```
//
// Created by marcos on 22/6/20.
//

#define CATCH_CONFIG_MAIN
#include "catch.hpp"

```

```
//
// Created by agustin on 22/6/20.
//

#ifndef ARGENTUM_MAPTESTS_H
#define ARGENTUM_MAPTESTS_H

class Map;

class MapTests {
private:
    static void _fillEmptyMap(Map& map, int iSize, int jSize, bool isCity = false);

public:
    static bool testAvailableMapHasAvailableTiles();
    static bool testCityMapHasCityTiles();
    static bool testMixedCityAndUnavailableTiles();
    static bool testAddedItemsToMap();
    static bool testAddedGoldToMap();
    static bool testAddedMultipleGoldsToMapWithList();
    static bool testAddedMultipleGoldsToMapWithoutList();
    static bool testAddedMultipleItemsListsToMap();
    static bool testGetTargetsOnEmptyMapReturnsEmptyList();
    static bool testGetTargetsOnMapWithPlayerReturnsListWithOneElement();
    static bool testGetTargetsOnMapWithDeadPlayerReturnsEmptyList();
    static bool testPositionWithPlayerIsOccupied();
    static bool testPositionWithMonsterIsOccupied();

    static bool testListOnEmptyTileReturnsEmptyList();
    static bool testListOnEmptyMapReturnsEmptyList();
    static bool testWithdrawOnEmptyTileGetsNoItem();
    static bool testWithdrawOnEmptyMapGetsNoItem();
    static bool testDepositExistentItemOnEmptyTileGetsNoItem();
    static bool testDepositExistentItemOnEmptyMapGetsNoItem();
    static bool testBuyItemFromEmptyTileGetsNoItem();
    static bool testBuyItemFromEmptyMapGetsNoItem();
    static bool testSellExistentItemToEmptyTileGetsNoItem();
    static bool testSellExistentItemToEmptyMapGetsNoItem();

    static bool testMoveEntity();
    static bool testRemoveEntityOnEmptyTileLeavesNoEntity();
};


#endif //ARGENTUM_MAPTESTS_H
```

```cpp
//
// Created by agustin on 22/6/20.
//

#include "MapTests.h"

#include <iostream>
#include <memory>
#include "../Map/Map.h"
#include "../Items/Miscellaneous/Gold.h"
#include "../Items/Miscellaneous/HealthPotion.h"
#include "../Items/Miscellaneous/ManaPotion.h"
#include "../Items/Defense/Head.h"
#include "../Items/Defense/Shield.h"
#include "../Items/Defense/Chest.h"
#include "../Items/Attack/Weapon.h"
#include "../Entities/Player.h"
#include "../Entities/Monster.h"
#include "../Game/Game.h"
#include "../Entities/Citizens/Storage.h"
#include "catch.hpp"
#include "fakeit.hpp"

using namespace fakeit;

////////////////////////PRIVATE////////////////////////

void MapTests::_fillEmptyMap(Map &map, int iSize, int jSize, bool isCity) {
    for (int i = 0; i < iSize; ++i) {
        map.tiles.emplace_back();
        for (int j = 0; j < jSize; ++j) {
            map.tiles[i].emplace_back(true, isCity, GameType::FloorType::GRASS0,
                                      GameType::Structure::NO_STRUCTURE,
                                      std::shared_ptr<Entity>(nullptr));
        }
    }
}

////////////////////////PUBLIC////////////////////////

bool MapTests::testAvailableMapHasAvailableTiles() {
    Map map;

    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    for (int i = 0; i < mapXSize; ++i) {
        for (int j = 0; j < mapYSize; ++j) {
            if (¬map.tiles[i][j].isAvailable()) {
                return false;
            }
        }
    }
    return true;
}

bool MapTests::testCityMapHasCityTiles() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize, true);
    for (int i = 0; i < mapXSize; ++i) {
        for (int j = 0; j < mapYSize; ++j) {
            if (¬map.tiles[i][j].isInCity()) {
                return false;
            }
```

```cpp
        }
    }
    return true;
}

bool MapTests::testMixedCityAndUnavailableTiles() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    bool isCity;
    bool isAvailable;
    bool isOccupable;
    for (int i = 0; i < mapXSize; ++i) {
        map.tiles.emplace_back();
        for (int j = 0; j < mapYSize; ++j) {
            isCity = (j % 2 ≡ 0);
            isOccupable = j % 3 ≡ 0;
            map.tiles[i].emplace_back(isOccupable, isCity, GameType::FloorType::
GRASS0,
                  GameType::Structure::NO_STRUCTURE,
                  nullptr);
        }
    }
    for (int i = 0; i < mapXSize; ++i) {
        for (int j = 0; j < mapYSize; ++j) {
            isCity = map.tiles[i][j].isInCity();
            isAvailable = map.tiles[i][j].isAvailable();
            if (j % 2 ≡ 0) {
                if (¬isCity) {
                    return false;
                }
            } else {
                if (isCity) {
                    return false;
                }
            }
            if (j % 3 ≡ 0) {
                if (¬isAvailable) {
                    return false;
                }
            } else {
                if (isAvailable) {
                    return false;
                }
            }
        }
    }
    return true;
}

bool MapTests::testAddedItemsToMap() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    std::list<std::shared_ptr<Item>> items;
    items.emplace_back(new Gold(5));
    items.emplace_back(new HealthPotion());
    items.emplace_back(new ManaPotion());
    items.emplace_back(new Head(GameType::Clothing::MAGIC_HAT));
    items.emplace_back(new Head(GameType::Clothing::NO_HELMET));
    items.emplace_back(new Shield(GameType::Clothing::IRON_SHIELD));
    items.emplace_back(new Chest(GameType::Clothing::PLATE_ARMOR));
    items.emplace_back(new Chest(GameType::Clothing::COMMON_CLOTHING));
    items.emplace_back(new Weapon(GameType::Weapon::LONGSWORD));
    items.emplace_back(new Weapon(GameType::Weapon::FIST));

    Configuration& config = Configuration::getInstance();

    std::vector<std::string> itemsNames = {config.configGetGoldName(), config.configPotionData(GameType::Potion::HEALTH_POTION).name,
                                           config.configPotionData(GameType::Potion::MANA_POTION).name,
                                           config.configClothingData(GameType::Clothing::MAGIC_HAT).name,
                                           config.configClothingData(GameType::Clothing::NO_HELMET).name,
                                           config.configClothingData(GameType::Clothing::IRON_SHIELD).name,
                                           config.configClothingData(GameType::Clothing::PLATE_ARMOR).name,
                                           config.configClothingData(GameType::Clothing::COMMON_CLOTHING).name,
                                           config.configWeaponData(GameType::Weapon::LONGSWORD).name,
                                           config.configWeaponData(GameType::Weapon::FIST).name};
    map.addItemsToTile(std::move(items), {1, 1});
    int i = 0;
    for (const auto & item: map.tiles[1][1].items) {
        if (itemsNames[i] ≠ item→getName()) {
            return false;
        }
        i++;
    }
    return true;
}

bool MapTests::testAddedGoldToMap() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    std::shared_ptr<Gold> gold(new Gold(1000));
    map.addItemsToTile(std::move(gold), {1, 1});
    return Configuration::getInstance().configGetGoldName() ≡ map.tiles[1][1].items.front()→getName();
}

bool MapTests::testAddedMultipleGoldsToMapWithList() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    std::list<std::shared_ptr<Item>> items;
    items.emplace_back(new Gold(1000));
    items.emplace_back(new Gold(1000));
    map.addItemsToTile(std::move(items), {1, 1});
    return (Configuration::getInstance().configGetGoldName() ≡ map.tiles[1][1].items.front()→getName()) ∧
           (Configuration::getInstance().configGetGoldName() ≡ map.tiles[1][1].items.back()→getName());
}

bool MapTests::testAddedMultipleGoldsToMapWithoutList() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    std::shared_ptr<Gold> gold(new Gold(1000));
    map.addItemsToTile(std::move(gold), {1, 1});
    gold.reset(new Gold(1000));
```

```
    map.addItemsToTile(std::move(gold), {1, 1});
    return (Configuration::getInstance().configGetGoldName() ≡ map.tiles[1][1].items.front()→getName()) ∧
           (Configuration::getInstance().configGetGoldName() ≡ map.tiles[1][1].items.back()→getName());
}

bool MapTests::testAddedMultipleItemsListsToMap() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    std::list<std::shared_ptr<Item>> items;
    items.emplace_back(new Gold(5));
    items.emplace_back(new HealthPotion());
    items.emplace_back(new ManaPotion());
    items.emplace_back(new Head(GameType::MAGIC_HAT));
    items.emplace_back(new Head(GameType::NO_HELMET));
    items.emplace_back(new Shield(GameType::IRON_SHIELD));
    items.emplace_back(new Chest(GameType::PLATE_ARMOR));
    items.emplace_back(new Chest(GameType::COMMON_CLOTHING));
    items.emplace_back(new Weapon(GameType::LONGSWORD));
    items.emplace_back(new Weapon(GameType::FIST));

    Configuration& config = Configuration::getInstance();
    std::vector<std::string> itemsNames = {config.configGetGoldName(), config.configPotionData(GameType::HEALTH_POTION).name,
                                           config.configPotionData(GameType::MANA_POTION).name,
                                           config.configClothingData(GameType::MAGIC_HAT).name,
                                           config.configClothingData(GameType::NO_HELMET).name,
                                           config.configClothingData(GameType::IRON_SHIELD).name,
                                           config.configClothingData(GameType::PLATE_ARMOR).name,
                                           config.configClothingData(GameType::COMMON_CLOTHING).name,
                                           config.configWeaponData(GameType::LONGSWORD).name,
                                           config.configWeaponData(GameType::FIST).name,
                                           config.configGetGoldName(), config.configPotionData(GameType::HEALTH_POTION).name,
                                           config.configPotionData(GameType::MANA_POTION).name,
                                           config.configClothingData(GameType::MAGIC_HAT).name,
                                           config.configClothingData(GameType::NO_HELMET).name,
                                           config.configClothingData(GameType::IRON_SHIELD).name,
                                           config.configClothingData(GameType::PLATE_ARMOR).name,
                                           config.configClothingData(GameType::COMMON_CLOTHING).name,
                                           config.configWeaponData(GameType::LONGSWORD).name,
                                           config.configWeaponData(GameType::FIST).name};

    map.addItemsToTile(std::move(items), {1, 1});

    items.clear();
    items.emplace_back(new Gold(5));
```

```cpp
    items.emplace_back(new HealthPotion());
    items.emplace_back(new ManaPotion());
    items.emplace_back(new Head(GameType::MAGIC_HAT));
    items.emplace_back(new Head(GameType::NO_HELMET));
    items.emplace_back(new Shield(GameType::IRON_SHIELD));
    items.emplace_back(new Chest(GameType::PLATE_ARMOR));
    items.emplace_back(new Chest(GameType::COMMON_CLOTHING));
    items.emplace_back(new Weapon(GameType::LONGSWORD));
    items.emplace_back(new Weapon(GameType::FIST));
    map.addItemsToTile(std::move(items), {1, 1});


    int i = 0;
    for (const auto & item: map.tiles[1][1].items) {
        if (itemsNames[i] ≠ item→getName()) {
            return false;
        }
        i++;
    }
    return true;
}


bool MapTests::testGetTargetsOnEmptyMapReturnsEmptyList() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    std::vector<Coordinate> targets;
    map.getAttackTargets({25, 25}, 25, targets);
    //map.getTargets({25, 25}, 25, targets);
    return targets.empty();
}

bool MapTests::testGetTargetsOnMapWithPlayerReturnsListWithOneElement() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    std::shared_ptr<Player> player(new Player(game.get(),
            {25,25}, data));
    map.addEntity({25, 25}, player);
    std::vector<Coordinate> targets;
    map.getAttackTargets({25, 25}, 25, targets);
    return (targets.size() ≡ 1) ∧ (targets[0].jPosition ≡ 25) ∧ (targets[0].iPos
ition ≡ 25);
}

bool MapTests::testGetTargetsOnMapWithDeadPlayerReturnsEmptyList() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    std::shared_ptr<Player> player(new Player(game.get(),
            {25,25}, PlayerData()));
    player→stats.currentLife = 0;
    map.addEntity({25, 25}, player);
    std::vector<Coordinate> targets;
    map.getAttackTargets({25, 25}, 25, targets);
    return targets.empty();
}

```

```cpp
bool MapTests::testPositionWithPlayerIsOccupied() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    std::shared_ptr<Player> player(new Player(game.get(),
            {25,25}, PlayerData()));
    map.addEntity({25, 25}, std::move(player));
    return ¬map.tiles[25][25].isAvailable();
}

bool MapTests::testPositionWithMonsterIsOccupied() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    std::shared_ptr<Monster> monster(new Monster(game.get(),
            {25, 25}, GameType::SKELETON, GameType::SKELETON_ATTACK));
    map.addEntity({25, 25}, std::move(monster));
    return ¬map.tiles[25][25].isAvailable();
}

bool MapTests::testListOnEmptyTileReturnsEmptyList() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    std::shared_ptr<Player> player(new Player(game.get(),
                                {0,0}, data));
    map.list(*player, {5, 5});
    return player→chat.getMessages().empty();
}

bool MapTests::testListOnEmptyMapReturnsEmptyList() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    Player player(game.get(), {0, 0}, data);
    if (¬player.chat.getMessages().empty()) return false;
    for (int i = 0; i < mapXSize; ++i) {
        for (int j = 0; j < mapYSize; ++j) {
            map.list(player, {i, j});
            if (¬player.chat.getMessages().empty()) {
                return false;
            }
        }
    }
    return true;
}

bool MapTests::testWithdrawOnEmptyTileGetsNoItem() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    Player player(game.get(), {0, 0}, PlayerData());
    map.withdraw(player, "product", {5, 5});
    return player.inventory.storedItemsAmount ≡ 0;
}

bool MapTests::testWithdrawOnEmptyMapGetsNoItem() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    Player player(game.get(), {0, 0}, PlayerData());
    for (int i = 0; i < mapXSize; ++i) {
        for (int j = 0; j < mapYSize; ++j) {
            map.withdraw(player, "product", {i, j});
            if (player.inventory.storedItemsAmount ≠ 0) {
                return false;
            }
        }
    }
    return true;
}

bool MapTests::testDepositExistentItemOnEmptyTileGetsNoItem() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    Player player(game.get(), {0, 0}, data);
    std::shared_ptr<Item> item(new Weapon(GameType::GNARLED_STAFF));
    player.storeItem(item);
    map.deposit(player, "product", {5, 5});
    return player.inventory.storedItemsAmount ≡ 1;
}

bool MapTests::testDepositExistentItemOnEmptyMapGetsNoItem() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    PlayerData data;
    data.isNewPlayer = true;
    Mock<Game> game;
    Player player(game.get(), {0, 0}, data);
    std::shared_ptr<Item> item(new Weapon(GameType::GNARLED_STAFF));
    player.storeItem(item);
    for (int i = 0; i < mapXSize; ++i) {
        for (int j = 0; j < mapYSize; ++j) {
            map.deposit(player, "product", {5, 5});
            if (player.inventory.storedItemsAmount ≠ 1) {
                return false;
            }
        }
    }
    return true;
}


bool MapTests::testBuyItemFromEmptyTileGetsNoItem() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
```

```cpp
    Player player(game.get(), {0, 0}, PlayerData());
    map.buy(player, "product", {5, 5});
    return player.inventory.storedItemsAmount ≡ 0;
}

bool MapTests::testBuyItemFromEmptyMapGetsNoItem() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    Player player(game.get(), {0, 0}, PlayerData());
    for (int i = 0; i < mapXSize; ++i) {
        for (int j = 0; j < mapYSize; ++j) {
            map.buy(player, "product", {5, 5});
            if (player.inventory.storedItemsAmount ≠ 0) {
                return false;
            }
        }
    }
    return true;
}

bool MapTests::testSellExistentItemToEmptyTileGetsNoItem() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    Player player(game.get(), {0, 0}, data);
    std::shared_ptr<Item> item(new Weapon(GameType::GNARLED_STAFF));
    player.storeItem(item);
    map.sell(player, "product", {5, 5});
    return player.inventory.storedItemsAmount ≡ 1;
}

bool MapTests::testSellExistentItemToEmptyMapGetsNoItem() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    Player player(game.get(), {0, 0}, data);
    std::shared_ptr<Item> item(new Weapon(GameType::GNARLED_STAFF));
    player.storeItem(item);
    for (int i = 0; i < mapXSize; ++i) {
        for (int j = 0; j < mapYSize; ++j) {
            map.sell(player, "product", {5, 5});
            if (player.inventory.storedItemsAmount ≠ 1) {
                return false;
            }
        }
    }
    return true;
}

bool MapTests::testMoveEntity() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
    Mock<Game> game;
    std::shared_ptr<Monster> monster(new Monster(game.get(), {25, 25}, GameType::SKELETON, GameType::SKELETON_ATTACK));
    map.addEntity({25, 25}, std::move(monster));
    map.moveEntity({25, 25}, {26, 26});
    return map.isPlaceAvailable({25, 25}) ∧ ¬map.isPlaceAvailable({26, 26});
}

bool MapTests::testRemoveEntityOnEmptyTileLeavesNoEntity() {
    Map map;
    int mapXSize = 50;
    int mapYSize = 50;
    _fillEmptyMap(map, mapXSize, mapYSize);
//    Mock<Game> game;
    map.removeEntity({5, 5});
    return map.isPlaceAvailable({5, 5});
}

