture_mgr.get(GUI::GameTextures::AirMissile),
                                            false, MISSILE_0_DEG_FRAME, false);
            break;
        case Worm::WeaponID::WBaseballBat:
            break;
        case Worm::WeaponID::WTeleport:
            break;
        case Worm::WeaponID::WExplode:
            break;
        case Worm::WeaponID::WFragment:
            this→animation =
                GUI::Animation(this→texture_mgr.get(GUI::GameTextures::Fragment), false, 0, true);
            this→updateManually = false;
            break;
        case Worm::WeaponID::WDynamite:
            this→animation =
                GUI::Animation(this→texture_mgr.get(GUI::GameTextures::Dynamite), false, 0, true);
            this→updateManually = false;
            break;
        case Worm::WeaponID::WNone:
            break;
    }
    this→wid = id;
}

void Ammo::Bullet::update(float dt) {
    if (¬this→explode) {
        if (this→updateManually) {
            float angle = (this→angle − 90);
            if (angle ≥ 360) {
                angle −= 360;
            }
            float angleStep = MISSILE_ANGLE_STEP;
            this→animation.setFrame((int)std::floor(angle / angleStep));
        } else {
            this→animation.update(dt);
        }
    } else {
        this→explosion.update(dt);
    }
}

void Ammo::Bullet::render(GUI::Position p, GUI::Camera &cam) {
    if (¬this→explode) {
        this→animation.render(p, cam, SDL_FLIP_HORIZONTAL);
    } else {
        this→explosion.render(cam);
    }
}

void Ammo::Bullet::setAngle(float angle) {
    this→angle = angle;
}

bool Ammo::Bullet::exploded() {
    return this→explosion.finished();
}

void Ammo::Bullet::madeImpact() {
    this→explode = true;
    this→soundEffectPlayer = std::shared_ptr<GUI::SoundEffectPlayer>(new GUI::SoundEffectPlayer{
        this→sound_effect_mgr.get(GUI::GameSoundEffects::Explosion), true});
    this→soundEffectPlayer→play();
}

void Ammo::Bullet::setPosition(GUI::Position p) {
    this→position = p;
    this→explosion.position = p;
}

bool Ammo::Bullet::exploding() {
    return this→explode;
}

GUI::Position Ammo::Bullet::getPosition() {
    return this→position;
```

jun 26, 18 17:16 **Bullet.cpp** Page 3/3

```
121 }
```

jun 26, 18 17:16 **Bazooka.h** Page 1/1

```
1  /*
2   *  Created by Federico Manuel Gomez Peter.
3   *  date: 04/06/18
4   */
5
6  #ifndef __BAZOOKA_H__
7  #define __BAZOOKA_H__
8
9  #include <vector>
10
11 #include "PowerBar.h"
12 #include "Scope.h"
13 #include "Weapon.h"
14
15 #define BAZOOKA_CENTER_FRAME 16
16
17 namespace Worm {
18 class Bazooka : public Weapon {
19    public:
20     explicit Bazooka(const GUI::GameTextureManager &textureManager);
21     ~Bazooka() = default;
22     void update(float dt) override;
23     void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) over
   ride;
24     void setAngle(float angle, Direction d) override;
25     void startShot() override;
26     void endShot() override;
27     bool positionSelected() override;
28
29    private:
30     ::Weapon::Scope scope;
31     ::Weapon::PowerBar powerBar;
32 };
33 }  // namespace Worm
34
35 #endif  //__BAZOOKA_H__
```

jun 26, 18 17:16 **Bazooka.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#include <cmath>

#include "Bazooka.h"

Worm::Bazooka::Bazooka(const GUI::GameTextureManager &tex)
    : Weapon(tex, GUI::GameTextures::Bazooka, BAZOOKA_CENTER_FRAME, WeaponID::WBazooka),
      scope(this→textureMgr),
      powerBar(this→textureMgr) {}

void Worm::Bazooka::update(float dt) {
    this→weaponAnimation.update(dt);
    this→scope.update(dt);
    this→powerBar.update(dt);
}

void Worm::Bazooka::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) {
    this→weaponAnimation.render(p, cam, flip);
    this→scope.render(p, cam, flip);
    this→powerBar.render(p, cam, flip);
}

void Worm::Bazooka::setAngle(float angle, Direction d) {
    this→weaponAnimation.setFrame((int)std::ceil(angle / ANGLE_STEP) + this→centerFrame);
    this→scope.setAngle(angle, d);
    this→powerBar.setAngle(angle, d);
}

void Worm::Bazooka::startShot() {
    this→powerBar.startShot();
}

void Worm::Bazooka::endShot() {
    this→powerBar.endShot();
}

bool Worm::Bazooka::positionSelected() {
    return false;
}
```

jun 26, 18 17:16 **BaseballBat.h** Page 1/1

```cpp
//
// Created by rodrigo on 16/06/18.
//

#ifndef INC_4_WORMS_BASEBALLBAT_H
#define INC_4_WORMS_BASEBALLBAT_H

#include "Scope.h"
#include "Weapon.h"

#define BASEBALL_BAT_CENTER_FRAME 16

namespace Worm {
class BaseballBat : public Weapon {
   public:
    explicit BaseballBat(const GUI::GameTextureManager &textureManager);
    ~BaseballBat() = default;
    void update(float dt) override;
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) override;
    void setAngle(float angle, Direction d) override;
    void startShot() override;
    void endShot() override;
    bool positionSelected() override;

   private:
    ::Weapon::Scope scope;
};
}  // namespace Worm

#endif  // INC_4_WORMS_BASEBALLBAT_H
```

jun 26, 18 17:16 **BaseballBat.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 16/06/18.
//

#include "BaseballBat.h"
#include <cmath>
#include <iostream>

Worm::BaseballBat::BaseballBat(const GUI::GameTextureManager &tex)
    : Weapon(tex, GUI::GameTextures::WormBaseballBat, BASEBALL_BAT_CENTER_FRAME,
             WeaponID::WBaseballBat),
      scope(this→textureMgr) {}

void Worm::BaseballBat::update(float dt) {
    this→weaponAnimation.update(dt);
    this→scope.update(dt);
}

void Worm::BaseballBat::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererF
lip &flip) {
    this→weaponAnimation.render(p, cam, flip);
    this→scope.render(p, cam, flip);
}

void Worm::BaseballBat::setAngle(float angle, Direction d) {
    this→weaponAnimation.setFrame((int)std::ceil(angle / ANGLE_STEP) + this→ce
nterFrame);
    this→scope.setAngle(angle, d);
}

void Worm::BaseballBat::startShot() {}

void Worm::BaseballBat::endShot() {}

bool Worm::BaseballBat::positionSelected() {
    return false;
}
```

jun 26, 18 17:16 **Banana.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#ifndef __BANANA_H__
#define __BANANA_H__

#include <vector>

#include "PowerBar.h"
#include "Scope.h"
#include "Weapon.h"

#define BANANA_CENTER_FRAME 14

namespace Worm {
class Banana : public Weapon {
   public:
    explicit Banana(const GUI::GameTextureManager &textureManager);
    ~Banana() = default;
    void update(float dt) override;
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) over
ride;
    void setAngle(float angle, Direction d) override;
    void startShot() override;
    void endShot() override;
    bool positionSelected() override;

   private:
    ::Weapon::Scope scope;
    ::Weapon::PowerBar powerBar;
};
}  // namespace Worm

#endif  //__BANANA_H__
```

jun 26, 18 17:16 **Banana.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 04/06/18
 */

#include <cmath>

#include "Banana.h"

Worm::Banana::Banana(const GUI::GameTextureManager &tex)
    : Weapon(tex, GUI::GameTextures::WormBanana, BANANA_CENTER_FRAME, WeaponID::WBanana),
      scope(this→textureMgr),
      powerBar(this→textureMgr) {}

void Worm::Banana::update(float dt) {
    this→weaponAnimation.update(dt);
    this→scope.update(dt);
    this→powerBar.update(dt);
}

void Worm::Banana::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) {
    this→weaponAnimation.render(p, cam, flip);
    this→scope.render(p, cam, flip);
    this→powerBar.render(p, cam, flip);
}

void Worm::Banana::setAngle(float angle, Worm::Direction d) {
    this→weaponAnimation.setFrame((int)std::ceil(angle / ANGLE_STEP) + this→centerFrame);
    this→scope.setAngle(angle, d);
    this→powerBar.setAngle(angle, d);
}

void Worm::Banana::startShot() {
    this→powerBar.startShot();
}

void Worm::Banana::endShot() {
    this→powerBar.endShot();
}

bool Worm::Banana::positionSelected() {
    return false;
}
```

jun 26, 18 17:16 **AerialAttack.h** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 16/06/18
 */

#ifndef __AerialAttack_H__
#define __AerialAttack_H__

#define AERIAL_ATTACK_CENTER_FRAME 0

#include "Weapon.h"

namespace Worm {
class AerialAttack : public Weapon {
   public:
    explicit AerialAttack(const GUI::GameTextureManager &textureManager);
    ~AerialAttack() = default;
    void update(float dt) override;
    void render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) override;
    void setAngle(float angle, Direction d) override;
    void startShot() override;
    void endShot() override;
    bool positionSelected() override;

   private:
    void endAnimation();
};
}  // namespace Worm

#endif  //__AerialAttack_H__
```

jun 26, 18 17:16 **AerialAttack.cpp** Page 1/1

```cpp
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 16/06/18
 */

#include "AerialAttack.h"

Worm::AerialAttack::AerialAttack(const GUI::GameTextureManager &tex)
    : Weapon(tex, GUI::GameTextures::WormAirAttack, AERIAL_ATTACK_CENTER_FRAME, WeaponID::WAerial) {
    this→weaponAnimation.setAnimateOnce();
}

void Worm::AerialAttack::update(float dt) {
    if (¬this→weaponAnimation.finished()) {
        this→weaponAnimation.update(dt);
    } else {
        this→endAnimation();
    }
}

void Worm::AerialAttack::render(GUI::Position &p, GUI::Camera &cam, SDL_RendererFlip &flip) {
    this→weaponAnimation.render(p, cam, flip);
}

void Worm::AerialAttack::setAngle(float angle, Worm::Direction d) {}

void Worm::AerialAttack::startShot() {}

void Worm::AerialAttack::endShot() {}

bool Worm::AerialAttack::positionSelected() {
    this→weaponAnimation.setAutoUpdate(true);
    return true;
}

void Worm::AerialAttack::endAnimation() {
    this→weaponAnimation.setFrame(AERIAL_ATTACK_CENTER_FRAME);
    this→weaponAnimation.setAutoUpdate(false);
}
```

jun 26, 18 17:16 **Water.h** Page 1/1

```cpp
#ifndef WATER_H_
#define WATER_H_

#include "Camera.h"
#include "GameTextures.h"

namespace GUI {
class Water {
   public:
    Water(const GameTextureManager &tm);
    ~Water() = default;

    void update(float dt);
    void render(Camera &camera);

   private:
    const GUI::GameTextureManager &textureManager;
    float elapsed{0};
    float yDelta{0};
};
}  // namespace GUI

#endif
```

jun 26, 18 17:16 **Water.cpp** Page 1/1

```cpp
#include "Water.h"
#include <cmath>
#include "WrapTexture.h"

GUI::Water::Water(const GUI::GameTextureManager &tm) : textureManager(tm) {}

/**
 * @brief Updates the water animation state
 *
 * @param dt Time elapsed since the last call to this function.
 */
void GUI::Water::update(float dt) {
    this→elapsed += dt;
    this→yDelta = std::sin(this→elapsed) * 1;
}

/**
 * @brief Renders the water.
 *
 * @param camera Camera where the water is rendered.
 */
void GUI::Water::render(GUI::Camera &camera) {
    const GUI::Texture &texture = this→textureManager.get(GUI::GameTextures::Water);
    GUI::WrapTexture water{texture, camera.screenWidth(), texture.getHeight() / camera.getScale()};
    water.render(Position{camera.getPosition().x, −6.5f + this→yDelta}, camera);
}
```

jun 26, 18 17:16 **WaitingPlayersWindow.h** Page 1/1

```cpp
//
// Created by rodrigo on 24/06/18.
//

#ifndef INC_4_WORMS_WAITINGPLAYERSWINDOW_H
#define INC_4_WORMS_WAITINGPLAYERSWINDOW_H


#include <vector>

#include "Window.h"
#include "Font.h"
#include "GameStateMsg.h"
#include "GameWindow.h"
#include "Button.h"

namespace GUI {
    class WaitingPlayersWindow : public GameWindow {
    public:
        uint8_t playersConnected{0};

        WaitingPlayersWindow(GUI::Window &window, GUI::Font &font, GUI::Camera &cam, uint8_t playersQuantity);
        WaitingPlayersWindow(Window &window, Font &font, Camera &cam, uint8_t playersQuantity, uint8_t playersConnected);

        void start() override;
        void render() override;
        void handleKeyDown(SDL_Keycode key) override;
        void appendCharacter(char text[32]) override;
        void buttonPressed(ScreenPosition sp) override;

    private:
        std::vector<Button> buttons;
        unsigned int playersQuantity{0};
    };
}


#endif //INC_4_WORMS_WAITINGPLAYERSWINDOW_H
```

jun 26, 18 17:16 **WaitingPlayersWindow.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 24/06/18.
//

#include "WaitingPlayersWindow.h"

GUI::WaitingPlayersWindow::WaitingPlayersWindow(GUI::Window &window, GUI::Font &font, GUI::Camera &cam,
                                                uint8_t playersQuantity) :
        GameWindow(window, font, cam),
        playersQuantity(playersQuantity) {
}

GUI::WaitingPlayersWindow::WaitingPlayersWindow(GUI::Window &window, GUI::Font &font, GUI::Camera &cam,
                                                uint8_t playersQuantity, uint8_t playersConnected) :
        WaitingPlayersWindow(window, font, cam, playersQuantity) {
    this→playersConnected = playersConnected;
}

void GUI::WaitingPlayersWindow::start() {

}

void GUI::WaitingPlayersWindow::render() {
    this→window.clear(SDL_Color{0xFF, 0xFF, 0xFF});