```

```
//
// Created by agustin on 22/6/20.
//

#include "catch.hpp"
#include "MapTests.h"

TEST_CASE("Test Available Map Has Available Tiles") {
    REQUIRE(MapTests::testAvailableMapHasAvailableTiles());
}

TEST_CASE("Test City Map Has City Tiles") {
    REQUIRE(MapTests::testCityMapHasCityTiles());
}

TEST_CASE("Test Mixed City And Unavailable Tiles") {
    REQUIRE(MapTests::testMixedCityAndUnavailableTiles());
}

TEST_CASE("Test Added Items To Map") {
    REQUIRE(MapTests::testAddedItemsToMap());
}

TEST_CASE("Test Added Gold To Map") {
    REQUIRE(MapTests::testAddedGoldToMap());
}

TEST_CASE("Test Added Multiple Golds To Map With List") {
    REQUIRE(MapTests::testAddedMultipleGoldsToMapWithList());
}

TEST_CASE("Test Added Multiple Golds To Map Without List") {
    REQUIRE(MapTests::testAddedMultipleGoldsToMapWithoutList());
}

TEST_CASE("Test Added Multiple Items Lists To Map") {
    REQUIRE(MapTests::testAddedMultipleItemsListsToMap());
}

TEST_CASE("Test List Items On Sale On Empty Tile") {
    REQUIRE(MapTests::testListOnEmptyTileReturnsEmptyList());
}

TEST_CASE("Test List Items On Sale On Empty Map") {
    REQUIRE(MapTests::testListOnEmptyMapReturnsEmptyList());
}

TEST_CASE("Test Get Targets On Empty Map Returns Empty List") {
    REQUIRE(MapTests::testGetTargetsOnEmptyMapReturnsEmptyList());
}

TEST_CASE("Test Get Targets On Map With Player Returns List With One Element") {
    REQUIRE(MapTests::testGetTargetsOnMapWithPlayerReturnsListWithOneElement());
}

TEST_CASE("Test Get Targets On Map With Dead Player Returns Empty List") {
    REQUIRE(MapTests::testGetTargetsOnMapWithDeadPlayerReturnsEmptyList());
}

TEST_CASE("Test Position With Player Is Occupied") {
    REQUIRE(MapTests::testPositionWithPlayerIsOccupied());
}

TEST_CASE("Test Position With Monster Is Occupied") {
    REQUIRE(MapTests::testPositionWithMonsterIsOccupied());
}
```

```cpp


TEST_CASE("Test Withdraw On Empty Tile Gets No Item") {
    REQUIRE(MapTests::testWithdrawOnEmptyTileGetsNoItem());
}

TEST_CASE("Test Withdraw On Empty Map Gets No Item") {
    REQUIRE(MapTests::testWithdrawOnEmptyMapGetsNoItem());
}


TEST_CASE("Test Deposit Existant Item On Empty Tile Gets No Item") {
    REQUIRE(MapTests::testDepositExistentItemOnEmptyTileGetsNoItem());
}

TEST_CASE("Test Deposit Existant Item On Empty Map Gets No Item") {
    REQUIRE(MapTests::testDepositExistentItemOnEmptyMapGetsNoItem());
}

TEST_CASE("Test Buy Item From Empty Tile Gets No Item") {
    REQUIRE(MapTests::testBuyItemFromEmptyTileGetsNoItem());
}

TEST_CASE("Test Buy Item From Empty Map Gets No Item") {
    REQUIRE(MapTests::testBuyItemFromEmptyMapGetsNoItem());
}

TEST_CASE("Test Sell Existent Item To Empty Tile Gets No Item") {
    REQUIRE(MapTests::testSellExistentItemToEmptyTileGetsNoItem());
}

TEST_CASE("Test Sell Existent Item To Empty Map Gets No Item") {
    REQUIRE(MapTests::testSellExistentItemToEmptyMapGetsNoItem());
}

TEST_CASE("Test Move Entity") {
    REQUIRE(MapTests::testMoveEntity());
}

TEST_CASE("Test Remove Entity On Empty Tile Leaves No Entity") {
    REQUIRE(MapTests::testRemoveEntityOnEmptyTileLeavesNoEntity());
}
```

```cpp
//
// Created by agustin on 22/6/20.
//

#ifndef ARGENTUM_ITEMTESTS_H
#define ARGENTUM_ITEMTESTS_H

class Configuration;

class ItemTests {
public:
    static bool testInitialValues();
    static bool testAreNonGoldItemsGold();
    static bool testIsGoldItemGold();
    static bool testCorrectItemsNames();
    static bool testCorrectGoldAmount();

private:
    static bool _testCorrectItemsNamesHelmets(Configuration& config);
    static bool _testCorrectItemsNamesWeapons(Configuration& config);
    static bool _testCorrectItemsNamesPotions(Configuration& config);
    static bool _testCorrectItemsNamesShields(Configuration& config);
    static bool _testCorrectItemsNamesClothing(Configuration& config);
};


#endif //ARGENTUM_ITEMTESTS_H
```

```
//
// Created by agustin on 22/6/20.
//

#include "ItemTests.h"
#include "../Items/Item.h"
#include "../Items/Miscellaneous/Gold.h"
#include "../Items/Attack/Weapon.h"
#include "../Items/Defense/Chest.h"
#include "../Items/Defense/Head.h"
#include "../Items/Defense/Shield.h"
#include "../Items/Miscellaneous/HealthPotion.h"
#include "../Items/Miscellaneous/ManaPotion.h"

bool ItemTests::testInitialValues() {
    std::string name = "Mi nombre es Item!";
    Item item(GameType::ITEM_TYPE_CLOTHING, name/*, price*/);
    bool status = (item.getName() ≡ name);
    status = status ∧ (item.type ≡ GameType::ITEM_TYPE_CLOTHING);
    return status;
}

bool ItemTests::testAreNonGoldItemsGold() {
    Chest armour(GameType::Clothing::PLATE_ARMOR);
    Head helmet(GameType::Clothing::IRON_HELMET);
    Shield shield(GameType::Clothing::TURTLE_SHIELD);
    bool status = armour.isGold();
    status = (status ∨ helmet.isGold());
    status = (status ∨ shield.isGold());
    return (¬status);
}

bool ItemTests::testIsGoldItemGold() {
    Gold gold(100);
    return (gold.isGold());
}

bool ItemTests::_testCorrectItemsNamesClothing(Configuration& config) {
    Chest chest1(GameType::Clothing::COMMON_CLOTHING);
    if (chest1.getName() ≠ config.configClothingData(GameType::Clothing::COMMON_CLOTHING).name) return false;
    Chest chest2(GameType::Clothing::LEATHER_ARMOR);
    if (chest2.getName() ≠ config.configClothingData(GameType::Clothing::LEATHER_ARMOR).name) return false;
    Chest chest3(GameType::Clothing::PLATE_ARMOR);
    if (chest3.getName() ≠ config.configClothingData(GameType::Clothing::PLATE_ARMOR).name) return false;
    Chest chest4(GameType::Clothing::BLUE_TUNIC);
    return ¬(chest4.getName() ≠ config.configClothingData(GameType::Clothing::BLUE_TUNIC).name);
}

bool ItemTests::_testCorrectItemsNamesHelmets(Configuration& config) {
    Head helmet1(GameType::Clothing::HOOD);
    if (helmet1.getName() ≠ config.configClothingData(GameType::Clothing::HOOD).name) return false;
    Head helmet2(GameType::Clothing::IRON_HELMET);
    if (helmet2.getName() ≠ config.configClothingData(GameType::Clothing::IRON_HELMET).name) return false;
    Head helmet3(GameType::Clothing::MAGIC_HAT);
    if (helmet3.getName() ≠ config.configClothingData(GameType::Clothing::MAGIC_HAT).name) return false;
    Head helmet4(GameType::Clothing::NO_HELMET);
    return ¬(helmet4.getName() ≠ config.configClothingData(GameType::Clothing::NO_HELMET).name);
}
```

```

bool ItemTests::_testCorrectItemsNamesShields(Configuration& config) {
    Shield shield1(GameType::Clothing::IRON_SHIELD);
    if (shield1.getName() ≠ config.configClothingData(GameType::Clothing::IRON_SHIELD).name) return false;
    Shield shield2(GameType::Clothing::TURTLE_SHIELD);
    if (shield2.getName() ≠ config.configClothingData(GameType::Clothing::TURTLE_SHIELD).name) return false;
    Shield shield3(GameType::Clothing::NO_SHIELD);
    return ¬(shield3.getName() ≠ config.configClothingData(GameType::Clothing::NO_SHIELD).name);
}

bool ItemTests::_testCorrectItemsNamesWeapons(Configuration& config) {
    Weapon weapon1(GameType::Weapon::LONGSWORD);
    if (weapon1.getName() ≠ config.configWeaponData(GameType::Weapon::LONGSWORD).name) return false;
    Weapon weapon2(GameType::Weapon::AXE);
    if (weapon2.getName() ≠ config.configWeaponData(GameType::Weapon::AXE).name) return false;
    Weapon weapon3(GameType::Weapon::WARHAMMER);
    if (weapon3.getName() ≠ config.configWeaponData(GameType::Weapon::WARHAMMER).name) return false;
    Weapon weapon4(GameType::Weapon::ASH_ROD);
    if (weapon4.getName() ≠ config.configWeaponData(GameType::Weapon::ASH_ROD).name) return false;
    Weapon weapon5(GameType::Weapon::ELVEN_FLUTE);
    if (weapon5.getName() ≠ config.configWeaponData(GameType::Weapon::ELVEN_FLUTE).name) return false;
    Weapon weapon6(GameType::Weapon::LINKED_STAFF);
    if (weapon6.getName() ≠ config.configWeaponData(GameType::Weapon::LINKED_STAFF).name) return false;
    Weapon weapon7(GameType::Weapon::SIMPLE_BOW);
    if (weapon7.getName() ≠ config.configWeaponData(GameType::Weapon::SIMPLE_BOW).name) return false;
    Weapon weapon8(GameType::Weapon::COMPOSITE_BOW);
    if (weapon8.getName() ≠ config.configWeaponData(GameType::Weapon::COMPOSITE_BOW).name) return false;
    Weapon weapon9(GameType::Weapon::GNARLED_STAFF);
    if (weapon9.getName() ≠ config.configWeaponData(GameType::Weapon::GNARLED_STAFF).name) return false;
    Weapon weapon10(GameType::Weapon::FIST);
    return ¬(weapon10.getName() ≠ config.configWeaponData(GameType::Weapon::FIST).name);
}

bool ItemTests::_testCorrectItemsNamesPotions(Configuration& config) {
    HealthPotion potion1;
    if (potion1.getName() ≠ config.configPotionData(GameType::Potion::HEALTH_POTION).name) return false;
    ManaPotion potion2;
    return ¬(potion2.getName() ≠ config.configPotionData(GameType::Potion::MANA_POTION).name);
}

bool ItemTests::testCorrectItemsNames() {
    Configuration& config = Configuration::getInstance();
    bool status = _testCorrectItemsNamesClothing(config);
    status = status ∧ _testCorrectItemsNamesHelmets(config);
    status = status ∧ _testCorrectItemsNamesPotions(config);
    status = status ∧ _testCorrectItemsNamesShields(config);
    status = status ∧ _testCorrectItemsNamesWeapons(config);
    return status;
}

bool ItemTests::testCorrectGoldAmount() {
    unsigned int amount = 504;
    Gold gold(amount);
    return (amount ≡ gold.getAmount());
}
```

```
//
// Created by marcos on 22/6/20.
//

#include "catch.hpp"
#include "ItemTests.h"

TEST_CASE("Initial Item Values Test") {
    REQUIRE(ItemTests::testInitialValues());
}

TEST_CASE("It Is Not Gold Test") {
    REQUIRE(ItemTests::testAreNonGoldItemsGold());
}

TEST_CASE("It Is Gold Test") {
    REQUIRE(ItemTests::testIsGoldItemGold());
}

TEST_CASE("Load All Items Names Correctly Test") {
    REQUIRE(ItemTests::testCorrectItemsNames());
}

TEST_CASE("Correct Gold Amount Test") {
    REQUIRE(ItemTests::testCorrectGoldAmount());
}
```

```
//
// Created by agustin on 22/6/20.
//

#ifndef ARGENTUM_ENTITYTESTS_H
#define ARGENTUM_ENTITYTESTS_H


#include "../Game/Game.h"

class EntityTests {
public:
    static bool testStoreItem();
    static bool testIsMonsterTarget();
    static bool testSpendGold();
    static bool testItemUse();
    static bool testPlayerNickname();
    static bool testLifeAndManaRecovery();
    static bool testUnequipGear();
    static bool testPlayerAttacksMonster();
    static bool testPlayerAttacksMonsterAndConsumesMana();
    static bool testPlayerAttacksNewbieAndViceversa();
    static bool testPlayerAttacksPlayerWithPastLevelDifferenceAndViceversa();
    static bool testPlayersAttackEachOther();
    static bool testMonsterAttacksPlayer();
    static bool testPlayerSellsItem();
    static bool testPlayerDepositsAnItem();

private:
    static bool _testUnequipWeapon(Game &game);
    static bool _testUnequipClothing(Game &game);
    static void _fillEmptyMap(Map &map, int iSize, int jSize, bool isCity);
};


#endif //ARGENTUM_ENTITYTESTS_H
```

```
//
// Created by agustin on 22/6/20.
//

#include "EntityTests.h"
#include <memory>
#include "../Items/Attack/Weapon.h"
#include "../Entities/Player.h"
#include "../Config/Configuration.h"
#include "../Items/Miscellaneous/Gold.h"
#include "../Items/Defense/Chest.h"
#include "../Entities/AttackResult.h"
#include "../Entities/Monster.h"
#include "../Entities/Citizens/Priest.h"
#include "../Entities/Citizens/Trader.h"

#include "catch.hpp"
#include "fakeit.hpp"
#include "../Entities/Citizens/Banker.h"

using namespace fakeit;


bool EntityTests::testStoreItem() {
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    Configuration& config = Configuration::getInstance();
    Player player(game.get(), {0,0}, data);
    std::shared_ptr<Item> item(new Weapon(GameType::Weapon::LONGSWORD));
    player.storeItem(item);
    return (player.removeItem(config.configWeaponData(GameType::Weapon::LONGSWOR
D).name)→getName()
            ≡ config.configWeaponData(GameType::Weapon::LONGSWORD).name);
}

bool EntityTests::testIsMonsterTarget() {
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    Player player(game.get(), {0,0}, data);
    if (¬player.isMonsterTarget()) return false;
    player.stats.currentLife = 0;
    return ¬player.isMonsterTarget();
}

bool EntityTests::testSpendGold() {
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    Player player(game.get(),{0,0}, data);
    player.receiveGold(30);
    if (player.gold ≠ 30) return false;
    std::shared_ptr<Item> gold(new Gold(105));
    player.storeItem(gold);
    if (player.gold ≠ 135) return false;
    player.spendGold(15);
    return (player.gold ≡ 120);
}

bool EntityTests::testItemUse() {
    Configuration& config = Configuration::getInstance();
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    Player player(game.get(), {0,0}, data);
    player.useItem(0); /*No deberia hacer nada*/
    if (player.inventory.equippedWeapon→getName() ≠
            config.configWeaponData(GameType::Weapon::FIST).name) return false;
    player.useItem(15); /*No deberia hacer nada*/
    if (player.inventory.equippedWeapon→getName() ≠
        config.configWeaponData(GameType::Weapon::FIST).name) return false;
    std::shared_ptr<Item> item(new Weapon(GameType::Weapon::LINKED_STAFF));
    player.storeItem(item);
    player.useItem(0); /*Deberia equiparse el LinkedStaff*/
    return (player.inventory.equippedWeapon→getName() ≡ config.configWeaponData(GameType::Weapon::LINKED_STAFF).name);
}

bool EntityTests::testPlayerNickname() {
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    data.nickname = "ElPantuflas";
    Player player(game.get(), {0,0}, data);
    return player.getNickname() ≡ "ElPantuflas";
}

bool EntityTests::testLifeAndManaRecovery() {
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    Player player(game.get(), {0,0}, data);
    int life = player.stats.getCurrentLife();
    player.stats.currentLife −= 10;
    player.restoreLife(55);
    if (player.stats.getCurrentLife() ≠ life) return false;
    int32_t mana = player.stats.getCurrentMana();
    player.stats.currentMana −= 10;
    player.restoreMana(55);
    return player.stats.getCurrentMana() ≡ mana;
}

bool EntityTests::_testUnequipWeapon(Game& game) {
    PlayerData data;
    data.isNewPlayer = true;
    Player player(game, {0,0}, data);
    player.unequip(); /*No deberia hacer nada*/
    if (player.inventory.items[0]) return false;
    std::shared_ptr<Item> item(new Weapon(GameType::GNARLED_STAFF));
    player.storeItem(item);
    player.useItem(0);
    if (player.inventory.items[0]) return false;
    if (¬player.inventory.equippedWeapon) return false;
    player.unequip();
    if (¬player.inventory.items[0]) return false;
    return (player.inventory.equippedWeapon→getId() ≡ GameType::FIST);
}

bool EntityTests::_testUnequipClothing(Game& game) {
    PlayerData data;
    data.isNewPlayer = true;
    Player player(reinterpret_cast<Game &>(game), {0,0}, data);
    player.unequip(GameType::EQUIPMENT_PLACE_CHEST); /*No deberia hacer nada*/
    if (player.inventory.items[0]) return false;
    std::shared_ptr<Item> item(new Chest(GameType::PLATE_ARMOR));
    player.storeItem(item);
    player.useItem(0);
    if (player.inventory.items[0]) return false;
    if (¬player.inventory.clothingEquipment.at(GameType::EQUIPMENT_PLACE_CHEST)) return false;
    player.unequip(GameType::EQUIPMENT_PLACE_CHEST);
```

```
    if (¬player.inventory.items[0]) return false;
    return (player.inventory.clothingEquipment.at(
            GameType::EQUIPMENT_PLACE_CHEST)→getId() ≡ GameType::COMMON_CLOTHING);
}

bool EntityTests::testUnequipGear() {
    Mock<Game> game;
    if (¬_testUnequipWeapon(game.get())) return false;
    return _testUnequipClothing(game.get());
}

bool EntityTests::testPlayerAttacksMonster() {
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    _fillEmptyMap(game.get().map, 10, 10, false);
    Player player(game.get(), {0,0}, data);
    std::shared_ptr<Monster> monster(new Monster(game.get(), {0, 1},
                                                  GameType::SPIDER, GameType::SPIDER_ATTACK));
    monster→stats.agility = 0; /*Para que no esquive el ataque*/
    game.get().map.addEntity({0, 1}, std::static_pointer_cast<Entity>(monster));
    player.attack({0, 1});
    return (monster→stats.getCurrentLife() ≠ monster→stats.getMaxLife());
}

bool EntityTests::testPlayerAttacksMonsterAndConsumesMana() {
    Mock<Game> game;
    Fake(Method(game, pushEvent));
    PlayerData data;
    data.isNewPlayer = true;
    _fillEmptyMap(game.get().map, 10, 10, false);
    Player player(game.get(), {0,0}, data);
    player.stats.level = 50; /*Para que no suba de nivel y se le restore el mana*/
    std::shared_ptr<Monster> monster(new Monster(game.get(), {0, 1},
                                        GameType::SKELETON, GameType::SKELETON_ATTACK));
    monster→stats.agility = 0; /*Para que no esquive el ataque*/
    game.get().map.addEntity({0, 1}, std::static_pointer_cast<Entity>(monster));
    std::shared_ptr<Item> weapon(new Weapon(GameType::ASH_ROD));
    player.storeItem(weapon);
    player.useItem(0);
    if (player.stats.getCurrentMana() ≠ player.stats.maxMana) return false;
    player.attack({0, 1});
    if (monster→stats.getCurrentLife() ≡ monster→stats.getMaxLife()) return false;
    return (player.stats.getCurrentMana() ≠ player.stats.maxMana);
}

bool EntityTests::testPlayerAttacksNewbieAndViceversa() {
    Mock<Game> game;
    Mock<Map> map;
    _fillEmptyMap(map.get(), 10, 10, false);
    std::shared_ptr<Player> player1(new Player(game.get(), {0,0}, PlayerData()));
    std::shared_ptr<Player> player2(new Player(game.get(), {0,1}, PlayerData()));
    std::shared_ptr<Entity> aux = player1;
    map.get().addEntity({0, 0}, std::move(aux));
    aux = player2;
    map.get().addEntity({0, 1}, std::move(aux));
    player1→stats.agility = 0; /*Para que no esquiven*/
    player2→stats.agility = 0; /*Para que no esquiven*/
    std::shared_ptr<Item> weapon(new Weapon(GameType::LONGSWORD));
    player1→storeItem(weapon);
    player1→useItem(0);
    weapon.reset(new Weapon(GameType::WARHAMMER));
    player2→storeItem(weapon);
    player2→useItem(0);
    player1→attack({0, 1});
    player2→attack({0, 0});
    if (player2→stats.getCurrentLife() ≠ player2→stats.getMaxLife()) return fa
lse;
    return (player1→stats.getCurrentLife() ≡ player1→stats.getMaxLife());
}

bool EntityTests::testPlayerAttacksPlayerWithPastLevelDifferenceAndViceversa() {
    Mock<Game> game;
    Mock<Map> map;
    _fillEmptyMap(map.get(), 10, 10, false);
    std::shared_ptr<Player> player1(new Player(game.get(), {0,0}, PlayerData()))
;
    std::shared_ptr<Player> player2(new Player(game.get(), {0,1}, PlayerData()))
;
    std::shared_ptr<Entity> aux = player1;
    map.get().addEntity({0, 0}, std::move(aux));
    aux = player2;
    map.get().addEntity({0, 1}, std::move(aux));
    player1→stats.agility = 0; /*Para que no esquiven*/
    player2→stats.agility = 0; /*Para que no esquiven*/
    std::shared_ptr<Item> weapon(new Weapon(GameType::LONGSWORD));
    player1→storeItem(weapon);
    player1→useItem(0);
    weapon.reset(new Weapon(GameType::WARHAMMER));
    player2→storeItem(weapon);
    player2→useItem(0);
    player1→attack({0, 1});
    player2→attack({0, 0});
    if (player2→stats.getCurrentLife() ≠ player2→stats.getMaxLife()) return fa
lse;
    return (player1→stats.getCurrentLife() ≡ player1→stats.getMaxLife());
}

bool EntityTests::testPlayersAttackEachOther() {
    Mock<Game> game;
    Configuration& config = Configuration::getInstance();
    _fillEmptyMap(game.get().map, 10, 10, false);
    PlayerData data;
    data.isNewPlayer = true;
    data.level = config.configNewbieLevel() + 1;
    std::shared_ptr<Player> player1(new Player(game.get(), {0,0}, data));
    std::shared_ptr<Player> player2(new Player(game.get(), {0,1}, data));
    std::shared_ptr<Entity> aux = player1;
    game.get().map.addEntity({0, 0}, std::move(aux));
    aux = player2;
    game.get().map.addEntity({0, 1}, std::move(aux));
    player1→stats.agility = 0; //Para que no esquiven
    player2→stats.agility = 0; //Para que no esquiven
    std::shared_ptr<Item> weapon(new Weapon(GameType::LONGSWORD));
    player1→storeItem(weapon);
    player1→useItem(0);
    weapon.reset(new Weapon(GameType::WARHAMMER));
    player2→storeItem(weapon);
    player2→useItem(0);
    player1→attack({0, 1});
    player2→attack({0, 0});
    if (player2→stats.getCurrentLife() ≡ player2→stats.getMaxLife()) return fa
lse;
    return (player1→stats.getCurrentLife() ≠ player1→stats.getMaxLife());
}
```

```

bool EntityTests::testMonsterAttacksPlayer() {
    Mock<Game> game;
    Fake(Method(game, pushEvent));
    _fillEmptyMap(game.get().map, 10, 10, false);
    PlayerData data;
    data.isNewPlayer = true;
    std::shared_ptr<Player> player(new Player(game.get(), {0,0}, data));
    MonstersFactory factory;
    std::shared_ptr<Monster> monster;
    factory.storeRandomMonster(game.get(), monster);
    player→stats.agility = 0; /*Para que no esquive el ataque*/
    game.get().map.addEntity({0, 1}, std::static_pointer_cast<Entity>(player));
    monster→attack({0, 1});
    return (player→stats.getCurrentLife() ≠ player→stats.getMaxLife());
}

bool EntityTests::testPlayerSellsItem() {
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    Player player(game.get(), {0,0}, data);
    Trader trader({0, 1});
    std::shared_ptr<Item> weapon(new Weapon(GameType::LONGSWORD));
    player.storeItem(weapon);
    trader.shop.storage.storedItems.at("Longsword").clear();
    if (¬trader.shop.storage.storedItems.at("Longsword").empty()) return false;
    trader.sell(player, "Longsword");
    return (¬trader.shop.storage.storedItems.at("Longsword").empty());
}

bool EntityTests::testPlayerDepositsAnItem() {
    Mock<Game> game;
    PlayerData data;
    data.isNewPlayer = true;
    Player player(game.get(), {0,0}, data);
    Banker banker({0, 1});
    std::shared_ptr<Item> weapon(new Weapon(GameType::LONGSWORD));
    player.storeItem(weapon);
    std::unordered_map<std::string, unsigned int> aux;
    data = player.getData();
    Banker::addPlayerItems(data);
    banker.deposit(player, "Longsword");
    banker.deposit(player, "Longsword"); //No deberia hacer nada
    if (player.inventory.items[0]) return false;
    banker.withdraw(player, "Longsword");
    return (player.inventory.items[0]→getName() ≡ "Longsword");
}

void EntityTests::_fillEmptyMap(Map &map, int iSize, int jSize, bool isCity) {
    for (int i = 0; i < iSize; ++i) {
        map.tiles.emplace_back();
        for (int j = 0; j < jSize; ++j) {
            map.tiles[i].emplace_back(true, isCity, GameType::FloorType::GRASS0,
                                      GameType::Structure::NO_STRUCTURE,
                                      std::shared_ptr<Entity>(nullptr));
        }
    }
}


```

```
//
// Created by marcos on 22/6/20.
//

#include "catch.hpp"
#include "EntityTests.h"

TEST_CASE("Store Item In Player Inventory Test") {
    REQUIRE(EntityTests::testStoreItem());
}

TEST_CASE("Player Is Monster Target Test") {
    REQUIRE(EntityTests::testIsMonsterTarget());
}

TEST_CASE("Gold Management By Player Test") {
    REQUIRE(EntityTests::testSpendGold());
}

TEST_CASE("Item Management By Player Test") {
    REQUIRE(EntityTests::testItemUse());
}

TEST_CASE("Correct Player Nickname Test") {
    REQUIRE(EntityTests::testPlayerNickname());
}

TEST_CASE("Life And Mana Recovery By Player Test") {
    REQUIRE(EntityTests::testLifeAndManaRecovery());
}

TEST_CASE("Unequip Gear Test") {
    REQUIRE(EntityTests::testUnequipGear());
}

TEST_CASE("Player Attacks Monster And Damages It Test") {
    REQUIRE(EntityTests::testPlayerAttacksMonster());
}

TEST_CASE("Player Attacks Monster And Consumes Weapon Mana Test") {
    REQUIRE(EntityTests::testPlayerAttacksMonsterAndConsumesMana());
}

TEST_CASE("Player Attacks Newbie And Viceversa Test") {
    REQUIRE(EntityTests::testPlayerAttacksNewbieAndViceversa());
}

TEST_CASE("Player Attacks Player With Past Level Difference And Viceversa Test") {
    REQUIRE(EntityTests::testPlayerAttacksPlayerWithPastLevelDifferenceAndViceve
rsa());
}

TEST_CASE("Players Attack Each Other Test") {
    REQUIRE(EntityTests::testPlayersAttackEachOther());
}

TEST_CASE("Monster Attacks Player Test") {
    REQUIRE(EntityTests::testMonsterAttacksPlayer());
}

TEST_CASE("Trader And Priest Buy Item From Player") {
    REQUIRE(EntityTests::testPlayerSellsItem());
}

TEST_CASE("Player Deposits And Withdraws An Item") {
    REQUIRE(EntityTests::testPlayerDepositsAnItem());
```

```
}