    Text playersConnected{this→font};
    int x = this→window.getWidth() * 2 / 5;
    int y = this→window.getHeight() / 2;
    playersConnected.set("Players connected", SDL_Color{0, 0, 0}, 50);
    playersConnected.renderFixed(ScreenPosition{x, y}, this→cam);
    x = this→window.getWidth() * 3 / 5;
    y = this→window.getHeight() / 2;
    playersConnected.setBackground(SDL_Color{0, 0, 0});
    playersConnected.set(std::to_string(this→playersConnected) + "/" + std::to_string(this→playersQuantity), SDL_Color{0xFF, 0xFF, 0xFF}, 50);
    playersConnected.renderFixed(ScreenPosition{x, y}, this→cam);

    this→window.render();
}

void GUI::WaitingPlayersWindow::buttonPressed(GUI::ScreenPosition sp) {

}

void GUI::WaitingPlayersWindow::appendCharacter(char *text) {

}

void GUI::WaitingPlayersWindow::handleKeyDown(SDL_Keycode key) {

}
```

jun 26, 18 17:16 **SoundEffectPlayer.h** Page 1/1

```cpp
//
// Created by rodrigo on 5/06/18.
//

#ifndef INC_4_WORMS_SOUNDEFFECTPLAYER_H
#define INC_4_WORMS_SOUNDEFFECTPLAYER_H

#include <SDL2/SDL.h>

#include "SoundEffect.h"

namespace GUI {
class SoundEffectPlayer {
   public:
    bool loop{false};

    explicit SoundEffectPlayer(const GUI::SoundEffect &soundEffect);
    SoundEffectPlayer(const SoundEffect &soundEffect, float duration);
    SoundEffectPlayer(const GUI::SoundEffect &soundEffect, bool autoUpdate);
    ~SoundEffectPlayer();
    void update(float dt);
    void play();

   private:
    const SoundEffect *soundEffect;
    float duration{0.0f};
    float timeElapsed{0.0f};
    bool autoUpdate{false};
};
}

#endif  // INC_4_WORMS_SOUNDEFFECTPLAYER_H
```

jun 26, 18 17:16 **SoundEffectPlayer.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 5/06/18.
//

#include "SoundEffectPlayer.h"

GUI::SoundEffectPlayer::SoundEffectPlayer(const GUI::SoundEffect &soundEffect)
    : soundEffect(&soundEffect) {}

GUI::SoundEffectPlayer::SoundEffectPlayer(const GUI::SoundEffect &soundEffect, float duration)
    : soundEffect(&soundEffect), duration(duration) {
    //    this->soundEffect->play();
}

GUI::SoundEffectPlayer::SoundEffectPlayer(const GUI::SoundEffect &soundEffect, bool autoUpdate)
    : soundEffect(&soundEffect), autoUpdate(autoUpdate) {}

GUI::SoundEffectPlayer::~SoundEffectPlayer() {}

void GUI::SoundEffectPlayer::update(float dt) {
    if (¬this→autoUpdate) {
        this→timeElapsed += dt;
        if (this→timeElapsed > this→duration) {
            this→play();
            this→timeElapsed = 0.0f;
        }
    }
}

void GUI::SoundEffectPlayer::play() {
    this→soundEffect→play(this→loop);
}
```

jun 26, 18 17:16 **SoundEffectManager.h** Page 1/1

```cpp
//
// Created by rodrigo on 4/06/18.
//

#ifndef INC_4_WORMS_SOUNDEFFECTMANAGER_H
#define INC_4_WORMS_SOUNDEFFECTMANAGER_H

#include <SDL2/SDL.h>
#include <functional>
#include <string>
#include <unordered_map>
#include "SoundEffect.h"

namespace GUI {
template <typename ID, typename HASH = std::hash<ID>>
class SoundEffectManager {
   public:
    SoundEffectManager();
    ~SoundEffectManager();
    SoundEffectManager& operator=(SoundEffectManager& other) = delete;

    void load(ID id, const std::string& file_name);
    const SoundEffect& get(ID id) const;

   private:
    std::unordered_map<ID, SoundEffect, HASH> cache;
};
}  // namespace GUI

template <typename ID, typename HASH>
GUI::SoundEffectManager<ID, HASH>::SoundEffectManager() {}

template <typename ID, typename HASH>
GUI::SoundEffectManager<ID, HASH>::~SoundEffectManager() {}

/**

 * @brief Loads a sound effect.
 *
 * @param file_name The image file name.
 */
template <typename ID, typename HASH>
void GUI::SoundEffectManager<ID, HASH>::load(ID id, const std::string& file_name) {
    GUI::SoundEffect soundEffect{file_name};
    this→cache.insert(std::make_pair(id, std::move(soundEffect)));
}

/**
 * @brief Gets a sound effect.
 *
 * @param file_name Name of the sound effect.
 */
template <typename ID, typename HASH>
const GUI::SoundEffect& GUI::SoundEffectManager<ID, HASH>::get(ID id) const {
    return this→cache.at(id);
}

#endif  // INC_4_WORMS_SOUNDEFFECTMANAGER_H
```

jun 26, 18 17:16 **SoundEffect.h** Page 1/1

```
1  //
2  // Created by rodrigo on 4/06/18.
3  //
4
5  #ifndef INC_4_WORMS_SOUNDEFFECT_H
6  #define INC_4_WORMS_SOUNDEFFECT_H
7
8  #include <SDL2/SDL.h>
9  #include <SDL2/SDL_mixer.h>
10 #include <string>
11
12 namespace GUI {
13 class SoundEffect {
14    public:
15     SoundEffect(const std::string &filename);
16     SoundEffect(SoundEffect ∧other);
17     ~SoundEffect();
18     Mix_Chunk *getChunk() const;
19     void play(bool loop) const;
20
21    private:
22     Mix_Chunk *soundEffect{nullptr};
23 };
24 }
25
26 #endif  // INC_4_WORMS_SOUNDEFFECT_H
```

jun 26, 18 17:16 **SoundEffect.cpp** Page 1/1

```
1  //
2  // Created by rodrigo on 4/06/18.
3  //
4
5  #include "SoundEffect.h"
6  #include "Exception.h"
7
8  GUI::SoundEffect::SoundEffect(const std::string &filename) {
9      this→soundEffect = Mix_LoadWAV(filename.c_str());
10     if (¬this→soundEffect) {
11         throw Exception{"Error loading %s: %s", filename.c_str(), Mix_GetError()};
12     }
13 }
14
15 GUI::SoundEffect::~SoundEffect() {
16     if (this→soundEffect ≠ nullptr) {
17         Mix_FreeChunk(this→soundEffect);
18     }
19 }
20
21 Mix_Chunk *GUI::SoundEffect::getChunk() const {
22     return this→soundEffect;
23 }
24
25 GUI::SoundEffect::SoundEffect(GUI::SoundEffect ∧other) {
26     std::swap(this→soundEffect, other.soundEffect);
27 }
28
29 void GUI::SoundEffect::play(bool loop) const {
30     Mix_PlayChannel(−1, this→soundEffect, −1 * loop);
31 }
```

jun 26, 18 17:16 **SelectActionWindow.h** Page 1/1

```
1  //
2  // Created by rodrigo on 19/06/18.
3  //
4
5  #ifndef INC_4_WORMS_SELECTACTIONWINDOW_H
6  #define INC_4_WORMS_SELECTACTIONWINDOW_H
7
8
9  #include <vector>
10
11 #include "Window.h"
12 #include "Font.h"
13 #include "GameWindow.h"
14 #include "Button.h"
15
16 namespace GUI {
17     class SelectActionWindow : public GameWindow {
18     public:
19         explicit SelectActionWindow(Window &window, Font &font, Camera &cam);
20
21         void start() override;
22         void render() override;
23         void handleKeyDown(SDL_Keycode key) override;
24         void appendCharacter(char text[32]) override;
25         void buttonPressed(ScreenPosition sp) override;
26
27     private:
28         std::vector<Button> buttons;
29     };
30 }
31
32
33 #endif //INC_4_WORMS_SELECTACTIONWINDOW_H
```

jun 26, 18 17:16 **SelectActionWindow.cpp** Page 1/1

```
1  //
2  // Created by rodrigo on 19/06/18.
3  //
4
5  #include <SDL2/SDL.h>
6  #include <iostream>
7
8  #include "SelectActionWindow.h"
9  #include "Text.h"
10 #include "Window.h"
11
12 #define MSG_CREATE_GAME "Create game"
13 #define MSG_JOIN_GAME "Join game"
14
15 GUI::SelectActionWindow::SelectActionWindow(Window &window, Font &font, Camera &
   cam) :
16         GameWindow(window, font, cam) {
17     std::string msg(MSG_CREATE_GAME);
18     this→buttons.emplace_back(ScreenPosition{this→window.getWidth() / 4, this→
   window.getHeight() / 2},
19                               50, 300, msg, this→font);
20     msg = MSG_JOIN_GAME;
21     this→buttons.emplace_back(ScreenPosition{this→window.getWidth() * 3 / 4, t
   his→window.getHeight() / 2},
22                               50, 300, msg, this→font);
23 }
24
25 void GUI::SelectActionWindow::start() {
26
27 }
28
29 void GUI::SelectActionWindow::render() {
30     this→window.clear(SDL_Color{0xFF, 0xFF, 0xFF});
31     for (auto &button : this→buttons) {
32         button.render(this→cam);
33     }
34
35     this→window.render();
36 }
37
38 void GUI::SelectActionWindow::buttonPressed(GUI::ScreenPosition sp) {
39     if (this→buttons[0].inside(sp)) {
40         this→notify(*this, Event::CreateGame);
41     }
42
43     if (this→buttons[1].inside(sp)) {
44         this→notify(*this, Event::JoinGame);
45     }
46 }
47
48 void GUI::SelectActionWindow::appendCharacter(char *text) {
49
50 }
51
52 void GUI::SelectActionWindow::handleKeyDown(SDL_Keycode key) {
53
54 }
```

jun 26, 18 17:16 **main.cpp** Page 1/1

```cpp
/*
 * Created by Federico Manuel Gomez Peter
 * Date: 02/05/2018.
 */
#include <cstdlib>
#include <iostream>
#include <string>

#include "ClientSocket.h"
#include "GUIGame.h"
#include "LobbyAssistant.h"
#include "GameEndWindow.h"

int main(int argc, const char *argv[]) {
    if (argc ≠ 1) {
        std::cout << "Usage: ./client" << std::endl;
        return EXIT_FAILURE;
    }

    try {
        GUI::Window window{};
        window.clear();
        GUI::LobbyAssistant lobby(window);
        lobby.run();

        if (¬lobby.exit) {
            ClientSocket socket = std::move(lobby.getSocket());

            char buffer[1];
            socket.receive(buffer, sizeof(buffer));

            GUI::Game game{window, Worms::Stage::fromFile(lobby.levelPath), lobby.backgroundPath, socket,
                           (std::uint8_t) buffer[0]};
            game.start();

            GUI::GameEndWindow gameEndWindow(window, lobby.getFont(), lobby.getCam(), game.youWin);
            gameEndWindow.start();
        }
    } catch (std::exception &e) {
        std::cerr << "In main()" << std::endl;
        std::cerr << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cerr << "Unkown error in main thread" << std::endl;
        return 1;
    }
    return 0;
}
```

jun 26, 18 17:16 **JoinGameWindow.h** Page 1/1

```cpp
#ifndef JOIN_GAME_WINDOW_H_
#define JOIN_GAME_WINDOW_H_

#include <vector>
#include "../Button.h"
#include "../GameWindow.h"
#include "GameStateMsg.h"
#include "Text.h"
#include "Texture.h"

namespace GUI {
class JoinGameWindow : public GameWindow {
   public:
    JoinGameWindow(Window &window, Font &font, Camera &cam, std::vector<IO::GameInfo> &info);
    std::vector<IO::GameInfo> &info;
    uint8_t currentGameIndex{0};

    void start() override;
    void render() override;
    void handleKeyDown(SDL_Keycode key) override;
    void appendCharacter(char text[32]) override;
    void buttonPressed(ScreenPosition sp) override;

   private:
    Text gameName;
    Text numPlayers;
    Button prev;
    Button next;
    Button join;
};
}  // namespace GUI

#endif
```

```cpp
#include "JoinGameWindow.h"

const SDL_Color WHITE = {0xff, 0xff, 0xff};
const SDL_Color BLACK = {0, 0, 0};

const int TEXT_SIZE = 30;

GUI::JoinGameWindow::JoinGameWindow(Window &window, Font &font, Camera &cam,
                                    std::vector<IO::GameInfo> &info)
    : GameWindow(window, font, cam),
      info(info),
      gameName(font),
      numPlayers(font),
      prev("Previous", font),
      next("Next", font),
      join("Join", font) {
    int height = TEXT_SIZE * 3 / 2;
    this→prev.textColor = WHITE;
    this→prev.textSize = TEXT_SIZE;
    this→prev.position = {this→window.getWidth() / 4, this→window.getHeight() / 2};
    this→prev.height = height;
    this→prev.width = this→prev.msg.size() * 9 + 20;

    this→next.textColor = WHITE;
    this→next.textSize = TEXT_SIZE;
    this→next.position = {this→window.getWidth() * 3 / 4, this→window.getHeight() / 2};
    this→next.height = height;
    this→next.width = this→next.msg.size() * 9 + 20;

    this→join.textColor = WHITE;
    this→join.textSize = TEXT_SIZE;
    this→join.position = {this→window.getWidth() / 2, this→window.getHeight() * 3 / 4};
    this→join.height = height;
    this→join.width = this→join.msg.size() * 9 + 20;
}

/**
 * @brief Called when the window is started.
 *
 */
void GUI::JoinGameWindow::start() {}

/**
 * @brief Renders the window.
 *
 */
void GUI::JoinGameWindow::render() {
    this→window.clear(SDL_Color{0xFF, 0xFF, 0xFF});

    const ScreenPosition center{this→window.getWidth() / 2, this→window.getHeight() / 2};

    this→prev.render(this→cam);
    this→next.render(this→cam);

    if (this→info.size() > 0) {
        const IO::GameInfo &info = this→info.at(this→currentGameIndex);

        this→gameName.set("Game #" + std::to_string(info.gameID), BLACK, TEXT_SIZE * 2);
        this→gameName.renderFixed(center − ScreenPosition{0, this→window.getHeight() / 4},
                                  this→cam);
```



```cpp

        std::string msg =
            std::to_string(info.numCurrentPlayers) + "/" + std::to_string(info.numTotalPlayers);
        this→numPlayers.set(msg, BLACK, TEXT_SIZE * 2);
        this→numPlayers.renderFixed(center, this→cam);

        if (info.numCurrentPlayers < info.numTotalPlayers) {
            this→join.render(this→cam);
        }
    }

    this→window.render();
}

/**
 * @brief Checks if a button was pressed.
 *
 * @param sp Position where there was a click.
 */
void GUI::JoinGameWindow::buttonPressed(ScreenPosition sp) {
    if (this→prev.inside(sp)) {
        if (this→currentGameIndex ≡ 0) {
            this→currentGameIndex = static_cast<uint8_t>(this→info.size()) − 1;
        } else {
            this→currentGameIndex−−;
        }
    } else if (this→next.inside(sp)) {
        this→currentGameIndex = (this→currentGameIndex + 1) % this→info.size();
    } else if (this→join.inside(sp)) {
        const IO::GameInfo &info = this→info.at(this→currentGameIndex);
        if (info.numCurrentPlayers < info.numTotalPlayers) {
            this→notify(*this, Event::LobbyToJoinSelected);
        }
    }
}

/**
 * @brief Handles key press events.
 *
 * @param key Key pressed.
 */
void GUI::JoinGameWindow::handleKeyDown(SDL_Keycode key) {}

void GUI::JoinGameWindow::appendCharacter(char text[32]) {}
```

jun 26, 18 17:16 **LobbyAssistant.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 17/06/18
 */