```

```
//
// Created by marcos on 6/24/20.
//

#ifndef ARGENTUM_SERVERPROTOCOL_H
#define ARGENTUM_SERVERPROTOCOL_H

#include "../Items/ItemData.h"
#include <msgpack.hpp>

class Player;
class Monster;
class Entity;
class PlayerProxy;
class Game;
class Item;

//Esta clase se encarga de almacenar de la forma apropiada la informacion a mand
ar a los
//clientes
class ServerProtocol {
private:
    std::vector<char> mapBuffer;
    std::stringstream generalData;
    std::vector<char> generalDataBuffer;
    std::vector<char> currentStateBuffer;
    const Game& game;

private:
    static void _loadBytes(std::vector<char>& buffer, void* data, unsigned int s
ize);

public:
    explicit ServerProtocol(const Game& game);

    //Retorna el buffer que contiene la informacion del mapa que no cambia
    const std::vector<char>& getMapInfo() const;

    //Arma el buffer que almacena todos los datos necesarios para que se conecte
 un
    //player con la informacion inicial apropiada y retorna una referencia a el
    const std::vector<char>& buildCurrentState(
                            const std::unordered_map<std::string, Player*>&
players,
                            const std::list<Monster*>& monsters,
                            const std::unordered_map<Coordinate, const Item*
>& mapItems);

    //Agrega la informacion del stringstream al buffer que contiene la informaci
on
    //general que se mandara a todos los clientes
    void addToGeneralData(std::stringstream& data);

    //Arma el mensaje a mandar con la informacion general, resetea el stringstre
am
    //que guarda la informacion general
    void buildGeneralDataBuffer();

    //Retorna una referencia al buffer que contiene toda la informacion de lo pa
sado
    //en el ultimo update de game
    const std::vector<char>& getGeneralData();

    //Retorna un buffer que contiene la informacion del player que almacena el
    //PlayerProxy
    static std::vector<char> getPlayerData(PlayerProxy& player);
```

```
};


#endif //ARGENTUM_SERVERPROTOCOL_H
```

```
//
// Created by marcos on 6/24/20.
//

#include "ServerProtocol.h"
#include <iostream>
#include "../Entities/PlayerProxy.h"
#include "../Entities/Player.h"
#include "../Entities/Monster.h"
#include "../Game/Game.h"
#include "../Items/ItemData.h"

#include <msgpack.hpp>

MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::ItemType)


///////////////////////////////PUBLIC/////////////////////////////

ServerProtocol::ServerProtocol(const Game& _game): game(_game) {
    std::stringstream aux;
    game.getMap() >> aux;
    uint32_t msgLength = aux.str().size();
    msgLength = htonl(msgLength); /*Enviamos la longitud en big endian 4 bytes*/
    mapBuffer.resize(sizeof(uint32_t));
    _loadBytes(mapBuffer, &msgLength, sizeof(uint32_t));
    std::string auxStr = aux.str();
    std::copy(auxStr.begin(), auxStr.end(), std::back_inserter(mapBuffer));
}

const std::vector<char> &ServerProtocol::getMapInfo() const {
    return mapBuffer;
}

const std::vector<char>& ServerProtocol::buildCurrentState(
                                          const std::unordered_map<std::string,
Player*>& players,
                                          //const std::list<Player*>& players,
                                          const std::list<Monster*>& monsters,
                                          const std::unordered_map<Coordinate, c
onst Item*>& mapItems) {
    std::stringstream data;
    for (const auto & player : players) {
        (*player.second) >> data;
    }
    for (const auto & monster : monsters) {
        (*monster) >> data;
    }
    for (const auto & item : mapItems) {
        item.second→loadDropItemData(data, item.first.iPosition, item.first.jPo
sition);
    }
    std::string auxString = data.str();
    uint32_t msgLength = htonl(auxString.size());
    currentStateBuffer.resize(sizeof(uint32_t));
    _loadBytes(currentStateBuffer, &msgLength, sizeof(uint32_t));
    std::copy(auxString.begin(), auxString.end(), std::back_inserter(currentStat
eBuffer));
    return currentStateBuffer;
}

void ServerProtocol::addToGeneralData(std::stringstream &data) {
    generalData << data.str();
}

```

```
const std::vector<char>& ServerProtocol::getGeneralData() {
    return generalDataBuffer;
}

void ServerProtocol::buildGeneralDataBuffer() {
    std::string auxString = generalData.str();
    uint32_t msgLength = htonl(auxString.size());
    generalDataBuffer.resize(sizeof(uint32_t));
    _loadBytes(generalDataBuffer, &msgLength, sizeof(uint32_t));
    std::copy(auxString.begin(), auxString.end(), std::back_inserter(generalData
Buffer));
    generalData.str("");
    generalData.clear();
}

std::vector<char> ServerProtocol::getPlayerData(PlayerProxy& player) {
    std::stringstream data;
    player.storeAllRelevantData(data);
    player.clearMinichat();
    std::string auxString = data.str();
    uint32_t msgLength = htonl(auxString.size());
    std::vector<char> buffer(sizeof(uint32_t));
    _loadBytes(buffer, &msgLength, sizeof(uint32_t));
    std::copy(auxString.begin(), auxString.end(), std::back_inserter(buffer));
    return buffer;
}

//////////////////////////////PRIVATE////////////////////////////

void ServerProtocol::_loadBytes(std::vector<char>& buffer, void* data, unsigned
int size) {
    for (unsigned int i = 0; i < size; ++i) {
        buffer[i] = *(reinterpret_cast<char *>(data) + i);
    }
}
```

```
#ifndef TP3TALLER_SERVERMONITOR_H
#define TP3TALLER_SERVERMONITOR_H

/*Esta clase es la que chequea cuando cerrar el server*/

#include "ArgentumServer.h"
#include "../../libs/Thread.h"

class ServerMonitor : public Thread {
private:
    ArgentumServer& server;
    bool reading{true};

public:
    explicit ServerMonitor(ArgentumServer& server) : server(server) {}
    void join() override;

    /*Retorna true si se cerro el servidor a pedido del usuario,
     * false en caso contrario*/
    bool closeRequest();

private:
    /*Implementa la funcion run heredada de Thread, la cual para esta clase
    * correra el metodo stopOnCommand*/
    void run() override;
    void _stopOnCommand();
};


#endif //TP3TALLER_SERVERMONITOR_H
```

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

const char FINISH_CHAR = 'q';

void ServerMonitor::_stopOnCommand() {
    char input = 0;
    while (input ≠ FINISH_CHAR) {
        input = std::getchar();
    }
    reading = false;
    server.finish();
}

void ServerMonitor::run() {
    _stopOnCommand();
}

void ServerMonitor::join() {
    if (reading) {
        Thread::detach();
    } else {
        Thread::join();
    }
}

bool ServerMonitor::closeRequest() {
    return ¬reading;
}
```

```cpp
//
// Created by marcos on 6/28/20.
//

#ifndef ARGENTUM_PLAYERMANAGER_H
#define ARGENTUM_PLAYERMANAGER_H

#include <string>
#include "../../libs/GameEnums.h"
#include "../Persistence/SaveFileManager.h"

class Game;
class PlayerProxy;
class ServerProtocol;
struct PlayerData;

class PlayerManager {
private:
    Game& game;
    ServerProtocol& protocol;
    SaveFileManager saveManager;

public:
    PlayerManager(Game& _game, ServerProtocol& _protocol,
                            const std::string& indexPath,
                            const std::string& savePath) : game(_game),
                            protocol(_protocol), saveManager(indexPath, savePath
) {}

    PlayerProxy addPlayer(PlayerData& playerData);
    PlayerData getSavedPlayerData(const std::string& nickname);
    void storeNewPlayer(PlayerData& playerData);
    void storeOldPlayer(const PlayerData& playerData);
    void removePlayer(const std::string& nickname);
};


#endif //ARGENTUM_PLAYERMANAGER_H
```

```cpp
//
// Created by marcos on 6/28/20.
//

#include "PlayerManager.h"
#include "../Game/Game.h"
#include "PlayerData.hpp"
#include "../Entities/PlayerProxy.h"
#include "../Exceptions/UnavailablePlayerException.h"

PlayerProxy PlayerManager::addPlayer(PlayerData& playerData) {
    PlayerProxy player(&game, &game.createPlayer(playerData, protocol));
    return player;
}

PlayerData PlayerManager::getSavedPlayerData(const std::string &nickname) {
    if (¬game.playerExists(nickname)) {
        return saveManager.getPlayerData(nickname);
    }
    throw UnavailablePlayerException();
}

void PlayerManager::storeNewPlayer(PlayerData& playerData) {
    saveManager.storeNewPlayer(playerData);
}

void PlayerManager::storeOldPlayer(const PlayerData& playerData) {
    saveManager.storeOldPlayer(playerData);
}

void PlayerManager::removePlayer(const std::string &nickname) {
    game.removePlayer(nickname, protocol);
}
```

```
//
// Created by agustin on 19/7/20.
//

#ifndef ARGENTUM_NONMODIFIABLECONSTANTS_H
#define ARGENTUM_NONMODIFIABLECONSTANTS_H

const int INVENTORY_SIZE = 16;
const int BANK_SIZE = 20;
const int MAX_NICKNAME_SIZE = 13;
const unsigned int INITIAL_PLAYER_GOLD = 150;

#endif //ARGENTUM_NONMODIFIABLECONSTANTS_H
```

```
//
// Created by marcos on 6/3/20.
//

#include "Server/ArgentumServerSide.h"
#include "../libs/TPException.h"
#include <iostream>

int main(int argc, char** argv) {
    try {
        ArgentumServerSide::run(argc, argv);
    } catch (TPException& e) {
        std::cerr << e.what() << "in Server!" << std::endl;
    } catch (...) {
        std::cerr << "Uknown error in Server!" << std::endl;
    }
    return 0;
}
```

```cpp
//
// Created by agustin on 26/6/20.
//
#ifndef ARGENTUM_CLIENTSMONITOR_H
#define ARGENTUM_CLIENTSMONITOR_H

#include <memory>
#include <list>
#include "ClientHandler.h"
#include <mutex>
#include "PlayerData.hpp"

class PlayerProxy;
class PlayerManager;
class Game;
class ServerProtocol;

//Clase functor que se usa para saber cuando un cliente debe ser removido en la
//lista de clientes de ClientsMonitor
class ClientShouldBeRemoved {
private:
    ServerProtocol& protocol;
    PlayerManager& manager;

public:
    explicit ClientShouldBeRemoved(ServerProtocol& _protocol, PlayerManager& _ma
nager)
                                : protocol(_protocol), manager(_manager) {}

    //Si el cliente termino de ejecutarse (si se desconecta) guarda su informaci
on
    //en el archivo de persistencia, espera a que termine la ejecucion del threa
d,
    //y delega a PlayerProxy la eliminacion de este de la logica del juego
    bool operator()(std::unique_ptr<ClientHandler>& client);
};

//Clase que se encarga de proteger la lista de clientes frente a posibles race
//conditions
class ClientsMonitor {
private:
    std::mutex mutex;
    std::list<std::unique_ptr<ClientHandler>> clients;
    std::list<std::tuple<std::unique_ptr<ClientHandler>, PlayerData>> waitingLis
t;
    PlayerManager& manager;

public:
    explicit ClientsMonitor(PlayerManager& _manager) : manager(_manager) {}

    //Agrega un ClientHandler a la lista de espera de los nuevos jugadores conec
tados,
    //para que terminen de agregarse al juego se debe hacer un merge
    void pushToWaitingList(Socket ∧peer, ServerProtocol &protocol, PlayerData∧
playerData);

    //Crea el PlayerProxy y Player de cada client handler y se lo asigna, mandan
dole luego
    //el estado actual de juego actual y agregandolo a la lista de clientes acti
vos para
    //cada cliente en la lista de espera
    //Resetea la lista de espera
    void mergeWaitingClients(Game& game, ServerProtocol& protocol);

    //Llama a update para cada cliente (ClientHandler) activo
    void mergeClientsEvents();
```

```cpp

    //Elimina todos los jugadores desconectados de la lista de clientes activos
    void removeDisconnectedClients(ServerProtocol& protocol);

    //Envia a todos los clientes activos la informacion del update
    void sendGameUpdate();

    //Fuerza el cierre de todos los clientes y espera a que sus threads
    //terminen de ser ejecutados
    void join();

    //Indica si tiene clientes en la lista de espera, retorna true si es el caso
,
    //sino retorna false
    bool hasWaitingClients();

    //Guarda la informacion actual de todos los players activos en el archivo de
    //persistencia
    void backup();
};


#endif //ARGENTUM_CLIENTSMONITOR_H
```

```cpp
//
// Created by agustin on 26/6/20.
//

#include "ClientsMonitor.h"
#include "PlayerManager.h"
#include "../Game/Game.h"

void ClientsMonitor::join() {
    for (auto & client : clients) {
        client->forceFinish();
        client->join();
    }
}

void ClientsMonitor::mergeClientsEvents() {
    for (auto & client : clients) {
        client->update();
    }
}

void ClientsMonitor::pushToWaitingList(Socket &&peer, ServerProtocol &protocol,
                                       PlayerData&& playerData) {
    std::lock_guard<std::mutex> lock(mutex);
    waitingList.emplace_back(new ClientHandler(std::move(peer), protocol,
                             std::move(playerData));
}

void ClientsMonitor::mergeWaitingClients(Game& game, ServerProtocol& protocol) {
    std::lock_guard<std::mutex> lock(mutex);

    for (auto & waitingClient: waitingList) {
        PlayerData playerData = std::move(std::get<1>(waitingClient)); /*creo lo
s players*/
        std::get<0>(waitingClient)->setPlayerProxy(manager.addPlayer(playerData)
);
    }

    const std::vector<char>& gameState = game.getCurrentState(protocol); /*armo
el buffer*/

    for (auto & waitingClient: waitingList) { /*disparo los nuevos client handle
rs*/
        clients.push_back(std::move(std::get<0>(waitingClient)));
        (*clients.back()).sendCurrentGameState(gameState);
        (*clients.back())();
    }
    waitingList.clear();
}

void ClientsMonitor::sendGameUpdate() {
    for (const auto& client : clients) {
        client->sendGameUpdate();
    }
}

bool ClientsMonitor::hasWaitingClients() {
    return !waitingList.empty();
}

void ClientsMonitor::removeDisconnectedClients(ServerProtocol& protocol) {
    ClientShouldBeRemoved shouldBeRemoved(protocol, manager);
    clients.erase(std::remove_if(clients.begin(), clients.end(),
                                 shouldBeRemoved), clients.end());
}

```

```cpp
void ClientsMonitor::backup() {
    for (auto & client : clients) {
        PlayerData dataToStore = client->getPlayerData();
        manager.storeOldPlayer(dataToStore);
    }
}

bool ClientShouldBeRemoved::operator()(std::unique_ptr<ClientHandler> &client) {
    if (client->hasFinished()) {
        PlayerData dataToStore = client->getPlayerData();
        manager.storeOldPlayer(dataToStore);
        manager.removePlayer(dataToStore.nickname);
        client->join();
        return true;
    } else {
        return false;
    }
}
```

```
#ifndef TP3TALLER_CLIENTHANDLER_H
#define TP3TALLER_CLIENTHANDLER_H

/*Esta clase se comunica con el cliente, es decir, el servidor
 * cuando acepta a un cliente crea un nuevo ClientHandler (un estilo de
 * subservidor) y lo dispara en un nuevo thread. Esta clase guarda una instancia
 * del protocolo, que guarda una instancia independiente del Juego de Adivinar
 * el Numero*/
#include "../../libs/Socket.h"
#include "../../libs/Thread.h"
#include "../Entities/PlayerProxy.h"
#include <queue>
#include <vector>
#include <mutex>
#include <utility>
#include <atomic>
#include <msgpack.hpp>

class ServerProtocol;

class ClientHandler;

typedef void (ClientHandler::*processEvent)(std::vector<char>& data);

class ClientHandler : public Thread {
private:
    std::unordered_map<GameType::PlayerEvent, processEvent> eventProcessors;
    Socket socket;
    std::atomic<bool> finished{};
    std::vector<char> buffer;
    std::size_t offset{0};
    msgpack::object_handle handler;
    ServerProtocol& protocol;
    PlayerProxy player;
    std::mutex m;

public:
    ClientHandler(Socket∧ socket, ServerProtocol& _protocol);
    ClientHandler(const ClientHandler&) = delete;
    ClientHandler operator=(const ClientHandler&) = delete;