#ifndef __LOBBY_ASSISTANT_H__
#define __LOBBY_ASSISTANT_H__

#include <Protocol.h>
#include <memory>
#include <Stream.h>
#include <Font.h>
#include <Camera.h>
#include "ClientSocket.h"
#include "CommunicationProtocol.h"
#include "Observer.h"
#include "Thread.h"
#include "GameWindow.h"
#include "GameStateMsg.h"

namespace GUI { // HabÃ-a una forward declaration con GameWindow pero no hace falta parece.
    class LobbyAssistant : public Observer {
    public:
        std::string levelPath;
        std::vector<std::string> backgroundPath;
        bool exit{false};

        explicit LobbyAssistant(Window &window);
        ~LobbyAssistant();
        //TODO overrrite
        void run();
        void onNotify(Subject &subject, Event event) override;

        ClientSocket getSocket();

        Font & getFont();

        Camera & getCam();

    private:
        Window &window;
        float scale{13.0f};
        bool quit{false};
        std::shared_ptr<GameWindow> gameWindow{nullptr};
        std::shared_ptr<GameWindow> nextGameWindow{nullptr};
        Font font;
        Camera cam;
        std::shared_ptr<IO::CommunicationProtocol> communicationProtocol;
        IO::Stream<IO::ClientGUIMsg> output;
        IO::Stream<IO::ServerResponse> serverStream;

        void handleServerResponse(IO::ServerResponse &response);
    };
} //namespace Worm


#endif //__LOBBY_ASSISTANT_H__
```

jun 26, 18 17:16 **LobbyAssistant.cpp** Page 1/4

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 17/06/18
 */

#include <iostream>
#include <GameStateMsg.h>
#include <SDL2/SDL.h>
#include <zconf.h>

#include "GameWindow.h"
#include "LobbyAssistant.h"
#include "Text.h"
#include "Window.h"
#include "SelectActionWindow.h"
#include "CreateGameWindow.h"
#include "WaitingPlayersWindow.h"
#include "Lobby/JoinGameWindow.h"
#include "ConnectionWindow.h"

GUI::LobbyAssistant::LobbyAssistant(Window &window) :
        window(window),
        font(std::string(ASSETS_PATH) + "/fonts/gruen_lemonograf.ttf", 28),
        cam(window, this→scale, 600, 600) {
    this→gameWindow = std::shared_ptr<GameWindow>(new ConnectionWindow{this→window, this→font, this→cam});
    this→gameWindow→addObserver(this);
}

void GUI::LobbyAssistant::run() {
    while (¬this→quit) {
        SDL_Event e;
        while (SDL_PollEvent(&e) ≠ 0) {
            switch (e.type) {
                case SDL_QUIT: {
                    this→quit = true;
                    this→exit = true;
                    break;
                }
                case SDL_KEYDOWN: {
                    this→gameWindow→handleKeyDown(e.key.keysym.sym);
                    break;
                }
                case SDL_KEYUP: {
                    break;
                }
                case SDL_TEXTINPUT: {
                    if(¬((e.text.text[0] ≡ 'c' ∨ e.text.text[0] ≡ 'C') ∧ (e.text.text[0] ≡ 'v' ∨ e.text.text[0] ≡ 'V') ∧ SDL_GetModState() & KMOD_CTRL)) {
                        //Append character
                        this→gameWindow→appendCharacter(e.text.text);
                    }
                    break;
                }
                case SDL_MOUSEBUTTONDOWN: {
                    int x, y;
                    SDL_GetMouseState(&x, &y);
                    GUI::Position global = this→cam.screenToGlobal(GUI::ScreenPosition{x, y});
                    this→gameWindow→buttonPressed(ScreenPosition{x, y});
                }
            }
        }

        IO::ServerResponse sr{};
        if (this→serverStream.pop(sr, false)) {
```

```
64              this→handleServerResponse(sr);
65          }
66
67          if (this→nextGameWindow) {
68              this→gameWindow = this→nextGameWindow;
69              this→nextGameWindow = nullptr;
70          }
71
72          if (this→gameWindow ≠ nullptr) {
73              this→gameWindow→render();
74          }
75
76          usleep(50 * 1000);
77      }
78  }
79
80
81
82  void GUI::LobbyAssistant::onNotify(Subject &subject, Event event) {
83      switch (event) {
84          case Event::ConnectionToServer: {
85              auto connectionWindow = dynamic_cast<ConnectionWindow *>(this→gameW
    indow.get());
86              ConnectionInfo info = connectionWindow→getConnectionInfo();
87              ClientSocket socket(info.ip, info.port);
88              this→communicationProtocol = std::shared_ptr<IO::CommunicationProto
    col>(
89                      new IO::CommunicationProtocol(socket, &this→output, &this→
    serverStream));
90              this→communicationProtocol→start();
91
92              this→nextGameWindow = std::shared_ptr<GameWindow>(new SelectActionW
    indow{this→window, this→font, this→cam});
93              this→nextGameWindow→addObserver(this);
94              break;
95          }
96          case Event::CreateGame: {
97              this→output << IO::ClientGUIMsg{IO::ClientGUIInput::startCreateGame
    };
98              break;
99          }
100         case Event::LevelSelected: {
101             auto createGameWindow = dynamic_cast<CreateGameWindow *>(this→gameW
    indow.get());
102             this→communicationProtocol→levelToCreate = createGameWindow→button
    Selected;
103             this→output << IO::ClientGUIMsg{IO::ClientGUIInput::levelSelected};
104             this→nextGameWindow = std::shared_ptr<GameWindow>(new WaitingPlayer
    sWindow{this→window,
105
        this→font,
106
        this→cam,
107
        createGameWindow→levelsInfo[createGameWindow→buttonSelected].playersQuanti
    ty});
108             this→nextGameWindow→addObserver(this);
109             break;
110         }
111         case Event::JoinGame: {
112             this→output << IO::ClientGUIMsg{IO::ClientGUIInput::startJoinGame};
113             break;
114         }
115         case Event::LobbyToJoinSelected: {
116             auto joinGameWindow = dynamic_cast<JoinGameWindow *>(this→gameWindo
    w.get());
```



```
117             this→communicationProtocol→gameToJoin = joinGameWindow→currentGame
    Index;
118             this→communicationProtocol→levelOfGameToJoin = joinGameWindow→info
    [joinGameWindow→currentGameIndex].levelID;
119             this→output << IO::ClientGUIMsg{IO::ClientGUIInput::joinGame};
120             this→nextGameWindow = std::shared_ptr<GameWindow>(new WaitingPlayer
    sWindow{this→window,
121
        this→font,
122
        this→cam,
123
        joinGameWindow→info[joinGameWindow→currentGameIndex].numTotalPlayers,
124
        joinGameWindow→info[joinGameWindow→currentGameIndex].numCurrentPlayers
    });
125             this→nextGameWindow→addObserver(this);
126             break;
127         }
128         default: {
129             break;
130         }
131     }
132 }
133
134 ClientSocket GUI::LobbyAssistant::getSocket() {
135     return std::move(this→communicationProtocol→getSocket());
136 }
137
138 void GUI::LobbyAssistant::handleServerResponse(IO::ServerResponse &response) {
139     switch (response.action) {
140         case IO::ServerResponseAction::startGame: {
141             this→levelPath = std::move(this→communicationProtocol→levelPath);
142             this→backgroundPath = std::move(this→communicationProtocol→backgro
    undPath);
143             this→output << IO::ClientGUIMsg{IO::ClientGUIInput::quit};
144             this→quit = true;
145             break;
146         }
147         case IO::ServerResponseAction::levelsInfo: {
148             this→nextGameWindow = std::shared_ptr<GameWindow>(new CreateGameWin
    dow{this→window,
149
    this→font,
150
    this→cam ,
151
    this→communicationProtocol→levelsInfo});
152             this→nextGameWindow→addObserver(this);
153             break;
154         }
155         case IO::ServerResponseAction::gamesInfo: {
156             this→nextGameWindow = std::shared_ptr<GameWindow>(new JoinGameWindo
    w{this→window,
157
      this→font,
158
      this→cam ,
159
      this→communicationProtocol→gamesInfo});
160             this→nextGameWindow→addObserver(this);
161             break;
162         }
163         case IO::ServerResponseAction::playerConnected: {
164             dynamic_cast<WaitingPlayersWindow *>(this→gameWindow.get())→player
    sConnected++;
```

```
165             break;
166         }
167         case IO::ServerResponseAction::serverClosed: {
168             this→quit = true;
169             this→exit = true;
170             this→gameWindow = nullptr;
171             break;
172         }
173         default: {
174             break;
175         }
176     }
177 }
178
179 GUI::Font & GUI::LobbyAssistant::getFont() {
180     return this→font;
181 }
182
183 GUI::Camera & GUI::LobbyAssistant::getCam() {
184     return this→cam;
185 }
186
187 GUI::LobbyAssistant::~LobbyAssistant() {
188     this→output.close();
189     this→serverStream.close();
190     if (this→communicationProtocol ≠ nullptr) {
191         this→communicationProtocol→stop();
192         this→communicationProtocol→join();
193     }
194 }
```



```
1  /*
2   *  Created by Federico Manuel Gomez Peter.
3   *  date: 18/05/18
4   */
5
6  #ifndef __GUIGame_H__
7  #define __GUIGame_H__
8
9  #include <atomic>
10 #include <list>
11 #include <thread>
12 #include <vector>
13
14 #include "Animation.h"
15 #include "Armory.h"
16 #include "Camera.h"
17 #include "ClientSocket.h"
18 #include "DoubleBuffer.h"
19 #include "Font.h"
20 #include "GameSoundEffects.h"
21 #include "GameStateMsg.h"
22 #include "GameTextures.h"
23 #include "Stage.h"
24 #include "Stream.h"
25 #include "TextureManager.h"
26 #include "Water.h"
27 #include "Weapons/Bullet.h"
28 #include "Weapons/Explosion.h"
29 #include "Wind.h"
30 #include "Window.h"
31 #include "Worm.h"
32 #include "BackgroundMusic.h"
33 #include "GameBackgroundMusic.h"
34 #include "BackgroundMusicPlayer.h"
35
36 namespace GUI {
37 using GameOutput = IO::Stream<IO::PlayerMsg>;
38 class Game {
39    public:
40     bool youWin{false};
41
42     Game(Window &w, Worms::Stage ∧stage, std::vector<std::string> &backgroundPa
   ths, ClientSocket &socket,
43              std::uint8_t team);
44     ~Game();
45     void start();
46     void update(float dt);
47     void render();
48
49     void exit();
50
51    private:
52     void renderStatic();
53     void renderBackground();
54     void handleCamera(float dt);
55
56     void inputWorker();
57     void outputWorker();
58
59     std::atomic<bool> quit{false};
60     float scale{13.0f};            // pixels per meter
61     float lastCameraUpdate{0.0f};  // pixels per meter
62     Window &window;
63     GameTextureManager texture_mgr;
64     GameSoundEffectManager sound_effect_mgr;
65     GameBackgroundMusicManager background_music_mgr;
```

jun 26, 18 17:16 **GUIGame.h** Page 2/2

```
66      std::vector<Worm::Worm> worms;
67      Worms::Stage stage;
68      std::list<std::shared_ptr<Ammo::Bullet>> bullets;
69      Camera cam;
70      Font font;
71      SDL_Color backgroundColor{0xba, 0x8d, 0xc6};
72      std::vector<SDL_Color> teamColors;
73      Armory armory;
74      std::thread inputThread;
75      std::thread outputThread;
76      IO::DoubleBuffer<IO::GameStateMsg> snapshotBuffer;
77      IO::GameStateMsg snapshot;
78      GameOutput output;
79      CommunicationSocket &socket;
80      std::uint8_t team{0};
81      uint8_t explodedQuantity{0};
82      GUI::Wind wind;
83      GUI::Water water;
84      std::unique_ptr<Animation> currentPlayerArrow{nullptr};
85      std::unique_ptr<GUI::BackgroundMusicPlayer> backGroundMusicPlayer{nullptr};
86
87      void loadTextureManager();
88      void loadBackgroundManager();
89      void loadSoundManager();
90  };
91  }  // namespace GUI
92
93  #endif  //__GUIGame_H__
```

jun 26, 18 17:16 **GUIGame.cpp** Page 1/12

```
1   /*
2    * Created by Federico Manuel Gomez Peter
3    * Date: 17/05/18.
4    */
5
6   #include <SDL2/SDL.h>
7   #include <unistd.h>
8   #include <cmath>
9   #include <iostream>
10  #include <sstream>
11
12  #include "GameStateMsg.h"
13  #include "GameWindow.h"
14  #include "GUIGame.h"
15  #include "Stream.h"
16  #include "Text.h"
17  #include "Weapons/Bullet.h"
18  #include "Window.h"
19  #include "WrapTexture.h"
20
21
22  // TODO DEHARDCODE
23  GUI::Game::Game(Window &w, Worms::Stage ∧stage, std::vector<std::string> &backg
    roundPaths, ClientSocket &socket,
24                  std::uint8_t team)
25      : window(w),
26        texture_mgr(w.getRenderer()),
27        sound_effect_mgr(),
28        stage(stage),
29        cam(w, this→scale, this→stage.getWidth(), this→stage.getHeight()),
30        font(std::string(ASSETS_PATH) + "/fonts/gruen_lemonograf.ttf", 28),
31        armory(this→texture_mgr, this→cam, this→font),
32        socket(socket),
33        team(team),
34        wind(this→texture_mgr, this→cam),
35        water(this→texture_mgr) {
36
37      this→loadTextureManager();
38      this→loadSoundManager();
39      this→loadBackgroundManager();
40
41      /* updates the armory */
42      this→armory.loadWeapons();
43      /* allocates space in the array to avoid the player addresses from changing
    */
44      int num_worms = 0;
45      this→worms.reserve(stage.getWorms().size());
46      for (const auto &wormData : this→stage.getWorms()) {
47          this→worms.emplace_back(num_worms, this→texture_mgr, this→sound_effect
    _mgr);
48          this→snapshot.positions[num_worms * 2] = wormData.position.x;
49          this→snapshot.positions[num_worms * 2 + 1] = wormData.position.y;
50          this→snapshot.wormsHealth[num_worms] = wormData.health;
51          num_worms += 1;
52      }
53
54      this→snapshot.num_worms = num_worms;
55      //    this->snapshot.processingInputs = true;
56
57      this→teamColors.push_back(SDL_Color{0xFF, 0, 0});
58      this→teamColors.push_back(SDL_Color{0, 0xFF, 0});
59      this→teamColors.push_back(SDL_Color{0, 0, 0xFF});
60      this→teamColors.push_back(SDL_Color{0xFF, 0, 0xFF});
61
62      this→currentPlayerArrow = std::unique_ptr<GUI::Animation>(
63          new GUI::Animation(this→texture_mgr.get(GUI::GameTextures::CurrentPlaye
```

```
    rArrow), false));
64      this→inputThread = std::thread([this] { this→inputWorker(); });
65      this→outputThread = std::thread([this] { this→outputWorker(); });
66
67      this→backGroundMusicPlayer =
68              std::unique_ptr<GUI::BackgroundMusicPlayer>(new GUI::BackgroundMusic
    Player{
69                      this→background_music_mgr.get(GUI::GameBackgroundMusic::Mur
    derTrain)});
70      this→backGroundMusicPlayer→play();
71  }
72
73  GUI::Game::~Game() {
74      this→exit();
75      this→outputThread.join();
76      this→inputThread.join();
77  }
78
79  void GUI::Game::inputWorker() {
80      IO::GameStateMsg msg;
81      try {
82          while (¬this→quit) {
83              /* receives the size of the msg */
84              std::uint32_t size(0);
85              socket.receive((char *)&size, sizeof(std::uint32_t));
86              size = ntohl(size);
87
88              std::vector<char> buffer(size, 0);
89              /* reads the raw data from the buffer */
90              socket.receive(buffer.data(), size);
91
92              std::string buff(buffer.data(), size);
93
94              /* sets the struct data from the buffer */
95              msg.deserialize(buff);
96              this→snapshotBuffer.set(msg);
97              this→snapshotBuffer.swap();
98          }
99      } catch (const std::exception &e) {
100         std::cerr << "GUI::Game::inputWorker:" << e.what() << std::endl;
101     } catch (...) {
102         std::cerr << "Unknown error in GUI::Game::inputWorker()" << std::endl;
103     }
104 }
105
106 void GUI::Game::outputWorker() {
107     IO::PlayerMsg msg;
108     try {
109         while (¬this→quit) {
110             this→output.pop(msg, true);
111             std::string buff = msg.serialize();
112             std::uint32_t size = buff.size();
113             std::uint32_t netSize = htonl(size);
114
115             this→socket.send((char *)&netSize, sizeof(std::uint32_t));
116             this→socket.send(buff.c_str(), size);
117         }
118     } catch (const std::exception &e) {
119         std::cerr << "GUI::Game::outputWorker:" << e.what() << std::endl;
120     }
121 }
122
123 //void GUI::Game::inputWorker() {
124 //    IO::GameStateMsg msg;
125 //    char *buffer = new char[msg.getSerializedSize()];
126 //
```



```
127 //    try {
128 //        while (!this->quit) {
129 //            this->socket.receive(buffer, msg.getSerializedSize());
130 //            msg.deserialize(buffer, msg.getSerializedSize());
131 //            this->snapshotBuffer.set(msg);
132 //            this->snapshotBuffer.swap();
133 //        }
134 //    } catch (const std::exception &e) {
135 //        std::cerr << "GUI::Game::inputWorker:" << e.what() << std::endl;
136 //    } catch (...) {
137 //        std::cerr << "Unknown error in GUI::Game::inputWorker()" << std::endl;
138 //    }
139 //
140 //    delete[] buffer;
141 //}
142 //
143 //void GUI::Game::outputWorker() {
144 //    IO::PlayerMsg msg;
145 //    char *buffer = new char[msg.getSerializedSize()];
146 //
147 //    try {
148 //        while (!this->quit) {
149 //            this->output.pop(msg, true);
150 //            msg.serialize(buffer, msg.getSerializedSize());
151 //            this->socket.send(buffer, msg.getSerializedSize());
152 //        }
153 //    } catch (const std::exception &e) {
154 //        std::cerr << "GUI::Game::outputWorker:" << e.what() << std::endl;
155 //    } catch (...) {
156 //        std::cerr << "Unknown error in GUI::Game::outputWorker()" << std::endl
    ;
157 //    }
158 //
159 //    delete[] buffer;
160 //}
161
162 void GUI::Game::start() {
163     try {
164         uint32_t prev = SDL_GetTicks();
165         while (¬this→quit) {
166             /* updates the snapshot */
167             this→snapshot = this→snapshotBuffer.get();
168             if (¬this→snapshot.gameEnded) {
169                 Worm::Worm &cur = this→worms[this→snapshot.currentWorm];
170
171                 /* handle events on queue */
172                 SDL_Event e;
173                 while (SDL_PollEvent(&e) ≠ 0) {
174                     switch (e.type) {
175                         case SDL_QUIT:
176                             this→exit();
177                             break;
178                         case SDL_KEYDOWN:
179                             if (this→snapshot.processingInputs ∧
180                                 this→team ≡ this→snapshot.currentTeam) {
181                                 cur.handleKeyDown(e.key.keysym.sym, &this→outpu
    t);
182                             }
183                             break;
184                         case SDL_KEYUP:
185                             if (this→snapshot.processingInputs ∧
186                                 this→team ≡ this→snapshot.currentTeam) {
187                                 cur.handleKeyUp(e.key.keysym.sym, &this→output)
    ;
188                             }
189                             break;
```

```
190                         case SDL_MOUSEBUTTONDOWN: {
191                             if (this→snapshot.processingInputs ∧
192                                 this→team ≡ this→snapshot.currentTeam) {
193                                 int x, y;
194                                 SDL_GetMouseState(&x, &y);
195                                 GUI::Position global =
196                                     this→cam.screenToGlobal(GUI::ScreenPosition
    {x, y});
197                                 cur.mouseButtonDown(global, &this→output);
198                                 break;
199                             }
200                         }
201                         default:
202                             break;
203                     }
204                 }
205
206                 /* synchronizes the worms states with the server's */
207                 for (std::size_t i = 0; i < this→worms.size(); i++) {
208                     this→worms[i].setState(this→snapshot.stateIDs[i]);
209                     this→worms[i].setWeapon((i ≠ this→snapshot.currentWorm)
210                                             ? Worm::WeaponID::WNone
211                                             : this→snapshot.activePlayerWe
    apon);
212                 }
213
214                 if (cur.getState() ≡ Worm::StateID::Still ∧
215                     cur.getWeaponID() ≠ Worm::WeaponID::WNone) {
216                     cur.setWeaponAngle(this→snapshot.activePlayerAngle);
217                 }
218                 if (this→snapshot.bulletsQuantity ≡ 0 ∧ this→snapshot.playerUs
    edTool) {
219                     this→bullets.erase(this→bullets.begin(), this→bullets.end(
    ));
220                     this→explodedQuantity = 0;
221                     this→worms[this→snapshot.currentWorm].reset();
222                 }
223                 if (this→snapshot.bulletsQuantity > 0) {
224                     for (int i = this→bullets.size(); i < this→snapshot.bullet
    sQuantity; i++) {
225                         std::shared_ptr<Ammo::Bullet> p(
226                             new Ammo::Bullet(this→texture_mgr, this→sound_effe
    ct_mgr,
227                                              this→snapshot.bulletType[i]));
228                         this→bullets.emplace_back(p);
229                     }
230                     int i = 0;
231                     for (auto &bullet : this→bullets) {
232                         if (this→snapshot.bulletType[i] ≡ Worm::WeaponID::WExpl
    ode ∧
233                             ¬bullet→exploding()) {
234                             bullet→madeImpact();
235                             this→explodedQuantity++;
236                         }
237                         bullet→setAngle(this→snapshot.bulletsAngle[i++]);
238                     }
239                 }
240
241                 uint32_t current = SDL_GetTicks();
242                 float dt = static_cast<float>(current − prev) / 1000.0f;
243                 prev = current;
244
245                 this→handleCamera(dt);
246                 this→update(dt);
247
248                 this→render();
```



```
249             } else {
250                 this→youWin = this→snapshot.winner ≡ this→team;
251                 this→quit = true;
252             }
253         }
254     } catch (std::exception &e) {
255         std::cerr << e.what() << std::endl << "In GUI::Game::start" << std::endl;
256     } catch (...) {
257         std::cerr << "Unkown error in GUI::Game::start()." << std::endl;
258     }
259 }
260
261 void GUI::Game::update(float dt) {
262     for (auto &worm : this→worms) {
263         worm.health = this→snapshot.wormsHealth[static_cast<int>(worm.id)];
264         worm.direction = this→snapshot.wormsDirection[static_cast<int>(worm.id)
    ];
265         worm.update(dt);
266     }
267     if (this→snapshot.waitingForNextTurn) {
268         this→armory.update(this→snapshot);
269         this→currentPlayerArrow→update(dt);
270     } else {
271         this→currentPlayerArrow→setFrame(0);
272     }
273
274     this→cam.update(dt);
275
276     for (auto &bullet : this→bullets) {
277         bullet→update(dt);
278     }
279
280     this→water.update(dt);
281 }
282
283 void GUI::Game::render() {
284     this→renderBackground();
285
286     for (uint8_t i = 0; i < this→snapshot.num_worms; i++) {
287         float cur_x = this→snapshot.positions[i * 2];
288         float cur_y = this→snapshot.positions[i * 2 + 1];
289
290         GUI::Position p{cur_x, cur_y};
291         this→worms[i].setPosition(p);
292         this→worms[i].render(p, this→cam);
293     }
294
295     for (auto &girder : this→stage.getGirders()) {
296         const GUI::Texture &texture = this→texture_mgr.get(GUI::GameTextures::L
    ongGirder);
297
298         GUI::WrapTexture wt{texture, girder.length, girder.height};
299         wt.render(GUI::Position{girder.pos.x, girder.pos.y}, girder.angle, this
    →cam);
300     }
301
302     int i = 0, j = 0;
303     for (auto &bullet : this→bullets) {
304         float local_x = this→snapshot.bullets[i++];
305         float local_y = this→snapshot.bullets[i++];
306         if (¬bullet→exploding()) {
307             bullet→setAngle(this→snapshot.bulletsAngle[j++]);
308             bullet→setPosition(GUI::Position{local_x, local_y});
309         }
310
311         if (¬bullet→exploded()) {
```

```
312             bullet→render(GUI::Position{local_x, local_y}, this→cam);
313         }
314     }
315
316     /* health bars are renderer after the worms so they appear on top */
317     for (uint8_t i = 0; i < this→snapshot.num_worms; i++) {
318         float cur_x = this→snapshot.positions[i * 2];
319         float cur_y = this→snapshot.positions[i * 2 + 1];
320         if (this→worms[i].getState() ≠ Worm::StateID::Dead) {
321             Text health{this→font};
322             health.setBackground(SDL_Color{0, 0, 0});
323             health.set(std::to_string(static_cast<int>(this→worms[i].health)),
324                        this→teamColors[this→snapshot.wormsTeam[i]], 20);
325             health.render(GUI::Position{cur_x, cur_y + 2.2f}, this→cam);
326         }
327     }
328
329     this→water.render(this→cam);
330
331     this→renderStatic();
332
333     this→window.render();
334 }
335
336 /**
337  * @brief interrupts all current game operations and leaves the main loop.
338  *
339  */
340 void GUI::Game::exit() {
341     this→quit = true;
342     this→output.close();
343     this→socket.shutdown();
344 }
345
346 /**
347  * @brief Renders the background images using a parallax effect.
348  *
349  */
350 void GUI::Game::renderBackground() {
351     SDL_Color bgColor{this→stage.backgroundColor.r, this→stage.backgroundColor
    .g, this→stage.backgroundColor.b};
352     this→window.clear(bgColor);
353
354     /* draws moving image further in the background */
355     const Texture &Bg1Tex = this→texture_mgr.get(GameTextures::Background1);
356     // TODO: use the stage size
357     WrapTexture bg1{Bg1Tex, this→stage.getWidth(), Bg1Tex.getHeight() / this→c
    am.getScale()};
358
359     Position pos{0.0f, (Bg1Tex.getHeight() / this→cam.getScale()) / 2};
360     pos.x += this→cam.getPosition().x * 0.8f;
361     bg1.render(pos, this→cam);
362
363     /* draws a moving image in the background at intermediate distance */
364     const Texture &Bg2Tex = this→texture_mgr.get(GameTextures::Background2);
365     // TODO: use the stage size
366     WrapTexture bg2{Bg2Tex, this→stage.getWidth(), Bg2Tex.getHeight() / this→c
    am.getScale()};
367
368     pos = {0.0f, (Bg2Tex.getHeight() / this→cam.getScale()) / 2};
369     pos.x += this→cam.getPosition().x * 0.6f;
370     bg2.render(pos, this→cam);
371
372     /* draws a moving image in the background at a closer distance */
373     const Texture &Bg3Tex = this→texture_mgr.get(GameTextures::Background3);
374     // TODO: use the stage size
```



```
375     WrapTexture bg3{Bg3Tex, this→stage.getWidth(), Bg3Tex.getHeight() / this→c
    am.getScale()};
376
377     pos = {0.0f, (Bg3Tex.getHeight() / this→cam.getScale()) / 2};
378     pos.x += this→cam.getPosition().x * 0.25f;
379     bg3.render(pos, this→cam);
380 }
381
382 /**
383  * @brief Draws the game controls.
384  */
385 void GUI::Game::renderStatic() {
386
387     /* render the arrow to notify the current player when wainting for next turn
    */
388     if (this→snapshot.waitingForNextTurn) {
389         float cur_x = this→snapshot.positions[this→snapshot.currentWorm * 2];
390         float cur_y = this→snapshot.positions[this→snapshot.currentWorm * 2 +
    1];
391         GUI::Position position = GUI::Position{cur_x, cur_y + 4.4f};
392         this→currentPlayerArrow→render(position, this→cam, SDL_FLIP_NONE);
393     }
394
395     /* health bars of the team */
396     uint8_t numTeams = this→snapshot.num_teams;
397     int textHeight = 25;