    //Retorna true si el socket ha terminado de comunicarse con su cliente
    bool hasFinished() const;

    //Le delega a PlayerProxy el otorgamiento de los eventos encolados a Game
    void update();

    //Envia toda la informacion del ultimo update del juego
    void sendGameUpdate();

    //Envia el estado inicial del juego
    void sendCurrentGameState(const std::vector<char>& gameState);

    //Cierra el socket y fuerza a que termine de ejecutarse el thread
    //que recibe los comandos del cliente
    void forceFinish();

    //Se apropia del PlayerProxy recibido
    void setPlayerProxy(PlayerProxy∧ _player);

    //Retorna los datos actuales del jugador del cliente
    PlayerData getPlayerData() const;

private:
    /*Implementa el metodo virtual run de Thread, que sera el metodo ejecutado
    * por el thread*/
```

```
    void run() override;
    void _processClientAction(std::vector<char>& data);
    //void _processMove(std::vector<char>& data);
    void _processAttack(std::vector<char>& data);
    void _processUseItem(std::vector<char>& data);
    void _processUnequip(std::vector<char>& data);
    void _processPickUp(std::vector<char>& data);
    void _processDrop(std::vector<char>& data);
    void _processList(std::vector<char>& data);
    void _processBuy(std::vector<char>& data);
    void _processSell(std::vector<char>& data);
    void _processWithdraw(std::vector<char>& data);
    void _processDeposit(std::vector<char>& data);
    void _processMeditate(std::vector<char>& data);
    void _processResurrect(std::vector<char>& data);
    void _processMessage(std::vector<char>& data);
    void _processHeal(std::vector<char>& data);
    void _processInventoryNames(std::vector<char>& data);
    void _processStartMoving(std::vector<char>& data);
    void _processStopMoving(std::vector<char>& data);
};


#endif //TP3TALLER_CLIENTHANDLER_H
```

```
#include "ClientHandler.h"
#include <vector>
#include <mutex>
#include "ServerProtocol.h"
#include "PlayerManager.h"
#include <iostream>
#include "../../libs/TPException.h"
#include <iostream>

MSGPACK_ADD_ENUM(GameType::PlayerEvent)
MSGPACK_ADD_ENUM(GameType::Race)
MSGPACK_ADD_ENUM(GameType::Class)
MSGPACK_ADD_ENUM(GameType::Direction)
MSGPACK_ADD_ENUM(GameType::EquipmentPlace)

///////////////////////////////PUBLIC///////////////////////////////

ClientHandler::ClientHandler(Socket ∧socket, ServerProtocol& _protocol) :
                        socket(std::move(socket)), protocol(_protocol) {
    eventProcessors = {{GameType::PLAYER_START_MOVING, &ClientHandler::_processStartMoving},
                       {GameType::PLAYER_STOP_MOVING, &ClientHandler::_processStopMoving},
                       {GameType::PLAYER_ATTACK, &ClientHandler::_processAttack},
                       {GameType::PLAYER_USE_ITEM, &ClientHandler::_processUseItem},
                       {GameType::PLAYER_UNEQUIP, &ClientHandler::_processUnequip},
                       {GameType::PLAYER_PICK_UP, &ClientHandler::_processPickUp},
                       {GameType::PLAYER_DROP, &ClientHandler::_processDrop},
                       {GameType::PLAYER_LIST, &ClientHandler::_processList},
                       {GameType::PLAYER_BUY, &ClientHandler::_processBuy},
                       {GameType::PLAYER_SELL, &ClientHandler::_processSell},
                       {GameType::PLAYER_WITHDRAW, &ClientHandler::_processWithdraw},
                       {GameType::PLAYER_DEPOSIT, &ClientHandler::_processDeposit},
                       {GameType::PLAYER_MEDITATE, &ClientHandler::_processMeditate},
                       {GameType::PLAYER_RESURRECT, &ClientHandler::_processResurrect},
                       {GameType::PLAYER_SEND_MSG, &ClientHandler::_processMessage},
                       {GameType::PLAYER_HEAL, &ClientHandler::_processHeal},
                       {GameType::PLAYER_REQUEST_INVENTORY_NAMES, &ClientHandler::_processInventoryNames}};
}

void ClientHandler::run() {
    try {
        uint32_t msgLength = 0;

        while (¬finished) {
            buffer.clear();
            socket.receive((char*)&(msgLength), sizeof(uint32_t));
            msgLength = ntohl(msgLength);
            buffer.resize(msgLength);
            socket.receive(buffer.data(), msgLength);
            _processClientAction(buffer);
        }

    } catch(std::exception& e) {
        socket.close();
        finished = true;
```

```
        std::cerr << e.what() << std::endl;
    }
}

void ClientHandler::sendGameUpdate() {
    try {
        const std::vector<char>& generalData = protocol.getGeneralData();
        socket.send(generalData.data(), generalData.size());
        std::vector<char> playerData = ServerProtocol::getPlayerData(player);
        socket.send(playerData.data(), playerData.size());
    } catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

bool ClientHandler::hasFinished() const {
    return finished;
}

void ClientHandler::update() {
    std::unique_lock<std::mutex> lk(m);
    player.giveEventsToGame();
}

void ClientHandler::sendCurrentGameState(const std::vector<char>& gameState) {
    try {
        const std::vector<char>& mapInfo = protocol.getMapInfo();
        socket.send(mapInfo.data(), mapInfo.size());
        socket.send(gameState.data(), gameState.size());
        std::vector<char> playerData = ServerProtocol::getPlayerData(player);
        socket.send(playerData.data(), playerData.size());
    } catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

void ClientHandler::forceFinish() {
    socket.close();
    finished = true;
}

void ClientHandler::setPlayerProxy(PlayerProxy∧ _player) {
    player = std::move(_player);
}

PlayerData ClientHandler::getPlayerData() const {
    return player.getData();
}

///////////////////////////////PRIVATE///////////////////////////////


void ClientHandler::_processClientAction(std::vector<char>& data) {
    offset = 0;
    msgpack::type::tuple<GameType::PlayerEvent> event;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler→convert(event);
    std::unique_lock<std::mutex> lk(m);
    try {
        (this→*eventProcessors.at(std::get<0>(event)))(data);
    } catch(std::out_of_range& e) {
        std::cerr << "Received an unknown command from the client" << std::endl;
    }
}

void ClientHandler::_processAttack(std::vector<char> &data) {
```

```cpp
    msgpack::type::tuple<int32_t, int32_t> attackInfo;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(attackInfo);
    player.attack({std::get<0>(attackInfo), std::get<1>(attackInfo)});
}

void ClientHandler::_processUseItem(std::vector<char> &data) {
    msgpack::type::tuple<int32_t> itemPosition;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(itemPosition);
    player.useItem(std::get<0>(itemPosition));
}

void ClientHandler::_processUnequip(std::vector<char> &data) {
    msgpack::type::tuple<GameType::EquipmentPlace> equipmentPlace;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(equipmentPlace);
    player.unequip(std::get<0>(equipmentPlace));
}

void ClientHandler::_processPickUp(std::vector<char> &data) {
    player.pickUpItem();
}

void ClientHandler::_processDrop(std::vector<char> &data) {
    msgpack::type::tuple<int32_t> itemPosition;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(itemPosition);
    player.dropItem(std::get<0>(itemPosition));
}

void ClientHandler::_processList(std::vector<char> &data) {
    msgpack::type::tuple<int32_t, int32_t> listPosition;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(listPosition);
    player.listFrom({std::get<0>(listPosition), std::get<1>(listPosition)});
}

void ClientHandler::_processBuy(std::vector<char> &data) {
    msgpack::type::tuple<std::string, int32_t, int32_t> buyArguments;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(buyArguments);
    player.buyFrom(std::move(std::get<0>(buyArguments)),
                   {std::get<1>(buyArguments), std::get<2>(buyArguments)});
}

void ClientHandler::_processSell(std::vector<char> &data) {
    msgpack::type::tuple<std::string, int32_t, int32_t> sellArguments;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(sellArguments);
    player.sellTo(std::move(std::get<0>(sellArguments)),
                   {std::get<1>(sellArguments), std::get<2>(sellArguments)});
}

void ClientHandler::_processWithdraw(std::vector<char> &data) {
    msgpack::type::tuple<std::string, int32_t, int32_t> sellArguments;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(sellArguments);
    player.withdrawFrom(std::move(std::get<0>(sellArguments)),
                  {std::get<1>(sellArguments), std::get<2>(sellArguments)});
}

void ClientHandler::_processDeposit(std::vector<char> &data) {
    msgpack::type::tuple<std::string, int32_t, int32_t> depositArguments;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(depositArguments);
```

```cpp
    player.depositTo(std::move(std::get<0>(depositArguments)),
                     {std::get<1>(depositArguments), std::get<2>(depositArgum
ents)});
}

void ClientHandler::_processMeditate(std::vector<char> &data) {
    player.meditate();
}

void ClientHandler::_processResurrect(std::vector<char> &data) {
    msgpack::type::tuple<int32_t, int32_t> resurrectArguments;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(resurrectArguments);
    player.requesResurrect({std::get<0>(resurrectArguments),
                                    std::get<1>(resurrectArguments)});
}

void ClientHandler::_processMessage(std::vector<char> &data) {
    msgpack::type::tuple<std::string, std::string> messageArguments;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(messageArguments);
    player.messageOtherPlayer(std::move(std::get<0>(messageArguments)),
                        std::move(std::get<1>(messageArguments)));
}

void ClientHandler::_processHeal(std::vector<char> &data) {
    msgpack::type::tuple<int32_t, int32_t> healArguments;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(healArguments);
    player.requestHeal({std::get<0>(healArguments), std::get<1>(healArguments)})
;
}


void ClientHandler::_processInventoryNames(std::vector<char> &data) {
    player.getInventoryNames();
}


void ClientHandler::_processStartMoving(std::vector<char> &data) {
    msgpack::type::tuple<GameType::Direction> moveInfo;
    handler = msgpack::unpack(data.data(), data.size(), offset);
    handler->convert(moveInfo);
    player.startMoving(std::get<0>(moveInfo));
}


void ClientHandler::_processStopMoving(std::vector<char> &data) {
    player.stopMoving();
}
```

```cpp
//
// Created by marcos on 6/24/20.
//

#ifndef ARGENTUM_CLIENTACCEPTER_H
#define ARGENTUM_CLIENTACCEPTER_H

#include <list>
#include <memory>
#include "../../libs/Thread.h"
#include <atomic>
#include <msgpack.hpp>
#include "../Entities/PlayerProxy.h"
#include "PlayerData.hpp"

class ServerProtocol;
class Socket;
class ClientHandler;
class ClientsMonitor;
class PlayerManager;

class ClientAccepter : public Thread {
private:
    ClientsMonitor& clients;
    ServerProtocol& protocol;
    Socket& serverSocket;
    std::atomic<bool>& keepRunning;
    PlayerManager& manager;
    msgpack::object_handle handler;

//Clase que se encarga de aceptar los nuevos clientes que intentan conectarse
public:
    ClientAccepter(ClientsMonitor& _clients, ServerProtocol& _protocol,
                   Socket& _serverSocket, std::atomic<bool>& _keepRunning,
                   PlayerManager& _manager) :
                    clients(_clients), protocol(_protocol),
                    serverSocket(_serverSocket), keepRunning(_keepRunning),
                    manager(_manager) {}

    //Comienza a ejecutar el thread aceptador
    void run() override;

private:
    PlayerData _receivePlayerInfo(Socket& clientSocket);
    PlayerData _createPlayer(std::vector<char>& buffer, std::size_t& offset);
    PlayerData _loadPlayer(std::vector<char>& buffer, std::size_t& offset);
    void _acceptClients();
    static bool _sendResponseToClient(Socket& clientSocket, GameType::Connection
Response status);
};


#endif //ARGENTUM_CLIENTACCEPTER_H
```

```cpp
//
// Created by marcos on 6/24/20.
//

#include "ClientAccepter.h"
#include "../../libs/Socket.h"
#include "ClientHandler.h"
#include "ClientsMonitor.h"
#include <iostream>
#include "PlayerManager.h"
#include "../Exceptions/UnavailablePlayerException.h"
#include "../Exceptions/InexistentPlayerException.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)
MSGPACK_ADD_ENUM(GameType::Class)
MSGPACK_ADD_ENUM(GameType::Race)

void ClientAccepter::_acceptClients() {
    GameType::ConnectionResponse status;
    while (keepRunning) {
        Socket clientSocket = serverSocket.accept();
        try {
            PlayerData playerData = _receivePlayerInfo(clientSocket);
            status = GameType::ACCEPTED;
            if (_sendResponseToClient(clientSocket, status)) {
                clients.pushToWaitingList(std::move(clientSocket), protocol, std
::move(playerData));
            }
        } catch(InexistentPlayerException& e) {
            status = GameType::INEXISTENT_PLAYER;
            _sendResponseToClient(clientSocket, status);
        } catch (UnavailablePlayerException& e) {
            status = GameType::UNAVAILABLE_PLAYER;
            _sendResponseToClient(clientSocket, status);
        } catch(std::exception& e) {
            std::cerr << e.what() << " in accepter" << std::endl;
            status = GameType::UNKOWN_SERVER_ERROR;
            _sendResponseToClient(clientSocket, status);
        } catch(...) {
            std::cerr << "Uknown error while reading a client player information in accepter!" << std
::endl;
            status = GameType::UNKOWN_SERVER_ERROR;
            _sendResponseToClient(clientSocket, status);
        }
    }
}

bool ClientAccepter::_sendResponseToClient(Socket& clientSocket, GameType::Conne
ctionResponse status) {
    status = static_cast<GameType::ConnectionResponse>(htonl(status));
    try {
        clientSocket.send(reinterpret_cast<char*>(&status), sizeof(status));
    } catch(...) {
        std::cerr << "Client disconnected suddenly in accepter" << std::endl;
        return false;
    }
    return true;
}

void ClientAccepter::run() {
    try {
        _acceptClients();
    } catch (std::exception& e) {
        std::cerr << e.what() << " in accepter socket" << std::endl;
    } catch (...) {
        std::cerr << "Unkown error in accepter socket" << std::endl;
    }
    keepRunning = false;
}

PlayerData ClientAccepter::_receivePlayerInfo(Socket& clientSocket) {
    std::size_t offset = 0;
    std::vector<char> buffer;
    uint32_t msgLen;
    clientSocket.receive(reinterpret_cast<char*>(&msgLen), sizeof(uint32_t));
    msgLen = ntohl(msgLen);
    buffer.clear();
    buffer.resize(msgLen);
    clientSocket.receive(buffer.data(), msgLen);
    msgpack::type::tuple<GameType::PlayerEvent> creationID;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(creationID);
    if (std::get<0>(creationID) ≡ GameType::CREATE_PLAYER) {
        return _createPlayer(buffer, offset);
    } else if (std::get<0>(creationID) ≡ GameType::LOAD_PLAYER) {
        return _loadPlayer(buffer, offset);
    } else {
        throw TPException("Invalidad load/create client messages!");
    }
}

PlayerData ClientAccepter::_createPlayer(std::vector<char>& buffer, std::size_t&
 offset) {
    msgpack::type::tuple<std::string, GameType::Race, GameType::Class> info;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(info);
    PlayerData playerData = {std::move(std::get<0>(info)),
                             std::get<1>(info), std::get<2>(info)};
    manager.storeNewPlayer(playerData);
    return playerData;
}

PlayerData ClientAccepter::_loadPlayer(std::vector<char>& buffer, std::size_t& o
ffset) {
    msgpack::type::tuple<std::string> nickname;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(nickname);
    return manager.getSavedPlayerData(std::get<0>(nickname));
}
```

```
#ifndef TP3TALLER_TP3SERVERSIDE_H
#define TP3TALLER_TP3SERVERSIDE_H

/*Esta es la clase principal del lado del ArgentumServer. Se encarga de validar
los
 * datos de ejecucion del ArgentumServer y, en caso de ser validos, instanciar a
l
 * ArgentumServer. Debe utilizarse esta clase para ejecutar el server */

class ArgentumServerSide {
public:
    static int run(int argc, char** argv);
};


#endif //TP3TALLER_TP3SERVERSIDE_H
```

```cpp
#include <iostream>
#include "ArgentumServerSide.h"
#include "ArgentumServer.h"
#include "../Config/Configuration.h"

#define INVALID_ARGUMENTS_MESSAGE "Error: argumentos invalidos."
//#define ARGUMENT_AMOUNT 2
#define ERROR 1
#define SUCCESS 0
//#define PORT_ARG_INDEX 1
//#define MAP_PATH_ARG_INDEX 1

int ArgentumServerSide::run(int argc, char** argv) {
    /*
    if (argc != ARGUMENT_AMOUNT) {
        std::cerr << INVALID_ARGUMENTS_MESSAGE << std::endl;
        return ERROR;
    }
    */
    try {
        ArgentumServer server;
        Configuration& config = Configuration::getInstance();
        server.connect(config.configPort(), config.configMapPath());
    } catch(std::exception& e) {
        std::cerr << e.what() << std::endl;
        return ERROR;
    }
    return SUCCESS;
}
```

```cpp
#ifndef TP3_SERVER_H
#define TP3_SERVER_H

/*Esta clase se encarga de manejar la
 * logica de las conexiones pero no de la comunicacion con los clientes, de eso
 * se encarga el Client Handler*/

#include "../../libs/Socket.h"
#include <string>
#include <atomic>
#include <vector>
#include "ClientHandler.h"
#include <memory>
#include <utility>
#include "../Game/Game.h"
#include "ServerProtocol.h"
#include "ClientsMonitor.h"

class ArgentumServer {
private:
    std::atomic<bool> keepRunning{true};
    Socket socket;

public:
    explicit ArgentumServer();
    ArgentumServer(const ArgentumServer&) = delete; /*Borro los constructores po
r copia*/
    ArgentumServer operator=(const ArgentumServer&) = delete;

    /*Levanta el servidor en el puerto pedido en el constructor*/
    /*Levanta el servidor en el puerto recibido*/
    void connect(const std::string& _port, const std::string& mapFilePath);

    /*Fuerza el cierre del servidor*/
    void finish();

private:
    void _execute(const std::string& mapFilePath);
};

#endif //TP3_SERVER_H
```

```cpp
#include <netdb.h>
#include "ArgentumServer.h"
#include "ServerMonitor.h"
#include "ClientAccepter.h"
#include "../Config/MapFileReader.h"
#include "PlayerManager.h"
#include "../Config/Configuration.h"
#include <unistd.h>
#include <iostream>

const double  FRAME_TIME = 1/60.f; /*ms que tarda en actualizarse el juego*/
const double TIME_FOR_CLIENTS_INITIALIZATION = 3; //ms dejados para mandarle la
data inicial a los clientes
const double BACKUP_TIME = 5*60; /*5 minutos*/

using namespace std::chrono;

const int MAX_LISTENERS = 10;

void ArgentumServer::finish() {
    keepRunning = false;
    socket.close();
}

void ArgentumServer::connect(const std::string& _port, const std::string& mapFil
ePath) {
    socket.bind(_port);
    socket.maxListen(MAX_LISTENERS);
    _execute(mapFilePath);
}

void ArgentumServer::_execute(const std::string& mapFilePath) {
    Timer timeBetweenUpdates, timeBetweenBackups;
    Game game((MapFileReader(mapFilePath)));
    ServerProtocol protocol(game);
    Configuration& config = Configuration::getInstance();
    PlayerManager manager(game, protocol, config.configIndexPath(), config.confi
gSavePath());
    ClientsMonitor clients(manager);
    ServerMonitor monitor(*this);
    monitor(); /*Espera la q para cerrar el server*/
    ClientAccepter accepter(clients, protocol, socket, keepRunning, manager);
    accepter(); /*Acepta conexiones de clientes*/

    timeBetweenBackups.start();

    try {
        double lastFrameTime = 0;
        double lastBackupTime;
        while (keepRunning) {
            timeBetweenUpdates.start();
            clients.removeDisconnectedClients(protocol);
            clients.mergeClientsEvents();
            game.update(lastFrameTime, protocol);
            protocol.buildGeneralDataBuffer();
            clients.sendGameUpdate();

            lastFrameTime = timeBetweenUpdates.getTime();
            lastBackupTime = timeBetweenBackups.getTime();
            if (lastBackupTime / 1000 ≥ BACKUP_TIME) {
                clients.backup();
                timeBetweenBackups.start();
                std::cout << "Backuped players. Next backup in 5 minutes" << std::endl;
            }
            if (clients.hasWaitingClients() ∧
                (FRAME_TIME*1000 − lastFrameTime) > TIME_FOR_CLIENTS_INITIALIZAT
```

```cpp
ION) {
                clients.mergeWaitingClients(game, protocol);
            }
            lastFrameTime = timeBetweenUpdates.getTime();
            if (lastFrameTime < FRAME_TIME*1000) {
                usleep((FRAME_TIME*1000 − lastFrameTime) * 1000);
                lastFrameTime = FRAME_TIME*1000;
            }
        }

    } catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    } catch (...) {
        std::cerr << "Uknown error in Main Game Loop!" << std::endl;
    }

    try {
        if (monitor.closeRequest()) {
            clients.backup();
        }
    } catch (std::exception& e) {
        std::cerr << e.what()  << "while backing up players!" << std::endl;
    } catch (...) {
        std::cerr << "Uknown error while backing up players!" << std::endl;
    }

    finish();
    monitor.join(); /*Joineamos los threads*/
    clients.join();
    accepter.join();
}

ArgentumServer::ArgentumServer() = default;
```

```
//
// Created by marcos on 9/7/20.
//

#ifndef ARGENTUM_SAVEFILEMANAGER_H
#define ARGENTUM_SAVEFILEMANAGER_H

#include "PlayerIndexFile.h"
#include "PlayerSaveFile.h"
#include <mutex>

/*Esta clase se encarga de administrar la persistencia del servidor. Maneja los
 * 2 archivos (indice y save file)*/

class SaveFileManager {
private:
    PlayerIndexFile indexFile;
    PlayerSaveFile saveFile;
    std::mutex m;

public:
    SaveFileManager(const std::string& indexPath, const std::string& savePath) :
                    indexFile(indexPath), saveFile(savePath) {}

    /*Retorna la data almacenada del player. Si el player no existe tira excepti
on*/
    PlayerData getPlayerData(const std::string& playerNickname);

    /*Almacena la data del player recibida en el correspondiente en el archivo.
     * Si el player no existia en el archivo tira exception*/
    void storeOldPlayer(const PlayerData& data);

    /*Almacena la data del player recibida, agregando la entrada correspondiente
     * en ambos archivos. Si el player ya existia tira exception*/
    void storeNewPlayer(const PlayerData& data);
};


#endif //ARGENTUM_SAVEFILEMANAGER_H
```

```
//
// Created by marcos on 9/7/20.
//

#include "SaveFileManager.h"
#include "../Exceptions/UnavailablePlayerException.h"

PlayerData SaveFileManager::getPlayerData(const std::string &playerNickname) {
    std::lock_guard<std::mutex> l(m);
    PlayerFilePosition filePosition = indexFile.getPlayerPosition(playerNickname
);
    return saveFile.getPlayerData(playerNickname, filePosition);
}

void SaveFileManager::storeNewPlayer(const PlayerData &data) {
    std::lock_guard<std::mutex> l(m);
    if (indexFile.playerExists(data.nickname)) {
        throw UnavailablePlayerException();
    }
    PlayerFilePosition filePosition = saveFile.storePlayerData(data);
    indexFile.storeNewPlayer(data.nickname, filePosition);
}

void SaveFileManager::storeOldPlayer(const PlayerData &data) {
    std::lock_guard<std::mutex> l(m);
    PlayerFilePosition filePosition = indexFile.getPlayerPosition(data.nickname)
;
    filePosition = saveFile.storePlayerData(data, filePosition.offset);
    indexFile.storeOldPlayer(data.nickname, filePosition);
}
```

```
//
// Created by marcos on 7/8/20.
//

#ifndef ARGENTUM_PLAYERSAVEFILE_H
#define ARGENTUM_PLAYERSAVEFILE_H

#include <fstream>
#include "PlayerFilePosition.h"
#include "../Server/PlayerData.hpp"
#include <vector>

/*Esta clase es la que maneja el archivo que contiene los datos del player
 * guardado*/
class PlayerSaveFile {
private:
    std::fstream saveFile;
    msgpack::object_handle handler;
    std::size_t readData{0};

public:
    explicit PlayerSaveFile(const std::string& filePath);

    /*Retorna la data guardada del player (stats, inventario, banker items, etc)
*/
    PlayerData getPlayerData(const std::string& playerNickname,
                             PlayerFilePosition filePosition);

    /*Almacena la informacion del player provisto en la posicion especificada
     * en el offset*/
    PlayerFilePosition storePlayerData(const PlayerData& playerData,
                                       int32_t fileOffset);

    /*Sobrecarga para storePlayerData, esta funcion se usa para almacenar un nue
vo
     * player en el archivo (no recibe offset porque siempre ira al final del ar
chivo*/
    PlayerFilePosition storePlayerData(const PlayerData& playerData);

private:
    static void _packPlayerType(std::stringstream& dataToStore, const PlayerData
& playerData);
    static void _packPlayerGeneralStats(std::stringstream& dataToStore,
                                        const PlayerData& playerData);
    static void _packPlayerInventory(std::stringstream& dataToStore,
                                     const PlayerData& playerData);
    void _loadPlayerType(PlayerData& playerData,
                         std::vector<char>& playerDataBuffer);
    void _loadPlayerGeneralStats(PlayerData& playerData,
                                 std::vector<char>& playerDataBu
ffer);
    void _loadPlayerInventory(PlayerData& playerData, std::vector<char>& playerD
ataBuffer);
    static void _packBankItems(std::stringstream& dataToStore, const PlayerData&
 playerData);
    void _loadPlayerBank(PlayerData& playerData, std::vector<char>& playerDataBu
ffer);
};


#endif //ARGENTUM_PLAYERSAVEFILE_H
```

```
//
// Created by marcos on 7/8/20.
//

#include "PlayerSaveFile.h"
#include <iostream>
#include "../../libs/TPException.h"

MSGPACK_ADD_ENUM(GameType::Race)
MSGPACK_ADD_ENUM(GameType::Class)
MSGPACK_ADD_ENUM(GameType::ItemType)
MSGPACK_ADD_ENUM(GameType::EquipmentPlace)

PlayerSaveFile::PlayerSaveFile(const std::string &filePath) {
    saveFile.open(filePath, std::ios::in | std::ios::out | std::ios::binary);
    if (¬saveFile.is_open()) {
        std::cout << "Could not find a Save File with the provided name."
                     " Creating one now." << std::endl;
        std::ofstream newSaveFile(filePath);
        newSaveFile.close();
        saveFile.open(filePath, std::ios::in | std::ios::out | std::ios::binary)
;
    }
}

PlayerData PlayerSaveFile::getPlayerData(const std::string& playerNickname,
                                         PlayerFilePosition filePosition) {
    saveFile.clear();
    saveFile.seekg(filePosition.offset, std::ios_base::beg);
    readData = 0;
    std::vector<char> playerDataBuffer(filePosition.length);
    saveFile.read(playerDataBuffer.data(), filePosition.length);
    PlayerData playerData;
    _loadPlayerType(playerData, playerDataBuffer);
    if (playerData.nickname ≠ playerNickname) {
        throw TPException("Stored player’s nickname doesnt match the one provided!");
    }
    _loadPlayerGeneralStats(playerData, playerDataBuffer);
    _loadPlayerInventory(playerData, playerDataBuffer);
    _loadPlayerBank(playerData, playerDataBuffer);
    return playerData;
}

void PlayerSaveFile::_loadPlayerBank(PlayerData& playerData,
                                     std::vector<char>& playerDataBuffer) {

    for (auto & currItem : playerData.bankerItems) {
        msgpack::type::tuple<GameType::ItemType, int32_t> item;
        handler = msgpack::unpack(playerDataBuffer.data(), playerData.size(), re
adData);
        handler→convert(item);
        currItem = item;
    }
    msgpack::type::tuple<int32_t> bankGold;
    handler = msgpack::unpack(playerDataBuffer.data(), playerData.size(), readDa
ta);
    handler→convert(bankGold);
    playerData.bankerGold = std::get<0>(bankGold);
}

void PlayerSaveFile::_loadPlayerInventory(PlayerData& playerData,
                                          std::vector<char>& playerDataBuffer
) {

    for (auto & currItem : playerData.inventory) {
        msgpack::type::tuple<GameType::ItemType, int32_t> item;
```

```cpp
        handler = msgpack::unpack(playerDataBuffer.data(), playerData.size(), readData);
        handler→convert(item);
        currItem = item;
    }
    msgpack::type::tuple<int32_t> equipment;
    handler = msgpack::unpack(playerDataBuffer.data(), playerData.size(), readData);
    handler→convert(equipment);
    playerData.equipment.at(GameType::EQUIPMENT_PLACE_HEAD) = std::get<0>(equipment);
    handler = msgpack::unpack(playerDataBuffer.data(), playerData.size(), readData);
    handler→convert(equipment);
    playerData.equipment.at(GameType::EQUIPMENT_PLACE_CHEST) = std::get<0>(equipment);
    handler = msgpack::unpack(playerDataBuffer.data(), playerData.size(), readData);
    handler→convert(equipment);
    playerData.equipment.at(GameType::EQUIPMENT_PLACE_SHIELD) = std::get<0>(equipment);
    handler = msgpack::unpack(playerDataBuffer.data(), playerData.size(), readData);
    handler→convert(equipment);
    playerData.equipment.at(GameType::EQUIPMENT_PLACE_WEAPON) = std::get<0>(equipment);
}

void PlayerSaveFile::_loadPlayerGeneralStats(PlayerData& playerData,
                                     std::vector<char>& playerDataBuffer) {
    msgpack::type::tuple<int32_t, int32_t, int32_t> generalStats;
    handler = msgpack::unpack(playerDataBuffer.data(), playerData.size(), readData);
    handler→convert(generalStats);
    playerData.level = std::get<0>(generalStats);
    playerData.experience = std::get<1>(generalStats);
    playerData.gold = std::get<2>(generalStats);
    msgpack::type::tuple<int32_t, int32_t, int32_t, int32_t> stats;
    handler = msgpack::unpack(playerDataBuffer.data(), playerData.size(), readData);
    handler→convert(stats);
    playerData.constitution = std::get<0>(stats);
    playerData.strength = std::get<1>(stats);
    playerData.agility = std::get<2>(stats);
    playerData.intelligence = std::get<3>(stats);
}

void PlayerSaveFile::_loadPlayerType(PlayerData& playerData,
                                        std::vector<char>& playerDataBuffer) {
    msgpack::type::tuple<bool> isNewPlayer;
    handler = msgpack::unpack(playerDataBuffer.data(), playerData.size(), readData);
    handler→convert(isNewPlayer);
    playerData.isNewPlayer = std::get<0>(isNewPlayer);
    msgpack::type::tuple<std::string, GameType::Race, GameType::Class> playerType;
    handler = msgpack::unpack(playerDataBuffer.data(), playerData.size(), readData);
    handler→convert(playerType);
    playerData.nickname = std::move(std::get<0>(playerType));
    playerData.pRace = std::get<1>(playerType);
    playerData.pClass = std::get<2>(playerType);
}