398     for (uint8_t i = 0; i < numTeams; i++) {
399         Text health{this→font};
400         std::ostringstream oss;
401         oss << "Team " << i + 1 << ":" << this→snapshot.teamHealths[i];
402
403         health.setBackground(SDL_Color{0, 0, 0});
404         health.set(oss.str(), this→teamColors[i], textHeight);
405         int x = this→window.getWidth() / 2;
406         int y = this→window.getHeight() - (textHeight * (numTeams - i));
407         health.renderFixed(ScreenPosition{x, y}, this→cam);
408     }
409
410     /* displays the remaining turn time */
411     std::int16_t turnTimeLeft =
412         this→snapshot.currentPlayerTurnTime - this→snapshot.elapsedTurnSeconds
    ;
413     turnTimeLeft = (turnTimeLeft < 0) ? 0 : turnTimeLeft;
414
415     int x = this→window.getWidth() / 2;
416     int y = 20;
417
418     SDL_Color color = {0, 0, 0};
419     Text text{this→font};
420     text.set(std::to_string(turnTimeLeft), color);
421     text.renderFixed(ScreenPosition{x, y}, this→cam);
422
423     /* renders armory */
424     this→armory.render();
425
426     this→wind.render(this→snapshot.windIntensity, this→window.getWidth());
427 }
428
429 /**
430  * @brief Handles the camera actions.
431  *
432  * @param dt Seconds elapsed since the last call to this function.
433  */
434 void GUI::Game::handleCamera(float dt) {
435     this→lastCameraUpdate += dt;
436
```

```
437     /* checks the mouse to see if the user wishes to move the camera */
438     int mx, my;
439     SDL_GetMouseState(&mx, &my);
440 
441     const float cameraSpeed = 15.0f;
442     const int cameraMargin = 50;
443 
444     /* checks if the camera should be moved horizontally */
445     if (this→window.containsMouse()) {
446         if (mx < cameraMargin) {
447             auto p = this→cam.getPosition() − GUI::Position{cameraSpeed, 0.0f}
    * dt;
448             this→cam.moveTo(this→cam.getPosition() − GUI::Position{cameraSpeed
    , 0.0f} * dt);
449             this→lastCameraUpdate = 0.0f;
450         } else if (mx > this→window.getWidth() − cameraMargin) {
451             this→cam.moveTo(this→cam.getPosition() + GUI::Position{cameraSpeed
    , 0.0f} * dt);
452             this→lastCameraUpdate = 0.0f;
453         }
454 
455         /* checks if the camera should be moved vertically */
456         if (my < cameraMargin) {
457             this→cam.moveTo(this→cam.getPosition() + GUI::Position{0.0f, camer
    aSpeed} * dt);
458             this→lastCameraUpdate = 0.0f;
459         } else if (my > this→window.getHeight() − cameraMargin) {
460             this→cam.moveTo(this→cam.getPosition() − GUI::Position{0.0f, camer
    aSpeed} * dt);
461             this→lastCameraUpdate = 0.0f;
462         }
463     }
464 
465     /* if the user hasn't changed the camera in a while, it becomes automatic ag
    ain */
466     if (this→lastCameraUpdate < 2.0f) {
467         return;
468     } else {
469         /* avoids overflow */
470         this→lastCameraUpdate = 2.0f;
471     }
472 
473     /* move the camera to the current player */
474     if (this→snapshot.bulletsQuantity > this→explodedQuantity) {
475         float cur_x{0};
476         float cur_y{0};
477         int i{0};
478         for (int j = 0; i < this→snapshot.bulletsQuantity; i++) {
479             if (this→snapshot.bulletType[i] ≠ Worm::WExplode) {
480                 cur_x = this→snapshot.bullets[j++];
481                 cur_y = this→snapshot.bullets[j];
482                 break;
483             }
484             j += 2;
485         }
486 
487         this→cam.moveTo(GUI::Position{cur_x, cur_y});
488     } else {
489         float cur_follow_x = this→snapshot.positions[this→snapshot.currentWorm
    ToFollow * 2];
490         float cur_follow_y = this→snapshot.positions[this→snapshot.currentWorm
    ToFollow * 2 + 1];
491 
492         /* move the camera to the current player */
493         this→cam.moveTo(GUI::Position{cur_follow_x, cur_follow_y});
494     }
```



```
495 }
496 
497 void GUI::Game::loadTextureManager(){
498     std::string path(ASSETS_PATH);
499     /* loads the required textures */
500     this→texture_mgr.load(GUI::GameTextures::CurrentPlayerArrow, path + "/img/Mis
    c/arrowdnb.png",
501                           GUI::Color{0x40, 0x40, 0x80});
502     this→texture_mgr.load(GUI::GameTextures::WindLeft, path + "/img/Misc/windl.png",
503                           GUI::Color{0x00, 0x00, 0x00});
504     this→texture_mgr.load(GUI::GameTextures::WindRight, path + "/img/Misc/windr.png"
    ,
505                           GUI::Color{0x00, 0x00, 0x00});
506     this→texture_mgr.load(GUI::GameTextures::WormWalk, path + "/img/Worms/wwalk2.pn
    g",
507                           GUI::Color{0x7f, 0x7f, 0xbb});
508     this→texture_mgr.load(GUI::GameTextures::WormIdle, path + "/img/Worms/wbrth1.pn
    g",
509                           GUI::Color{0x7f, 0x7f, 0xbb});
510     this→texture_mgr.load(GUI::GameTextures::LongGirder, path + "/img/Weapons/grdl4.
    png",
511                           GUI::Color{0x7f, 0x7f, 0xbb});
512     this→texture_mgr.load(GUI::GameTextures::StartJump, path + "/img/Worms/wjump.p
    ng",
513                           GUI::Color{0x7f, 0x7f, 0xbb});
514     this→texture_mgr.load(GUI::GameTextures::Jumping, path + "/img/Worms/wflyup.png
    ",
515                           GUI::Color{0x7f, 0x7f, 0xbb});
516     this→texture_mgr.load(GUI::GameTextures::EndJump, path + "/img/Worms/wland2.png
    ",
517                           GUI::Color{0x7f, 0x7f, 0xbb});
518     this→texture_mgr.load(GUI::GameTextures::BackFlipping, path + "/img/Worms/wba
    ckflp.png",
519                           GUI::Color{0x7f, 0x7f, 0xbb});
520     this→texture_mgr.load(GUI::GameTextures::Falling, path + "/img/Worms/wfall.png",
521                           GUI::Color{0x7f, 0x7f, 0xbb});
522     this→texture_mgr.load(GUI::GameTextures::Bazooka, path + "/img/Worms/wbaz.png",
523                           GUI::Color{0x7f, 0x7f, 0xbb});
524     this→texture_mgr.load(GUI::GameTextures::Fly, path + "/img/Worms/wfly1.png",
525                           GUI::Color{0x7f, 0x7f, 0xbb});
526     this→texture_mgr.load(GUI::GameTextures::Die, path + "/img/Worms/wdie.png",
527                           GUI::Color{0x7f, 0x7f, 0xbb});
528     this→texture_mgr.load(GUI::GameTextures::Sliding, path + "/img/Worms/wslided.png
    ",
529                           GUI::Color{0x7f, 0x7f, 0xbb});
530     this→texture_mgr.load(GUI::GameTextures::Dead, path + "/img/Misc/grave4.png",
531                           GUI::Color{0xC0, 0xC0, 0x80});
532     this→texture_mgr.load(GUI::GameTextures::Missile, path + "/img/Weapons/missile.pn
    g",
533                           GUI::Color{0x7f, 0x7f, 0xbb});
534     this→texture_mgr.load(GUI::GameTextures::Explosion, path + "/img/Effects/circle25.p
    ng",
535                           GUI::Color{0x80, 0x80, 0xC0});
536     this→texture_mgr.load(GUI::GameTextures::Flame, path + "/img/Effects/flame1.png",
537                           GUI::Color{0x80, 0x80, 0xC0});
538     this→texture_mgr.load(GUI::GameTextures::Smoke, path + "/img/Effects/smkdrk20.png"
    ,
539                           GUI::Color{0xC0, 0xC0, 0x80});
540     this→texture_mgr.load(GUI::GameTextures::Background1, path + "/img/background/b
    g1.png",
541                           GUI::Color{0xff, 0xff, 0xff});
542     this→texture_mgr.load(GUI::GameTextures::Background2, path + "/img/background/b
    g2.png",
543                           GUI::Color{0xff, 0xff, 0xff});
544     this→texture_mgr.load(GUI::GameTextures::Background3, path + "/img/background/b
    g3.png",
```

```
                                GUI::Color{0xff, 0xff, 0xff});
    this→texture_mgr.load(GUI::GameTextures::WormGrenade, path + "/img/Worms/wthrgrn.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::Grenade, path + "/img/Weapons/grenade.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::WormCluster, path + "/img/Worms/wthrcls.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::Cluster, path + "/img/Weapons/cluster.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::Mortar, path + "/img/Weapons/mortar.png",
                                GUI::Color{0xc0, 0xc0, 0x80});
    this→texture_mgr.load(GUI::GameTextures::Bazooka2, path + "/img/Worms/wbaz2.png",
                                GUI::Color{0xc0, 0xc0, 0x80});
    this→texture_mgr.load(GUI::GameTextures::Banana, path + "/img/Weapons/banana.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::WormBanana, path + "/img/Worms/wthrban.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::Holy, path + "/img/Weapons/hgrenade.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::WormHoly, path + "/img/Worms/wthrhgrd.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::Scope, path + "/img/Misc/crshairb.png",
                                GUI::Color{0x40, 0x40, 0x80});
    this→texture_mgr.load(GUI::GameTextures::Scope, path + "/img/Misc/crshairb.png",
                                GUI::Color{0x40, 0x40, 0x80});
    this→texture_mgr.load(GUI::GameTextures::PowerBar, path + "/img/Effects/blob.png",
                                GUI::Color{0x80, 0x80, 0xC0});
    this→texture_mgr.load(GUI::GameTextures::Fragment, path + "/img/Weapons/clustlet.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::WormAirAttack, path + "/img/Worms/wairtlk.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::AirMissile, path + "/img/Weapons/airmisl.png",
                                GUI::Color{0xc0, 0xc0, 0x80});
    this→texture_mgr.load(GUI::GameTextures::WormDynamite, path + "/img/Worms/wdynbak.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::Dynamite, path + "/img/Weapons/dynamite.png",
                                GUI::Color{0x7f, 0x7f, 0xbb});
    this→texture_mgr.load(GUI::GameTextures::WormBaseballBat, path + "/img/Worms/wbsbaim.png",
                                GUI::Color{0xc0, 0xc0, 0x80});
    this→texture_mgr.load(GUI::GameTextures::WormBaseballBatting, path + "/img/Worms/wbsbswn.png",
                                GUI::Color{0xc0, 0xc0, 0x80});
    this→texture_mgr.load(GUI::GameTextures::WormTeleport, path + "/img/Worms/wteltlk.png",
                                GUI::Color{0xc0, 0xc0, 0x80});
    this→texture_mgr.load(GUI::GameTextures::WormTeleporting, path + "/img/Worms/wteldsv.png",
                                GUI::Color{0xc0, 0xc0, 0x80});
    this→texture_mgr.load(GUI::GameTextures::BazookaIcon, path + "/img/Weapon Icons
```



```
/bazooka.2.png",
                                GUI::Color{0x00, 0x00, 0x00});
    this→texture_mgr.load(GUI::GameTextures::GrenadeIcon, path + "/img/Weapon Icons/grenade.2.png",
                                GUI::Color{0x00, 0x00, 0x00});
    this→texture_mgr.load(GUI::GameTextures::ClusterIcon, path + "/img/Weapon Icons/cluster.2.png",
                                GUI::Color{0x00, 0x00, 0x00});
    this→texture_mgr.load(GUI::GameTextures::MortarIcon, path + "/img/Weapon Icons/mortar.2.png",
                                GUI::Color{0x00, 0x00, 0x00});
    this→texture_mgr.load(GUI::GameTextures::BananaIcon, path + "/img/Weapon Icons/banana.2.png",
                                GUI::Color{0x00, 0x00, 0x00});
    this→texture_mgr.load(GUI::GameTextures::HolyIcon, path + "/img/Weapon Icons/hgrenade.2.png",
                                GUI::Color{0x00, 0x00, 0x00});
    this→texture_mgr.load(GUI::GameTextures::AirIcon, path + "/img/Weapon Icons/airstrike.1.png",
                                GUI::Color{0x00, 0x00, 0x00});
    this→texture_mgr.load(GUI::GameTextures::DynamiteIcon,
                                path + "/img/Weapon Icons/dynamite.1.png", GUI::Color{0x00, 0x00, 0x00});
    this→texture_mgr.load(GUI::GameTextures::BaseballBatIcon,
                                path + "/img/Weapon Icons/baseball.1.png", GUI::Color{0x00, 0x00, 0x00});
    this→texture_mgr.load(GUI::GameTextures::TeleportIcon,
                                path + "/img/Weapon Icons/teleport.1.png", GUI::Color{0x00, 0x00, 0x00});
    this→texture_mgr.load(GUI::GameTextures::Water,
                                path + "/img/background/water.png", GUI::Color{0x00, 0x00, 0x00});
}

void GUI::Game::loadSoundManager(){
    std::string path(ASSETS_PATH);
    this→sound_effect_mgr.load(GUI::GameSoundEffects::WalkCompress,
                                path + "/sound/Effects/Walk−Compress.wav");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::WormJump,
                                path + "/sound/Soundbanks/JUMP1.WAV");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::WormBackFlip,
                                path + "/sound/Soundbanks/JUMP2.WAV");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::WormLanding,
                                path + "/sound/Effects/WormLanding.wav");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::WormHit, path + "/sound/Soundbanks/OUCH.WAV");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::WormDrowning,
                                path + "/sound/Effects/UnderWaterLoop.wav");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::WormDie,
                                path + "/sound/Soundbanks/BYEBYE.WAV");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::Splash, path + "/sound/Effects/Splash.wav");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::Explosion,
                                path + "/sound/Effects/Explosion1.wav");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::Holy,
                                path + "/sound/Effects/HOLYGRENADE.WAV");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::AirStrike,
                                path + "/sound/Effects/Airstrike.wav");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::Teleport,
                                path + "/sound/Effects/TELEPORT.WAV");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::Shot,
                                path + "/sound/Effects/ROCKETRELEASE.WAV");
    this→sound_effect_mgr.load(GUI::GameSoundEffects::Banana,
                                path + "/sound/Effects/BananaImpact.wav");
}
```

```
644 void GUI::Game::loadBackgroundManager(){
645     std::string path(ASSETS_PATH);
646     this→background_music_mgr.load(GUI::GameBackgroundMusic::Original,
647                                    path + "/sound/Background/background.wav");
648     this→background_music_mgr.load(GUI::GameBackgroundMusic::MurderTrain, path
    + "/sound/Background/MurderTrain.wav");
649 }
```



```
1  //
2  // Created by rodrigo on 19/06/18.
3  //
4
5  #ifndef INC_4_WORMS_GAMEWINDOW_H
6  #define INC_4_WORMS_GAMEWINDOW_H
7
8
9  #include <vector>
10
11 #include "Button.h"
12 #include "Camera.h"
13 #include "Font.h"
14 #include "Subject.h"
15 #include "Window.h"
16
17 #define ASSETS_PATH "/var/Worms/assets"
18
19
20 namespace GUI {
21     struct TextField {
22         TextField(std::string &text, ScreenPosition sp, int height, int width, F
    ont &font) :
23                 inputText(sp, height, width, text, font),
24                 focus(false) {};
25
26         void selected(ScreenPosition sp) {
27             this→focus = inputText.inside(sp);
28         };
29
30         void render(GUI::Camera &cam) {
31             this→inputText.render(cam);
32         };
33
34         void appendCharacter(char *text) {
35             if (this→emptyString) {
36                 this→inputText.msg = text;
37                 this→emptyString = false;
38             } else {
39                 this→inputText.msg += text;
40             }
41         };
42
43         void backSpace() {
44             if (¬this→emptyString) {
45                 this→inputText.msg.pop_back();
46                 if (this→inputText.msg.length() ≡ 0) {
47                     this→inputText.msg = "";
48                     this→emptyString = true;
49                 }
50             }
51         };
52
53         Button inputText;
54         bool focus;
55
56     private:
57         bool emptyString{true};
58     };
59
60     class GameWindow : public Subject {
61     public:
62         uint8_t buttonSelected{0};
63
64         explicit GameWindow(Window &window, Font &font, Camera &cam);
65
```

jun 26, 18 17:16 **GameWindow.h** Page 2/2

```
66          virtual void start() = 0;
67          virtual void render() = 0;
68          virtual void handleKeyDown(SDL_Keycode key) = 0;
69          virtual void appendCharacter(char text[32]) = 0;
70          virtual void buttonPressed(ScreenPosition sp) = 0;
71
72      protected:
73          Window &window;
74          Font &font;
75          Camera &cam;
76          std::vector<TextField> textFields;
77          bool quit{false};
78      };
79  }
80
81
82  #endif //INC_4_WORMS_GAMEWINDOW_H
```

jun 26, 18 17:16 **GameWindow.cpp** Page 1/1

```
1   //
2   // Created by rodrigo on 19/06/18.
3   //
4
5   #include <Font.h>
6   #include <Camera.h>
7   #include "GameWindow.h"
8
9   GUI::GameWindow::GameWindow(Window &window, Font &font, Camera &cam) :
10          window(window),
11          font(font),
12          cam(cam) {
13  }
```

```
#ifndef GAME_TEXTURES_H_
#define GAME_TEXTURES_H_

#include "TextureManager.h"
#include "utils.h"

namespace GUI {
/** Different kinds of textures. */
enum class GameTextures {
    WormWalk,
    WormIdle,
    LongGirder,
    ShortGirder,
    StartJump,
    Jumping,
    EndJump,
    BackFlipping,
    Bazooka,
    Missile,
    Fly,
    Die,
    Dead,
    Sliding,
    StaticBackground,
    Background1,
    Background2,
    Background3,
    WormGrenade,
    Grenade,
    WormCluster,
    Cluster,
    Mortar,
    Bazooka2,
    WormBanana,
    Banana,
    WormHoly,
    Holy,
    Explosion,
    Flame,
    Smoke,
    Falling,
    Scope,
    PowerBar,
    Fragment,
    BazookaIcon,
    GrenadeIcon,
    ClusterIcon,
    MortarIcon,
    BananaIcon,
    HolyIcon,
    WormAirAttack,
    AirMissile,
    AirIcon,
    WormDynamite,
    Dynamite,
    DynamiteIcon,
    WormTeleport,
    WormTeleporting,
    TeleportIcon,
    WormBaseballBat,
    WormBaseballBatting,
    BaseballBatIcon,
    WindLeft,
    WindRight,
    CurrentPlayerArrow,
    Water,
```



```
};

/** Specialized TextureManager class. */
using GameTextureManager = TextureManager<GameTextures, Utils::EnumClassHash>;
}  // namespace GUI

#endif
```

jun 26, 18 17:16 **GameSoundEffects.h** Page 1/1

```
//
// Created by rodrigo on 4/06/18.
//

#ifndef INC_4_WORMS_GAMESOUNDEFFECTS_H
#define INC_4_WORMS_GAMESOUNDEFFECTS_H

#include "SoundEffectManager.h"
#include "utils.h"

namespace GUI {
/** Different kinds of sound effects. */
enum class GameSoundEffects {
    WalkCompress,
    Explosion,
    WormLanding,
    WormDrowning,
    Splash,
    WormJump,
    WormBackFlip,
    WormHit,
    WormDie,
    Holy,
    AirStrike,
    Teleport,
    Shot,
    Banana
};

/** Specialized SoundEffectManager class. */
using GameSoundEffectManager = SoundEffectManager<GameSoundEffects, Utils::EnumClassHash>;
}  // namespace GUI

#endif  // INC_4_WORMS_GAMESOUNDEFFECTS_H
```

jun 26, 18 17:16 **GameEndWindow.h** Page 1/1

```
//
// Created by rodrigo on 26/06/18.
//

#ifndef INC_4_WORMS_GAMEENDWINDOW_H
#define INC_4_WORMS_GAMEENDWINDOW_H


#include <vector>

#include "Window.h"
#include "Font.h"
#include "GameStateMsg.h"
#include "GameWindow.h"
#include "Button.h"


namespace GUI {
    class GameEndWindow : public GameWindow {
    public:
        explicit GameEndWindow(GUI::Window &window, GUI::Font &font, GUI::Camera &cam, bool youWin);

        void start() override;
        void render() override;
        void handleKeyDown(SDL_Keycode key) override;
        void appendCharacter(char text[32]) override;
        void buttonPressed(ScreenPosition sp) override;

    private:
        std::vector<Button> buttons;
        int textSize{50};
        std::string gameEndResultMsg;
    };
}


#endif //INC_4_WORMS_GAMEENDWINDOW_H
```

jun 26, 18 17:16 **GameEndWindow.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 26/06/18.
//

#include "GameEndWindow.h"

GUI::GameEndWindow::GameEndWindow(GUI::Window &window, GUI::Font &font, GUI::Camera &cam, bool youWin) :
        GameWindow(window, font, cam) {
    this→gameEndResultMsg = youWin ? "You Win!" : "You Lose!";
}

void GUI::GameEndWindow::start() {
    while (¬this→quit) {
        SDL_Event e;
        while (SDL_PollEvent(&e) ≠ 0) {
            switch (e.type) {
                case SDL_QUIT: {
                    this→quit = true;
//                    throw;
                    break;
                }
                default: {
                    break;
                }
            }
        }

        this→render();
    }
}

void GUI::GameEndWindow::render() {
    this→window.clear(SDL_Color{0xFF, 0xFF, 0xFF});

    SDL_Color black{0, 0, 0};

    Text gameResult{this→font};
    int x = this→window.getWidth() / 2;
    int y = this→window.getHeight() / 2;
    gameResult.set(this→gameEndResultMsg, black, 50);
    gameResult.renderFixed(ScreenPosition{x, y}, this→cam);

    this→window.render();
}

void GUI::GameEndWindow::buttonPressed(GUI::ScreenPosition sp) {
}

void GUI::GameEndWindow::appendCharacter(char *text) {
}

void GUI::GameEndWindow::handleKeyDown(SDL_Keycode key) {
}
```

jun 26, 18 17:16 **GameBackgroundMusic.h** Page 1/1

```cpp
//
// Created by rodrigo on 25/06/18.
//

#ifndef INC_4_WORMS_GAMEBACKGROUNDMUSIC_H
#define INC_4_WORMS_GAMEBACKGROUNDMUSIC_H

#include "BackgroundMusicManager.h"
#include "utils.h"

namespace GUI {
/** Different kinds of background music. */
    enum class GameBackgroundMusic {
        Original,
        MurderTrain
    };

/** Specialized BackgroundMusicManager class. */
    using GameBackgroundMusicManager = BackgroundMusicManager<GameBackgroundMusic, Utils::EnumClassHash>;
}  // namespace GUI

#endif //INC_4_WORMS_GAMEBACKGROUNDMUSIC_H
```

jun 26, 18 17:16 **CreateGameWindow.h** Page 1/1

```
//
// Created by rodrigo on 23/06/18.
//

#ifndef INC_4_WORMS_CREATEGAMEWINDOW_H
#define INC_4_WORMS_CREATEGAMEWINDOW_H


#include <vector>

#include "Window.h"
#include "Font.h"
#include "GameStateMsg.h"
#include "GameWindow.h"
#include "Button.h"

#define SELECT_LEVEL_MSG "Select"
#define LEVEL_MSG "Level"
#define PLAYERS_MSG "Players"
#define NEXT_LEVEL_MSG "Next"
#define PREVIOUS_LEVEL_MSG "Previous"

namespace GUI {
    class CreateGameWindow : public GameWindow {
    public:
        std::vector<IO::LevelInfo> &levelsInfo;

        explicit CreateGameWindow(GUI::Window &window, GUI::Font &font, GUI::Cam
era &cam,
                                  std::vector<IO::LevelInfo> &levelsInfo);

        void start() override;
        void render() override;
        void handleKeyDown(SDL_Keycode key) override;
        void appendCharacter(char text[32]) override;
        void buttonPressed(ScreenPosition sp) override;

    private:
        std::vector<Button> buttons;
        int levelInfoSize{30};
    };
}


#endif //INC_4_WORMS_CREATEGAMEWINDOW_H
```

jun 26, 18 17:16 **CreateGameWindow.cpp** Page 1/2

```
//
// Created by rodrigo on 23/06/18.
//

#include <iostream>
#include "GameStateMsg.h"
#include "CreateGameWindow.h"

GUI::CreateGameWindow::CreateGameWindow(GUI::Window &window, GUI::Font &font, GU
I::Camera &cam,
                                        std::vector<IO::LevelInfo> &levelsInfo)
:
        GameWindow(window, font, cam),
        levelsInfo(levelsInfo) {
    int height = this→levelInfoSize * 3 / 2;
    std::string msg(SELECT_LEVEL_MSG);
    int x = this→window.getWidth() / 2;
    int y = this→window.getHeight() * 3 / 4;
    this→buttons.emplace_back(msg, this→font, SDL_Color{0xFF, 0xFF, 0xFF}, thi
s→levelInfoSize);
    this→buttons.back().position = ScreenPosition{x, y};
    this→buttons.back().height = height;
    this→buttons.back().width = this→buttons.back().msg.size() * 9 + 20;
    msg = NEXT_LEVEL_MSG;
    x = this→window.getWidth() * 3 / 4;
    y = this→window.getHeight() / 2;
    this→buttons.emplace_back(msg, this→font, SDL_Color{0xFF, 0xFF, 0xFF}, thi
s→levelInfoSize);
    this→buttons.back().position = ScreenPosition{x, y};
    this→buttons.back().height = height;
    this→buttons.back().width = this→buttons.back().msg.size() * 9 + 20;
    msg = PREVIOUS_LEVEL_MSG;
    x = this→window.getWidth() / 4;
    y = this→window.getHeight() / 2;
    this→buttons.emplace_back(msg, this→font, SDL_Color{0xFF, 0xFF, 0xFF}, thi
s→levelInfoSize);
    this→buttons.back().position = ScreenPosition{x, y};
    this→buttons.back().height = height;
    this→buttons.back().width = this→buttons.back().msg.size() * 9 + 20;
}

void GUI::CreateGameWindow::start() {

}

void GUI::CreateGameWindow::render() {
    this→window.clear(SDL_Color{0xFF, 0xFF, 0xFF});

    SDL_Color white{0xFF, 0xFF, 0xFF};
    SDL_Color black{0, 0, 0};

    Text levelName{this→font};
    Text levelPlayersQuantity{this→font};
    levelName.setBackground(black);
    levelPlayersQuantity.setBackground(black);
    int x = this→window.getWidth() * 4 / 10;
    int y = this→window.getHeight() * 3 / 7;
    levelName.set(LEVEL_MSG, white, 50);
    levelName.renderFixed(ScreenPosition{x, y − 50}, this→cam);
    x = this→window.getWidth() * 6 / 10;
    levelName.set(PLAYERS_MSG, white, 50);
    levelName.renderFixed(ScreenPosition{x, y − 50}, this→cam);
    levelName.setBackground(white);
    levelPlayersQuantity.setBackground(white);
    x = this→window.getWidth() * 4 / 10;
    y = this→window.getHeight() / 2;
```

jun 26, 18 17:16 **CreateGameWindow.cpp** Page 2/2