PlayerFilePosition PlayerSaveFile::storePlayerData(const PlayerData& playerData,
                                            int32_t fileOffset) {
```

```cpp
    saveFile.clear();
    PlayerFilePosition playerPosition{};
    playerPosition.offset = fileOffset;
    std::stringstream dataToStore;
    _packPlayerType(dataToStore, playerData);
    _packPlayerGeneralStats(dataToStore, playerData);
    _packPlayerInventory(dataToStore, playerData);
    _packBankItems(dataToStore, playerData);
    std::string strDataToStore = dataToStore.str();
    playerPosition.length = strDataToStore.size();
    saveFile.seekp(fileOffset, std::ios_base::beg);
    saveFile.write(strDataToStore.data(), playerPosition.length);
    int32_t paddingSize = (playerData.size() − playerPosition.length);
    std::vector<char> paddingBuffer(paddingSize, 0);
    saveFile.write(paddingBuffer.data(), paddingSize);
    saveFile.sync();
    return playerPosition;
}

void PlayerSaveFile::_packBankItems(std::stringstream& dataToStore,
                                    const PlayerData& playerData) {
    for (auto & currItem : playerData.bankerItems) {
        msgpack::type::tuple<GameType::ItemType, int32_t> item(currItem);
        msgpack::pack(dataToStore, item);
    }
    msgpack::type::tuple<int32_t> gold(playerData.bankerGold);
    msgpack::pack(dataToStore, gold);
}

void PlayerSaveFile::_packPlayerType(std::stringstream& dataToStore,
                                    const PlayerData& playerData) {
    msgpack::type::tuple<bool> isNewPlayer(playerData.isNewPlayer);
    msgpack::pack(dataToStore, isNewPlayer);
    msgpack::type::tuple<std::string, GameType::Race, GameType::Class> playerType(
                        playerData.nickname, playerData.pRace, playerData.pClass);
    msgpack::pack(dataToStore, playerType);
}

void PlayerSaveFile::_packPlayerGeneralStats(std::stringstream& dataToStore,
                                    const PlayerData& playerData) {
    msgpack::type::tuple<int32_t, int32_t, int32_t> generalStats(playerData.level,
                            playerData.experience, playerData.gold);
    msgpack::pack(dataToStore, generalStats);
    msgpack::type::tuple<int32_t, int32_t, int32_t, int32_t> stats(playerData.constitution,
                            playerData.strength, playerData.agility, playerData.intelligence);
    msgpack::pack(dataToStore, stats);
}

void PlayerSaveFile::_packPlayerInventory(std::stringstream& dataToStore,
                                    const PlayerData& playerData) {
    for (auto & currItem : playerData.inventory) {
        msgpack::type::tuple<GameType::ItemType, int32_t> item(currItem);
        msgpack::pack(dataToStore, item);
    }
    msgpack::type::tuple<int32_t> helmet(playerData.equipment.at(GameType::EQUIPMENT_PLACE_HEAD));
    msgpack::pack(dataToStore, helmet);
    msgpack::type::tuple<int32_t> chest(playerData.equipment.at(GameType::EQUIPMENT_PLACE_CHEST));
    msgpack::pack(dataToStore, chest);
    msgpack::type::tuple<int32_t> shield(playerData.equipment.at(GameType::EQUIP
MENT_PLACE_SHIELD));
    msgpack::pack(dataToStore, shield);
    msgpack::type::tuple<int32_t> weapon(playerData.equipment.at(GameType::EQUIP
MENT_PLACE_WEAPON));
    msgpack::pack(dataToStore, weapon);
}

PlayerFilePosition PlayerSaveFile::storePlayerData(const PlayerData &playerData)
 {
    saveFile.seekp(0, std::ios_base::end);
    return storePlayerData(playerData, saveFile.tellp());
}
```

```
//
// Created by marcos on 7/8/20.
//

#ifndef ARGENTUM_PLAYERINDEXFILE_H
#define ARGENTUM_PLAYERINDEXFILE_H

#include <fstream>
#include <unordered_map>
#include "PlayerFilePosition.h"

/*Esta clase maneja el archivo de indice de jugadores, el cual contiene
 * el nombre del jugador, el offset donde arranca su informacion y la cantidad d
e
 * bytes que su informacion ocupa (para saber cuanto leer con msgpack)*/
class PlayerIndexFile {
private:
    std::fstream indexFile;
    std::unordered_map<std::string, PlayerFilePosition> players;
    std::unordered_map<std::string, int32_t> indexPlayersPosition; /*Guardo la p
osicion en el indice del largo en bytes*/
                                                                   /*de la info
 del player para poder acceder rapido cada vez que cambie*/

public:
    explicit PlayerIndexFile(const std::string& filePath);

    /*Almacena los datos actualizados de un player player ya estaba almacenado e
n el archivo*/
    void storeOldPlayer(const std::string& playerNickname, PlayerFilePosition fi
lePosition);

    /*Almacena en el archivo los datos de un player que acaba de ser creado*/
    void storeNewPlayer(const std::string& playerNickname, PlayerFilePosition fi
lePosition);

    /*Retorna la posicion del player en el archivo que contiene todos sus datos*
/
    PlayerFilePosition getPlayerPosition(const std::string& nickname);

    /*Retorna true si el player existe en el archivo de datos, false en caso con
trario*/
    bool playerExists(const std::string& nickname) const;

private:
    void _loadFileData();
};


#endif //ARGENTUM_PLAYERINDEXFILE_H
```

```
//
// Created by marcos on 7/8/20.
//

#include "PlayerIndexFile.h"
#include <iostream>
#include <arpa/inet.h>
#include <vector>
#include "../../libs/TPException.h"
#include "../Exceptions/InexistentPlayerException.h"

PlayerIndexFile::PlayerIndexFile(const std::string& filePath) {
    indexFile.open(filePath, std::ios::in | std::ios::out | std::ios::binary);
    if (¬indexFile.is_open()) {
        std::cout << "Could not find an Index File with the provided name."
                     " Creating one now." << std::endl;
        std::ofstream newIndexFile(filePath);
        newIndexFile.close();
        indexFile.open(filePath, std::ios::in | std::ios::out | std::ios::binary);
    } else {
        _loadFileData();
    }
}

void PlayerIndexFile::_loadFileData() {
    PlayerFilePosition offData{};
    uint32_t nicknameSize = 0;
    indexFile.seekg(0, std::ios_base::beg);
    while (indexFile.peek() ≠ std::fstream::traits_type::eof() ∨ ¬indexFile.eof()) {
        indexFile.read(reinterpret_cast<char*>(&nicknameSize), sizeof(nicknameSize));
        nicknameSize = ntohl(nicknameSize);
        std::vector<char> playerNickname(nicknameSize);
        indexFile.read(playerNickname.data(), nicknameSize);
        indexFile.read(reinterpret_cast<char*>(&offData.offset), sizeof(offData.offset));
        indexPlayersPosition.emplace(playerNickname.data(), indexFile.tellg());
        indexFile.read(reinterpret_cast<char*>(&offData.length), sizeof(offData.length));
        offData.offset = ntohl(offData.offset);
        offData.length = ntohl(offData.length);
        players.emplace(playerNickname.data(), offData);
    }
}

void PlayerIndexFile::storeOldPlayer(const std::string& playerNickname, PlayerFilePosition filePosition) {
    indexFile.clear();
    if (players.count(playerNickname) ≡ 1) {
        players.at(playerNickname) = filePosition;
        filePosition = {htonl(filePosition.offset), htonl(filePosition.length)};
        indexFile.seekp(indexPlayersPosition.at(playerNickname), std::ios_base::beg);
        indexFile.write(reinterpret_cast<char*>(&filePosition.length), sizeof(filePosition.length));
        indexFile.sync();
    } else {
        throw TPException("Tried to store a logged in player that didnt exist!");
    }
}

PlayerFilePosition PlayerIndexFile::getPlayerPosition(const std::string& nickname) {
    if (playerExists(nickname)) {
```

```
        return players.at(nickname);
    }
    throw InexistentPlayerException();
}

void PlayerIndexFile::storeNewPlayer(const std::string &playerNickname,
                                     PlayerFilePosition filePosition) {

    players.emplace(playerNickname, filePosition);
    indexFile.clear();
    indexFile.seekp(0, std::ios_base::end);
    filePosition = {htonl(filePosition.offset), htonl(filePosition.length)};
    uint32_t nameLength = playerNickname.size() + 1;
    nameLength = htonl(nameLength);
    indexFile.write(reinterpret_cast<char*>(&nameLength), sizeof(nameLength));
    indexFile.write(playerNickname.data(), playerNickname.size() + 1);
    indexFile.write(reinterpret_cast<char*>(&filePosition.offset), sizeof(filePosition.offset));
    indexPlayersPosition.emplace(playerNickname, indexFile.tellp());
    indexFile.write(reinterpret_cast<char*>(&filePosition.length), sizeof(filePosition.length));
    indexFile.sync();
}

bool PlayerIndexFile::playerExists(const std::string& nickname) const {
    return (players.count(nickname) ≡ 1);
}
```

```
//
// Created by marcos on 7/8/20.
//

#ifndef ARGENTUM_PLAYERFILEPOSITION_H
#define ARGENTUM_PLAYERFILEPOSITION_H

/*Este struct se usa para saber el offset de un player en el archivo (donde comi
enza)
 * y la longitud que ocupan sus datos en este*/

struct PlayerFilePosition {
    uint32_t offset;
    uint32_t length;
};

#endif //ARGENTUM_PLAYERFILEPOSITION_H
```

```
//
// Created by agustin on 6/6/20.
//

#ifndef ARGENTUM_TILE_H
#define ARGENTUM_TILE_H

#include <memory>
#include <list>
#include "../../libs/GameEnums.h"
#include "../Entities/Entity.h"
#include "../Items/ItemData.h"

class MapTests;
class Item;
struct AttackResult;
struct ProductData;
class Player;

class Tile {
private:
    std::shared_ptr<Entity> entity;
    std::list<std::shared_ptr<Item>> items;
    bool isOccupable{};
    bool isFromCity;
    GameType::FloorType floor;
    GameType::Structure structure{GameType::Structure::TREE};

    friend MapTests;

private:
    void _storeItemsNames(Player& player);

public:
    explicit Tile(bool isOccupable, bool isFromCity, GameType::FloorType floor,
                  GameType::Structure structure, std::shared_ptr<Entity>∧ initi
alEntity);

    //El tile se queda con la entity de other y setea la de other en nullptr,
    //actualizando tambien el estado de si es ocupable o no en ambos tiles
    void moveEntity(Tile& otherTile, Coordinate position);

    //Intenta agregar la entity al tile
    //Si la posicion es ocupable entonces se apropia del puntero
    void addEntity(std::shared_ptr<Entity>∧ received_entity);

    //Elimina la entity guardada, habilita la ocupacion del tile por otra
    //entity
    void removeEntity();

    //Intenta agregar el item al tile, sumandolo a los items ya guardados
    void addItem(std::shared_ptr<Item>∧ received_item);

    //Agrega los items recibidos en la lista a los items que contiene el tile
    void addItem(std::list<std::shared_ptr<Item>>∧ _items);

    //Elimina uno de los items que se encuentran en el tile y lo retorna}
    //Si no hay ningun item retorna nullptr
    std::shared_ptr<Item> removeItem();

    //Ataca la entidad que se encuentre guardada en el mapa
    //Retorna la cantidad de daÃ±o que recibio la entidad atacada, si no hay
    //una entidad retorna 0, el booleano indica si se realizo o no el ataque al
tile,
    //valiendo true si se realizo, false en otro caso (que no haya un entity no
hace
```

```cpp
    //necesariamente que sea false)
    std::pair<AttackResult, bool> attacked(int damage, unsigned int level, bool
isAPlayer);

    //Retorna true si almacena un entity que es un target de un monster
    bool hasMonsterTarget() const;

    //Indica si el tile puede ser ocupado por una entity
    //Retorna true si puede, false en otro caso
    bool isAvailable() const;

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    //void list(Player& player, std::list<ProductData>& products);
    void list(Player& player);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void withdraw(Player& player, const std::string& itemName);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void deposit(Player& player, const std::string& itemName);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void buy(Player& player, const std::string& itemName);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void sell(Player& player, const std::string& itemName);

    //Retorna si es de una city
    bool isInCity() const;

    //Guarda en el buffer el entity almacenado en el tile, junto con el tipo de
    //piso y la estructura almacenadas
    void operator>>(std::stringstream & mapBuffer) const;

    //Retorna un puntero al item que se debe mostrar en el mapa
    const Item* peekShowedItemData();

    //Le pide al entity guardado en el tile que restaure la vida y el mana del
    //player recibido, si no hay un entity entonces no hace nada
    void requestRestore(Player& player);
};


#endif //ARGENTUM_TILE_H
```

```cpp
//
// Created by agustin on 6/6/20.
//

#include "Tile.h"
#include <memory>
#include "../Entities/AttackResult.h"
#include "../../libs/TPException.h"
#include "../Items/Item.h"
#include "../Entities/Player.h"
#include <msgpack.hpp>

MSGPACK_ADD_ENUM(GameType::FloorType)
MSGPACK_ADD_ENUM(GameType::Entity)
MSGPACK_ADD_ENUM(GameType::Structure)

#define NO_ITEMS_MESSAGE "There are no entities or items on this tille\n"
#define ITEMS_MESSAGE "The stored items are:\n"

/////////////////////////////////////PUBLIC/////////////////////////////////
//////

Tile::Tile(bool _isOccupable, bool _isFromCity, GameType::FloorType _floor, Game
Type::Structure _structure,
           std::shared_ptr<Entity>&& initialEntity): entity(nullptr) {
    floor = _floor;
    structure = _structure;
    isFromCity = _isFromCity;
    isOccupable = _isOccupable;

    if (isOccupable && initialEntity) {
        entity = std::move(initialEntity);
        isOccupable = false;
    }
}

void Tile::moveEntity(Tile& otherTile, Coordinate position) {
    this->entity = std::move(otherTile.entity);
    otherTile.entity = nullptr;
    entity->move(position);
    isOccupable = false;
    otherTile.isOccupable = true;
}

void Tile::addEntity(std::shared_ptr<Entity>&& received_entity) {
    if (isOccupable) {
        entity = std::move(received_entity);
        isOccupable = false;
    } else {
        throw TPException("Tried to add an entity to a non occupable tile!");
    }
}

void Tile::removeEntity() {
    if (entity) {
        entity.reset();
        isOccupable = true;
    }
}

void Tile::addItem(std::shared_ptr<Item>&& received_item) {
    if (received_item) {
        items.push_back(std::move(received_item));
    }
}

```