```
62      levelName.set(this→levelsInfo[this→buttonSelected].name, black, this→level
    InfoSize);
63      levelName.renderFixed(ScreenPosition{x, y}, this→cam);
64      x = this→window.getWidth() * 6 / 10;
65      levelName.set(std::to_string(this→levelsInfo[this→buttonSelected].playersQ
    uantity), black, this→levelInfoSize);
66      levelName.renderFixed(ScreenPosition{x, y}, this→cam);
67
68      for (auto &button : this→buttons) {
69          button.render(this→cam);
70      }
71
72      this→window.render();
73  }
74
75  void GUI::CreateGameWindow::buttonPressed(GUI::ScreenPosition sp) {
76      if (this→buttons[0].inside(sp)) {
77          this→notify(*this, Event::LevelSelected);
78      }
79
80      if (this→buttons[1].inside(sp)) {
81          this→buttonSelected = (this→buttonSelected + 1) % this→levelsInfo.size
    ();
82      }
83
84      if (this→buttons[2].inside(sp)) {
85          this→buttonSelected = (this→buttonSelected ≡ 0) ? this→levelsInfo.size
    () − 1 : this→buttonSelected − 1;
86      }
87  }
88
89  void GUI::CreateGameWindow::appendCharacter(char *text) {
90
91  }
92
93  void GUI::CreateGameWindow::handleKeyDown(SDL_Keycode key) {
94
95  }
```

jun 26, 18 17:16 **ConnectionWindow.h** Page 1/1

```
1   //
2   // Created by rodrigo on 24/06/18.
3   //
4
5   #ifndef INC_4_WORMS_CONNECTIONWINDOW_H
6   #define INC_4_WORMS_CONNECTIONWINDOW_H
7
8
9   #include <vector>
10
11  #include "Window.h"
12  #include "Font.h"
13  #include "GameStateMsg.h"
14  #include "GameWindow.h"
15  #include "Button.h"
16
17  #define CONNECT_MSG "Connect"
18  #define IP_FOCUS 0
19  #define PORT_FOCUS 1
20
21  namespace GUI {
22      struct ConnectionInfo {
23          const char *ip;
24          const char *port;
25      };
26      class ConnectionWindow : public GameWindow {
27      public:
28          uint8_t playersConnected{0};
29
30          explicit ConnectionWindow(GUI::Window &window, GUI::Font &font, GUI::Cam
    era &cam);
31
32          void start() override;
33          void render() override;
34          void handleKeyDown(SDL_Keycode key) override;
35          void appendCharacter(char text[32]) override;
36          void buttonPressed(ScreenPosition sp) override;
37
38          ConnectionInfo getConnectionInfo();
39
40      private:
41          std::vector<Button> buttons;
42          int textSize{50};
43      };
44  }
45
46
47  #endif //INC_4_WORMS_CONNECTIONWINDOW_H
```

jun 26, 18 17:16 **ConnectionWindow.cpp** Page 1/2

```
1  //
2  // Created by rodrigo on 24/06/18.
3  //
4
5  #include "ConnectionWindow.h"
6
7  GUI::ConnectionWindow::ConnectionWindow(GUI::Window &window, GUI::Font &font, GU
   I::Camera &cam) :
8          GameWindow(window, font, cam) {
9      std::string msg(CONNECT_MSG);
10     this→buttons.emplace_back(msg, this→font, SDL_Color{0xFF, 0xFF, 0xFF}, thi
   s→textSize);
11     int x = this→window.getWidth() / 2;
12     int y = this→window.getHeight() * 3 / 4;
13     this→buttons.back().position = ScreenPosition{x, y};
14     this→buttons.back().height = this→textSize * 3 / 2;
15     this→buttons.back().width = this→buttons.back().msg.size() * 20 + 20;
16
17     x = this→window.getWidth() * 6 / 10;
18     y = this→window.getHeight() * 2 / 7;
19     int textFieldHeight = this→textSize * 3 / 2;
20     int textFieldWidth = 400;
21     std::string emptyMsg(" ");
22     this→textFields.emplace_back(emptyMsg, ScreenPosition{x, y}, textFieldHeigh
   t, textFieldWidth, this→font);
23     y = this→window.getHeight() * 4 / 7;
24     emptyMsg = " ";
25     this→textFields.emplace_back(emptyMsg, ScreenPosition{x, y}, textFieldHeigh
   t, textFieldWidth, this→font);
26 }
27
28 void GUI::ConnectionWindow::start() {
29
30 }
31
32 void GUI::ConnectionWindow::render() {
33     this→window.clear(SDL_Color{0xFF, 0xFF, 0xFF});
34
35     SDL_Color black{0, 0, 0};
36
37     Text ip{this→font};
38     Text port{this→font};
39     port.setBackground(black);
40     int x = this→window.getWidth() * 3 / 10;
41     int y = this→window.getHeight() * 2 / 7;
42     ip.set("IP:", black, 50);
43     ip.renderFixed(ScreenPosition{x, y}, this→cam);
44     y = this→window.getHeight() * 4 / 7;
45     ip.set("Server port:", black, 50);
46     ip.renderFixed(ScreenPosition{x, y}, this→cam);
47
48     for (auto &button : this→buttons) {
49         button.render(this→cam);
50     }
51
52     for (auto &textField : this→textFields) {
53         textField.render(this→cam);
54     }
55
56     this→window.render();
57 }
58
59 void GUI::ConnectionWindow::buttonPressed(GUI::ScreenPosition sp) {
60     for (auto &textField : this→textFields) {
61         textField.selected(sp);
62     }
```

jun 26, 18 17:16 **ConnectionWindow.cpp** Page 2/2

```
63
64     if (this→buttons[0].inside(sp)) {
65         this→notify(*this, Event::ConnectionToServer);
66     }
67 }
68
69 void GUI::ConnectionWindow::appendCharacter(char *text) {
70     for (auto &textField : this→textFields) {
71         if (textField.focus) {
72             textField.appendCharacter(text);
73         }
74     }
75 }
76
77 void GUI::ConnectionWindow::handleKeyDown(SDL_Keycode key) {
78     switch (key) {
79         case SDLK_BACKSPACE: {
80             for (auto &textField : this→textFields) {
81                 if (textField.focus) {
82                     textField.backSpace();
83                 }
84             }
85             break;
86         }
87     }
88 }
89
90 GUI::ConnectionInfo GUI::ConnectionWindow::getConnectionInfo() {
91     return ConnectionInfo{this→textFields[0].inputText.msg.c_str(),
92                           this→textFields[1].inputText.msg.c_str()};
93 }
```

jun 26, 18 17:16 **CommunicationProtocol.h** Page 1/1

```
1  //
2  // Created by rodrigo on 20/06/18.
3  //
4
5  #ifndef INC_4_WORMS_COMMUNICATIONPROTOCOL_H
6  #define INC_4_WORMS_COMMUNICATIONPROTOCOL_H
7
8
9  #include "ClientSocket.h"
10 #include <Protocol.h>
11 #include <Stream.h>
12 #include "Thread.h"
13
14 namespace IO {
15     class CommunicationProtocol : public Thread {
16     public:
17         std::vector<LevelInfo> levelsInfo;
18         uint8_t levelToCreate{0};
19         std::vector<GameInfo> gamesInfo;
20         uint8_t gameToJoin{0};
21         uint8_t levelOfGameToJoin{0};
22         std::string levelPath;
23         std::vector<std::string> backgroundPath;
24
25
26         explicit CommunicationProtocol(ClientSocket &socket, IO::Stream<IO::Clie
   ntGUIMsg> *clientStream,
27                                        IO::Stream<IO::ServerResponse> *output);
28
29         void run() override;
30         void stop() override;
31
32         ClientSocket getSocket();
33
34     private:
35         Protocol<ClientSocket> protocol;
36         unsigned char command{0};
37         std::uint8_t playersQuantity{0};
38         IO::Stream<IO::ClientGUIMsg> *clientStream;
39         IO::Stream<IO::ServerResponse> *output;
40         bool quit{false};
41
42         void startCreateGame();
43         void startJoinGame();
44         void joinGame();
45         void waitGameStart(uint8_t playersQuantity);
46
47         void handleClientInput(ClientGUIMsg &msg);
48
49         void createGame();
50
51         void getLevelFiles();
52     };
53 }
54
55
56 #endif //INC_4_WORMS_COMMUNICATIONPROTOCOL_H
```

jun 26, 18 17:16 **CommunicationProtocol.cpp** Page 1/3

```
1  //
2  // Created by rodrigo on 20/06/18.
3  //
4
5  #include <fstream>
6  #include <iostream>
7
8  #include "CommunicationProtocol.h"
9  #include "GameStateMsg.h"
10 #include "Stream.h"
11
12 IO::CommunicationProtocol::CommunicationProtocol(ClientSocket &socket, IO::Strea
   m<IO::ClientGUIMsg> *clientStream,
13                                                  IO::Stream<IO::ServerResponse>
   *output)
14         :
15         protocol(socket),
16         clientStream(clientStream),
17         output(output) {
18 }
19
20 void IO::CommunicationProtocol::run() {
21     try {
22         while (¬this→quit) {
23             IO::ClientGUIMsg msg;
24             if (clientStream→pop(msg)) {
25                 this→handleClientInput(msg);
26             }
27         }
28     } catch (std::exception &e){
29         if (¬this→quit){
30             std::cerr << "In CommunicationProtocol::run()" << std::endl;
31             std::cerr << e.what() << std::endl;
32             IO::ServerResponse sr{IO::ServerResponseAction::serverClosed};
33             *this→output << sr;
34         }
35     } catch (...){
36         std::cerr << "Unknown Error in CommunicationProtocol::run()" << std::endl;
37     }
38 }
39
40 void IO::CommunicationProtocol::startCreateGame(){
41     this→command = COMMAND_GET_LEVELS;
42     this→protocol << this→command;
43     this→protocol >> this→levelsInfo;
44     *this→output << IO::ServerResponse{IO::ServerResponseAction::levelsInfo};
45 }
46
47 void IO::CommunicationProtocol::createGame() {
48     this→command = COMMAND_CREATE_GAME;
49     this→protocol << this→command;
50     this→protocol << this→levelToCreate;
51     this→getLevelFiles();
52     this→waitGameStart(this→levelsInfo[this→levelToCreate].playersQuantity);
53 }
54
55 void IO::CommunicationProtocol::startJoinGame(){
56     this→command = COMMAND_GET_GAMES;
57     this→protocol << this→command;
58     this→protocol >> this→gamesInfo;
59
60     IO::ServerResponse sr;
61     sr.action = IO::ServerResponseAction::gamesInfo;
62     *this→output << sr;
63 }
64
```

```
65  void IO::CommunicationProtocol::joinGame() {
66      this→command = COMMAND_JOIN_GAME;
67      this→protocol << this→command;
68      this→protocol << this→gameToJoin;
69      this→protocol << this→levelOfGameToJoin;
70      this→getLevelFiles();
71      this→waitGameStart(this→gamesInfo[this→gameToJoin].numTotalPlayers);
72  }
73
74  ClientSocket IO::CommunicationProtocol::getSocket() {
75      return std::move(this→protocol.getSocket());
76  }
77
78  void IO::CommunicationProtocol::waitGameStart(uint8_t playersQuantity) {
79      while (this→playersQuantity < playersQuantity) {
80          this→protocol >> this→playersQuantity;
81          *this→output << IO::ServerResponse{IO::ServerResponseAction::playerConn
    ected};
82      }
83      IO::ServerResponse sr{};
84      sr.action = IO::ServerResponseAction::startGame;
85      *this→output << sr;
86  }
87
88  void IO::CommunicationProtocol::stop() {
89      this→quit = true;
90      this→protocol.stopCommunication();
91  }
92
93  void IO::CommunicationProtocol::handleClientInput(IO::ClientGUIMsg &msg) {
94      switch (msg.input) {
95          case IO::ClientGUIInput::startCreateGame: {
96              this→startCreateGame();
97              break;
98          }
99          case IO::ClientGUIInput::levelSelected: {
100             this→createGame();
101             break;
102         }
103         case IO::ClientGUIInput::startJoinGame: {
104             this→startJoinGame();
105             break;
106         }
107         case IO::ClientGUIInput::joinGame: {
108             this→joinGame();
109             break;
110         }
111         case IO::ClientGUIInput::quit: {
112             this→quit = true;
113             break;
114         }
115         default: {
116             break;
117         }
118     }
119 }
120
121 void IO::CommunicationProtocol::getLevelFiles() {
122     this→protocol >> this→levelPath;
123     std::ofstream levelFile(this→levelPath, std::ofstream::binary);
124     this→protocol >> levelFile;
125
126     this→protocol >> this→backgroundPath;
127     for (auto &background : this→backgroundPath) {
128         std::ofstream backgroundFile(background, std::ofstream::binary);
129         this→protocol >> backgroundFile;
```



```
130     }
131 }
132
```

jun 26, 18 17:16 **ClientSocket.h** Page 1/1

```
1  /*
2   * Created by Federico Manuel Gomez Peter
3   * Date: 02/05/2018.
4   */
5
6  #ifndef __ClientSocket_H__
7  #define __ClientSocket_H__
8
9  #include <string>
10
11 #include "CommunicationSocket.h"
12
13 /**
14  * Socket que tiene la capacidad de realizar una conexion con el servidor,
15  * partiendo del dato del host y el port a donde conectarse
16  */
17 class ClientSocket : public CommunicationSocket {
18    public:
19     ClientSocket(const char *hostName, const char *port);
20 };
21
22 #endif  //__ClientSocket_H__
```

jun 26, 18 17:16 **ClientSocket.cpp** Page 1/1

```
1  /*
2   * Created by Federico Manuel Gomez Peter
3   * Date: 02/05/2018.
4   */
5
6  #include <netdb.h>
7  #include <unistd.h>
8  #include <cstring>
9
10 #include "ClientSocket.h"
11 #include "ErrorMessages.h"
12 #include "Exception.h"
13
14 ClientSocket::ClientSocket(const char *hostName, const char *port) {
15     int status;
16     bool is_connected = false;
17
18     struct addrinfo hints = {AI_PASSIVE, AF_INET, SOCK_STREAM, 0, 0, nullptr, nu
   llptr, nullptr};
19     struct addrinfo *result, *ptr;
20
21     status = getaddrinfo(hostName, port, &hints, &result);
22     if (status ≠ 0) {
23         throw Exception(ERR_MSG_SOCKET_INVALID_HOST_OR_PORT, hostName, port, str
   error(errno));
24     }
25
26     for (ptr = result; ptr ≠ nullptr ∧ ¬is_connected; ptr = ptr→ai_next) {
27         this→fd = socket(ptr→ai_family, ptr→ai_socktype, ptr→ai_protocol);
28         /*
29         * si la creaciÃ³n del socket falla, no debo hacer nada mas
30         * en el ciclo (ya que no se abrio ningun fd)
31         */
32         if (this→fd ≡ −1) {
33             continue;
34         }
35
36         status = ::connect(this→fd, ptr→ai_addr, ptr→ai_addrlen);
37         if (status ≡ −1) {
38             ::close(this→fd);
39             this→fd = −1;
40         } else {
41             is_connected = true;
42         }
43     }
44
45     freeaddrinfo(result);
46     if (¬is_connected) {
47         throw Exception(ERR_MSG_CONNECTION_COULD_NOT_BE_STABLISHED, hostName, po
   rt);
48     }
49 }
```

jun 26, 18 17:16 **Button.h** Page 1/1

```
1  //
2  // Created by rodrigo on 20/06/18.
3  //
4
5  #ifndef INC_4_WORMS_BUTTON_H
6  #define INC_4_WORMS_BUTTON_H
7
8
9  #include <Camera.h>
10 #include <Text.h>
11
12 namespace GUI {
13     class Button {
14     public:
15         GUI::ScreenPosition position{0, 0};
16         int height{0};
17         int width{0};
18         std::string msg;
19         SDL_Color textColor{0, 0, 0};
20         int textSize{10};
21
22         Button(ScreenPosition sp, int height, int width, const std::string &msg,
    Font &font);
23         Button(const std::string &msg, GUI::Font &font, SDL_Color textColor, int
    textSize);
24         Button(const std::string &msg, Font &font);
25         Button(ScreenPosition sp, int height, int width, Font &font);
26
27         bool inside(ScreenPosition sp);
28         void render(GUI::Camera &cam);
29
30         void setBackground(SDL_Color color);
31
32     private:
33         Text text;
34     };
35 }
36
37 #endif //INC_4_WORMS_BUTTON_H
```

jun 26, 18 17:16 **Button.cpp** Page 1/2

```
1  //
2  // Created by rodrigo on 20/06/18.
3  //
4
5  #include <Font.h>
6  #include "Button.h"
7
8  GUI::Button::Button(ScreenPosition sp, int height, int width, const std::string
   &msg, Font &font) :
9          position(sp),
10         height(height),
11         width(width),
12         msg(msg),
13         textColor(SDL_Color{0xFF, 0xFF, 0xFF}),
14         textSize(40),
15         text(font) {
16     this→text.set(this→msg, SDL_Color{0xFF, 0xFF, 0xFF}, 40);
17 }
18
19 GUI::Button::Button(const std::string &msg, GUI::Font &font, SDL_Color textColor
   , int textSize) :
20         msg(msg),
21         textColor(textColor),
22         textSize(textSize),
23         text(font) {
24     this→text.set(this→msg, textColor, textSize);
25 }
26
27 GUI::Button::Button(const std::string &msg, GUI::Font &font) :
28         msg(msg),
29         text(font) {
30 }
31
32 GUI::Button::Button(GUI::ScreenPosition sp, int height, int width, GUI::Font &fo
   nt) :
33         position(sp),
34         height(height),
35         width(width),
36         text(font) {
37
38 }
39
40 void GUI::Button::render(GUI::Camera &cam) {
41     SDL_Rect fillRect = {this→position.x − this→width / 2, this→position.y + t
   his→height / 2,
42                          this→width / (int) cam.getScale(), this→height / (int
   ) cam.getScale()};
43     cam.drawLocal(ScreenPosition{this→position.x, this→position.y}, fillRect,
   SDL_Color{0, 0, 0});
44     this→text.set(this→msg, this→textColor, this→textSize);
45     this→text.renderFixed(this→position, cam);
46 }
47
48 bool GUI::Button::inside(GUI::ScreenPosition sp) {
49     bool inside = true;
50
51     if(sp.x < this→position.x − this→width / 2) {
52         //Mouse is left of the button
53         inside = false;
54     } else if(sp.x > this→position.x + this→width / 2) {
55         //Mouse is right of the button
56         inside = false;
57     } else if(sp.y < this→position.y − this→height / 2) {
58         //Mouse below the button
59         inside = false;
60     } else if(sp.y > this→position.y + this→width / 2) {
```

jun 26, 18 17:16 **Button.cpp** Page 2/2

```
61          //Mouse above the button
62          inside = false;
63      }
64      return inside;
65  }
66
67  void GUI::Button::setBackground(SDL_Color color) {
68      this→text.setBackground(color);
69  }
```

jun 26, 18 17:16 **BackgroundMusicPlayer.h** Page 1/1

```
1   //
2   // Created by rodrigo on 25/06/18.
3   //
4
5   #ifndef INC_4_WORMS_BACKGROUNDMUSICPLAYER_H
6   #define INC_4_WORMS_BACKGROUNDMUSICPLAYER_H
7
8
9   #include <SDL2/SDL.h>
10
11  #include "BackgroundMusic.h"
12
13  namespace GUI {
14      class BackgroundMusicPlayer {
15      public:
16          bool loop{false};
17
18          explicit BackgroundMusicPlayer(const GUI::BackgroundMusic &backgroundMus
    ic);
19          ~BackgroundMusicPlayer();
20          void play();
21
22      private:
23          const BackgroundMusic *backgroundMusic;
24      };
25  }
26
27
28  #endif //INC_4_WORMS_BACKGROUNDMUSICPLAYER_H
```

jun 26, 18 17:16 **BackgroundMusicPlayer.cpp** Page 1/1

```cpp
//
// Created by rodrigo on 25/06/18.
//

#include "BackgroundMusicPlayer.h"

GUI::BackgroundMusicPlayer::BackgroundMusicPlayer(const GUI::BackgroundMusic &backgroundMusic)
        : backgroundMusic(&backgroundMusic) {}

GUI::BackgroundMusicPlayer::~BackgroundMusicPlayer() {}

void GUI::BackgroundMusicPlayer::play() {
    this→backgroundMusic→play();
}
```

jun 26, 18 17:16 **BackgroundMusicManager.h** Page 1/1

```cpp
//
// Created by rodrigo on 25/06/18.
//

#ifndef INC_4_WORMS_BACKGROUNDMUSICMANAGER_H
#define INC_4_WORMS_BACKGROUNDMUSICMANAGER_H

#include <SDL2/SDL.h>
#include <functional>
#include <string>
#include <unordered_map>
#include "BackgroundMusic.h"

namespace GUI {
    template <typename ID, typename HASH = std::hash<ID>>
    class BackgroundMusicManager {
    public:
        BackgroundMusicManager();
        ~BackgroundMusicManager();
        BackgroundMusicManager& operator=(BackgroundMusicManager& other) = delete;

        void load(ID id, const std::string& file_name);
        const BackgroundMusic& get(ID id) const;

    private:
        std::unordered_map<ID, BackgroundMusic, HASH> cache;
    };
}  // namespace GUI

template <typename ID, typename HASH>
GUI::BackgroundMusicManager<ID, HASH>::BackgroundMusicManager() {}

template <typename ID, typename HASH>
GUI::BackgroundMusicManager<ID, HASH>::~BackgroundMusicManager() {}

/**

 * @brief Loads a background music file.
 *
 * @param file_name The image file name.
 */
template <typename ID, typename HASH>
void GUI::BackgroundMusicManager<ID, HASH>::load(ID id, const std::string& file_name) {
    GUI::BackgroundMusic backgroundMusic{file_name};
    this→cache.insert(std::make_pair(id, std::move(backgroundMusic)));
}

/**
 * @brief Gets a background music.
 *
 * @param file_name Name of the background music.
 */
template <typename ID, typename HASH>
const GUI::BackgroundMusic& GUI::BackgroundMusicManager<ID, HASH>::get(ID id) const {
    return this→cache.at(id);
}

#endif //INC_4_WORMS_BACKGROUNDMUSICMANAGER_H
```

jun 26, 18 17:16 **BackgroundMusic.h** Page 1/1

```
1  //
2  // Created by rodrigo on 25/06/18.
3  //
4
5  #ifndef INC_4_WORMS_BACKGROUNDMUSIC_H
6  #define INC_4_WORMS_BACKGROUNDMUSIC_H
7
8
9  #include <SDL2/SDL.h>
10 #include <SDL2/SDL_mixer.h>
11 #include <string>
12
13 namespace GUI {
14     class BackgroundMusic {
15     public:
16         BackgroundMusic(const std::string &filename);
17         BackgroundMusic(BackgroundMusic ∧other);
18         ~BackgroundMusic();
19         Mix_Music *getMusic() const;
20         void play() const;
21
22     private:
23         Mix_Music *backgroundMusic{nullptr};
24     };
25 }
26
27
28 #endif //INC_4_WORMS_BACKGROUNDMUSIC_H
```

jun 26, 18 17:16 **BackgroundMusic.cpp** Page 1/1

```
1  //
2  // Created by rodrigo on 25/06/18.
3  //
4
5  #include "BackgroundMusic.h"
6  #include "Exception.h"
7
8  GUI::BackgroundMusic::BackgroundMusic(const std::string &filename) {
9      this→backgroundMusic = Mix_LoadMUS(filename.c_str());
10     if (¬this→backgroundMusic) {
11         throw Exception{"Error loading %s: %s", filename.c_str(), Mix_GetError()};
12     }
13 }
14
15 GUI::BackgroundMusic::~BackgroundMusic() {
16     if (this→backgroundMusic ≠ nullptr) {
17         Mix_FreeMusic(this→backgroundMusic);
18     }
19 }
20
21 Mix_Music *GUI::BackgroundMusic::getMusic() const {
22     return this→backgroundMusic;
23 }
24
25 GUI::BackgroundMusic::BackgroundMusic(GUI::BackgroundMusic ∧other) {
26     std::swap(this→backgroundMusic, other.backgroundMusic);
27 }
28
29 void GUI::BackgroundMusic::play() const {
30     Mix_PlayMusic(this→backgroundMusic, −1);
31 }
```

jun 26, 18 17:16 **Armory.h** Page 1/1

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 10/06/18
 */

#ifndef __ARMORY_H__
#define __ARMORY_H__

#define BUTTON_ROOT_STR "F"

#include <vector>

#include <Animation.h>
#include <Font.h>
#include <GameStateMsg.h>
#include <Text.h>
#include "GameTextures.h"

namespace GUI {
class Armory {
   public:
    Armory(const GameTextureManager &textureManager, Camera &cam, Font &font);
    ~Armory() = default;
    void loadWeapons();
    void render();
    void update(IO::GameStateMsg &msg);

   private:
    const GameTextureManager &manager;
    Camera &camera;
    std::vector<const Texture *> weaponIcons;
    const Font &font;
    Text weaponButton;
    std::vector<std::int16_t> ammunition;
};
}  // namespace GUI

#endif  //__ARMORY_H__
```

jun 26, 18 17:16 **Armory.cpp** Page 1/2

```
/*
 *  Created by Federico Manuel Gomez Peter.
 *  date: 10/06/18
 */

#include <sstream>

#include "Armory.h"

GUI::Armory::Armory(const GUI::GameTextureManager &textureManager, GUI::Camera &
cam,
                    GUI::Font &font)
    : manager(textureManager),
      camera(cam),
      font(font),
      weaponButton(font),
      ammunition(WEAPONS_QUANTITY, 0) {}

void GUI::Armory::render() {
    const Texture *temp = this->weaponIcons.back();
    ScreenPosition ammoPos{-temp->getWidth() / 2, 10};
    ScreenPosition iconPos{-temp->getWidth() / 2, 20 + temp->getHeight() / 2};
    ScreenPosition textPos{-temp->getWidth() / 2, 20 + temp->getHeight() * 3 / 2
};
    int i = 1;
    for (auto &weapon : this->weaponIcons) {
        ammoPos.x += weapon->getWidth();
        iconPos.x += weapon->getWidth();
        textPos.x += weapon->getWidth();

        std::int16_t weaponAmmo = this->ammunition[i - 1];
        std::ostringstream button;
        button << BUTTON_ROOT_STR << i++;

        if (weaponAmmo == -1) {
            weaponButton.set(std::string("inf"), SDL_Color{0, 0, 0}, 20);
            weaponButton.renderFixed(ammoPos, this->camera);
        } else {
            weaponButton.set(std::to_string(weaponAmmo), SDL_Color{0, 0, 0}, 20)
;
            weaponButton.renderFixed(ammoPos, this->camera);
        }

        weaponButton.set(button.str(), SDL_Color{0, 0, 0}, 25);
        weaponButton.renderFixed(textPos, this->camera);
        this->camera.drawLocal(*weapon, iconPos);
    }
}

void GUI::Armory::loadWeapons() {
    this->weaponIcons.emplace_back(&this->manager.get(GUI::GameTextures::Bazooka
Icon));
    this->weaponIcons.emplace_back(&this->manager.get(GUI::GameTextures::Grenade
Icon));
    this->weaponIcons.emplace_back(&this->manager.get(GUI::GameTextures::Cluster
Icon));
    this->weaponIcons.emplace_back(&this->manager.get(GUI::GameTextures::MortarI
con));
    this->weaponIcons.emplace_back(&this->manager.get(GUI::GameTextures::BananaI
con));
    this->weaponIcons.emplace_back(&this->manager.get(GUI::GameTextures::HolyIco
n));
    this->weaponIcons.emplace_back(&this->manager.get(GUI::GameTextures::AirIcon
));
    this->weaponIcons.emplace_back(&this->manager.get(GUI::GameTextures::Dynamit
eIcon));
```

jun 26, 18 17:16 **Armory.cpp** Page 2/2

```
56      this→weaponIcons.emplace_back(&this→manager.get(GUI::GameTextures::Basebal
    lBatIcon));
57      this→weaponIcons.emplace_back(&this→manager.get(GUI::GameTextures::Telepor
    tIcon));
58  }
59
60  void GUI::Armory::update(IO::GameStateMsg &msg) {
61      for (int i = 0; i < WEAPONS_QUANTITY; i++) {
62          this→ammunition[i] = msg.weaponAmmunition[i];
63      }
64  }
```

jun 26, 18 17:16 **Table of Content** Page 1/4

jun 26, 18 17:16 **Table of Content** Page 2/4

jun 26, 18 17:16 **Table of Content** Page 3/4

jun 26, 18 17:16 **Table of Content** Page 4/4