```cpp
std::shared_ptr<Item> Tile::removeItem() {
    if (items.empty()) {
        return nullptr;
    }
    std::shared_ptr<Item> return_item = std::move(items.back());
    items.pop_back();
    return return_item;
}

std::pair<AttackResult, bool> Tile::attacked(int damage, unsigned int level, boo
l isAPlayer) {
    if (isFromCity) {
        return {{0, 0, "You can’t attack a tile inside a city\n"}, false};
    }
    if (¬entity) {
        return {{0, 0, ""}, true};
    }
    return {entity→attacked(damage, level, isAPlayer), true};
}

bool Tile::hasMonsterTarget() const {
    if (entity) {
        return entity→isMonsterTarget();
    }
    return false;
}

bool Tile::isAvailable() const {
    return isOccupable;
}

void Tile::list(Player &player) {
    if (entity) {
        entity→list(player);
    } else {
        _storeItemsNames(player);
    }
}


void Tile::withdraw(Player &player, const std::string &itemName) {
    if (entity) {
        entity→withdraw(player, itemName);
    }
}

void Tile::deposit(Player &player, const std::string &itemName) {
    if (entity) {
        entity→deposit(player, itemName);
    }
}

void Tile::buy(Player &player, const std::string &itemName) {
    if (entity) {
        entity→buy(player, itemName);
    }
}

void Tile::sell(Player &player, const std::string &itemName) {
    if (entity) {
        entity→sell(player, itemName);
    }
}

void Tile::addItem(std::list<std::shared_ptr<Item>>∧ _items) {
    for (auto & item : _items) {
```

```cpp
        if (item) {
            items.push_back(std::move(item));
        }
    }
}

bool Tile::isInCity() const {
    return isFromCity;
}

void Tile::operator>>(std::stringstream &mapBuffer) const {
    GameType::Entity entityType = GameType::Entity::NOTHING;
    std::string nickname = "";
    if (entity) {
        entityType = entity→getType();
        nickname = entity→getNickname();
    }
    /*de izquierda a derecha es el tipo de piso, tipo de estructura y citizen*/
    msgpack::type::tuple<GameType::FloorType, GameType::Structure,
            GameType::Entity, std::string> tileInfo(floor, structure, entityType
, nickname);
    msgpack::pack(mapBuffer, tileInfo);
}

const Item* Tile::peekShowedItemData() {
    if (items.empty()) {
        return nullptr;
    }
    return items.back().get();
}

void Tile::requestRestore(Player& player) {
    if (entity) {
        entity→requestHeal(player);
    }
}

//////////////////////////PRIVATE////////////////////////////////////

void Tile::_storeItemsNames(Player& player) {
    if (¬items.empty()) {
        player.addMessage(ITEMS_MESSAGE);
        for (const auto & item: items) {
            player.addMessage(item→getName() + "\n");
        }
    }
}
```

```cpp
//
// Created by agustin on 10/6/20.
//

#ifndef ARGENTUM_POINTANDDISTANCE_H
#define ARGENTUM_POINTANDDISTANCE_H

#include "Coordinate.h"

//Este struct es auxiliar, solo deberia ser usado por el mapa para pathfinding,
//se declara en el header por tener que declarar la funcion privada
struct PointAndDistance {
    Coordinate point;
    unsigned int distance;
};


#endif //ARGENTUM_POINTANDDISTANCE_H
```

```cpp
//
// Created by agustin on 7/6/20.
//

#ifndef ARGENTUM_MAP_H
#define ARGENTUM_MAP_H

#include <vector>
#include <list>
#include <unordered_map>
#include <queue>
#include <memory>
#include "InverseCoordinateDistance.h"
#include "Tile.h"

struct AttackResult;
class Item;
class Entity;
class Monster;
class MapFileReader;
class EntityTests;
class MapTests;

class Map {
private:
    std::vector<std::vector<Tile>> tiles;

    friend EntityTests;
    friend MapTests;

private:
    void _storeAdjacentPositions(PointAndDistance refference,
            std::unordered_map<Coordinate, unsigned int>& distances,
            std::unordered_map<Coordinate, Coordinate>& parentsAndChilds,
            std::priority_queue<PointAndDistance, std::vector<PointAndDistance>,
                                InverseCoordinateDistance>& nodes, Coordinate de
stination) const;
    static unsigned int _getDistance(Coordinate a, Coordinate b);
    bool _isCoordinateValid(Coordinate coordinate) const;
    Coordinate _getValidCoordinate(Coordinate coordinate) const;
    static void _storePath(Coordinate initialPosition, Coordinate desiredPositio
n, const std::unordered_map<Coordinate,
                           Coordinate>& parentsAndChilds, std::list<Coordinate>&
 path);
    void _buildSearchRegion(Coordinate center, unsigned int range, Coordinate& t
opLeft, Coordinate& bottomRight) const;
    static bool _areCoordinatesEqual(Coordinate a, Coordinate b);
    static void _initializeConstructorMaps(std::unordered_map<std::string, GameT
ype::Entity>& entities,
                                           std::unordered_map<std::string, GameT
ype::Structure>& structures,
                                           std::unordered_map<std::string, GameT
ype::FloorType>& floors);
    bool _isReachable(Coordinate position) const;
    void _getTargets(Coordinate center, unsigned int range, std::vector<Coordina
te>& targets,
                     bool detectUnreachableTargets) const;

public:
    explicit Map(MapFileReader& mapFile, std::list<Coordinate>& priests);

    Map() = default; /*Crea mapa de 0x0, lo usamos para las pruebas*/

    //Ataca la tile y retorna el resultado del ataque, el booleano indica si el
ataque
    //fue realizado (true) o no (false)
    std::pair<AttackResult, bool> attackTile(int damage, unsigned int level, bool isAPlayer,
                            Coordinate coordinate);

    //Almacena en el vector la cantidad de targets de un monstruo en un cuadrado centrado en
    //center de lado 2*range+1
    void getMoveTargets(Coordinate center, unsigned int range, std::vector<Coordinate>& targets) const;

    //Almacena en el vector la cantidad de targets de un monstruo para atacar en un cuadrado centrado en
    //center de lado 2*range+1
    void getAttackTargets(Coordinate center, unsigned int range, std::vector<Coordinate>& targets) const;

    //Almacena en la lista el camino que se debe seguir para llegar a la coordenada deseada
    //Si existe un camino retorna true y la informacion es guardada en path, sino retorna
    //false y no guarda nada
    bool getPath(Coordinate currentPosition, Coordinate desiredPosition, std::list<Coordinate>& path) const;

    //Intenta agregar la entity al tile que se encuentra en la coordenada recibida apropiandose
    //del shared_ptr, si la coordenada es invalida tira invalid_argument y no se apropia del puntero
    //Si la posicion es ocupable entonces se apropia del puntero, sino tira TPException
    void addEntity(Coordinate position, std::shared_ptr<Entity>∧ entity);

    //Toma el primer item almacenado en el tile que se encuentra en la coordenada
    //pasada, lo elimina de la tile y lo retorna, si la coordenada es invalida se
    //tira una exception de invalid_argument
    //Si el tile no tiene items retorna un shared_ptr que almacena nullptr
    std::shared_ptr<Item> removeItem(Coordinate position);

    //Elimina la entity almacenada en la coordenada, liberando sus recursos y permitiendo
    //el almacenamiento de otra entity, si la coordenada es invalida tira invalid_argument
    void removeEntity(Coordinate position);

    //Intenta mover la entity de starting a final position, si finalPosition esta ocupada
    //entonces retorna false, sino retorna true
    //Si alguna de las coordenadas es invalida tira invalid_argument
    void moveEntity(Coordinate startingPosition, Coordinate finalPosition);

    //Retorna true si el lugar puede ser ocupado por una entity, false en caso contrario
    bool isPlaceAvailable(Coordinate position) const;

    //Agrega los items de la lista al tile al tile que se encuentra en la coordenada recibida apropiandose de la lista,
    //si la coordenada es invalida tira invalid_argument y no se apropia del puntero
    //Se pueden guardar smart_pointers que contengan nullptr, pero no se deberia hacer, el chequeo
    //de si se esta guardando null o no tiene que venir de afuera
    void addItemsToTile(std::list<std::shared_ptr<Item>>∧ items, Coordinate position);

```

```cpp
    //Agrega el item al tile que se encuentra en la coordenada recibida apropiandose del shared_ptr,
    //si la coordenada es invalida tira invalid_argument y no se apropia del puntero
    //Se pueden guardar smart_pointers que contengan nullptr, pero no se deberia hacer, el chequeo
    //de si se esta guardando null o no tiene que venir de afuera
    void addItemsToTile(std::shared_ptr<Item>∧ item, Coordinate position);

    //Retorna una coordenada aleatoria en la que puede ponerse un monstruo
    Coordinate getMonsterCoordinate();

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    //unsigned int list(Player& player, std::list<ProductData>& products, Coordinate coordinate);
    void list(Player& player, Coordinate coordinate);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void withdraw(Player& player, const std::string& itemName, Coordinate coordinate);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void deposit(Player& player, const std::string& itemName, Coordinate coordinate);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void buy(Player& player, const std::string& itemName, Coordinate coordinate);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void sell(Player& player, const std::string& itemName, Coordinate coordinate);

    //Guarda el estado actual del mapa para que pueda ser enviado a un nuevo cliente
    void operator>>(std::stringstream& mapBuffer) const;

    //Retorna un pair que almacena el tipo del item y su id, si la coordenada es inexistente
    //guarda -2 en el id (second), si el tile no tiene items guarda -1, sino guarda el it del item
    //std::pair<GameType::ItemType, int32_t> peekShowedItemData(Coordinate coordinate);
    const Item* peekShowedItemData(Coordinate coordinate);

    //Retorna una posicion disponible alrededor de la posicion recibida
    Coordinate getSpawnCoordinateArroundPosition(Coordinate refference);

    void requestRestore(Player& player, Coordinate target);

    //Retorna una coordenada aleatoria alrededor de refference que no este ocupada ni
    //pertenezca a una ciudad
    Coordinate getMonsterRandomPosition(Coordinate refference) const;

    virtual ~Map() = default; /*Para que FakeIt lo pueda mockear*/
};


#endif //ARGENTUM_MAP_H
```

```cpp
//
// Created by agustin on 7/6/20.
//

#include <queue>
#include <unordered_map>
#include <memory>
#include "Map.h"
#include "../Entities/AttackResult.h"
#include "../Config/Calculator.h"
#include "../Entities/Citizens/CitizenFactory.h"
#include "../Config/MapFileReader.h"
#include <msgpack.hpp>


#define RESPAWN_RANGE 3

/////////////////////////////PRIVATE/////////////////////////////

//Retorna la distancia (siempre positiva) entre las dos coordenadas
unsigned int Map::_getDistance(Coordinate a, Coordinate b) {
    return std::abs((a.iPosition − b.iPosition) + (a.jPosition − b.jPosition));
}


//Indica si la coordenada esta en el rango de posiciones del mapa
bool Map::_isCoordinateValid(Coordinate coordinate) const {
    return (coordinate.jPosition ≥ 0) ∧ (coordinate.jPosition < (int)tiles[0].size())
           ∧ (coordinate.iPosition ≥ 0) ∧ (coordinate.iPosition < (int)tiles.size());
}


bool Map::_areCoordinatesEqual(Coordinate a, Coordinate b){
    return (a.iPosition ≡ b.iPosition) ∧ (a.jPosition ≡ b.jPosition);
}

void Map::_buildSearchRegion(Coordinate center, unsigned int range, Coordinate& topLeft, Coordinate& bottomRight) const {
    Coordinate aux{};
    aux.iPosition = static_cast<int>(center.iPosition − range);
    aux.jPosition = static_cast<int>(center.jPosition − range);
    topLeft = _getValidCoordinate(aux);
    aux.iPosition = static_cast<int>(center.iPosition + range);
    aux.jPosition = static_cast<int>(center.jPosition + range);
    bottomRight = _getValidCoordinate(aux);
}

Coordinate Map::_getValidCoordinate(Coordinate coordinate) const {
    if (coordinate.jPosition ≥ (int)tiles[0].size()) {
        coordinate.jPosition = (int)tiles[0].size() − 1;
    } else if (coordinate.jPosition < 0) {
        coordinate.jPosition = 0;
    }
    if (coordinate.iPosition ≥ (int)tiles.size()) {
        coordinate.iPosition = (int)tiles.size() − 1;
    } else if (coordinate.iPosition < 0) {
        coordinate.iPosition = 0;
    }
    return coordinate;
}


//Guarda en nodes y parentsAndChilds los nodos correspondientes, revisando los
//nodos que se encuentren adyacentes a referencia, tambien actualiza las distancias
```

```cpp
//de los nodos de ser necesario
void Map::_storeAdjacentPositions(
        PointAndDistance refference, std::unordered_map<Coordinate, unsigned int>& distances,
        std::unordered_map<Coordinate, Coordinate>& parentsAndChilds,
        std::priority_queue<PointAndDistance, std::vector<PointAndDistance>,
                            InverseCoordinateDistance>& nodes,
        Coordinate destination) const {
    Coordinate topLeft{}, bottomRight{};
    PointAndDistance aux{};
    _buildSearchRegion(refference.point, 1, topLeft, bottomRight);
    for (int i = topLeft.iPosition; i ≤ bottomRight.iPosition; ++i) {
        for (int j = topLeft.jPosition; j ≤ bottomRight.jPosition; ++j) {
            aux.point.iPosition = i;
            aux.point.jPosition = j;
            aux.distance = _getDistance(refference.point, aux.point);
            //if (esta a distancia 1 y (la posicion es alcanzable o tiene un jugador) y no esta en una ciudad)
            if (((aux.distance ≡ 1) ∧ (tiles[i][j].isAvailable() ∨
                tiles[i][j].hasMonsterTarget())) ∧ ¬tiles[i][j].isInCity()) {
                aux.distance += refference.distance + _getDistance(aux.point, destination);
                if ((distances.count(aux.point) ≡ 0) ∨
                    (distances.at(aux.point) > aux.distance)) {
                    nodes.push(aux);
                    distances[aux.point] = aux.distance;
                    parentsAndChilds[aux.point] = refference.point;
                }
            }
        }
    }
}

void Map::_storePath(Coordinate initialPosition, Coordinate desiredPosition,
                     const std::unordered_map<Coordinate, Coordinate>& parentsAndChilds,
                     std::list<Coordinate>& path) {
    Coordinate aux = desiredPosition;
    while (¬_areCoordinatesEqual(aux, initialPosition)) {
        path.push_front(aux);
        aux = parentsAndChilds.at(aux);
    }
}


bool Map::_isReachable(Coordinate position) const {
    Coordinate topLeft{};
    Coordinate bottomRight{};
    Coordinate aux{};
    _buildSearchRegion(position, 1, topLeft, bottomRight);
    for (int i = topLeft.iPosition; i ≤ bottomRight.iPosition; ++i) {
        for (int j = topLeft.jPosition; j ≤ bottomRight.jPosition; ++j) {
            aux = {i, j};
            if ((_getDistance(position, aux) ≡ 1) ∧ (tiles[i][j].isAvailable() ∧ ¬tiles[i][j].isInCity())) {
                return true;
            }
        }
    }
    return false;
}



void Map::_initializeConstructorMaps(
        std::unordered_map<std::string, GameType::Entity> &entities,
        std::unordered_map<std::string, GameType::Structure> &structures,
        std::unordered_map<std::string, GameType::FloorType> &floors) {
    entities ={{"Nothing", GameType::Entity::GUARD}, {"Priest", GameType::Entity::PRIEST},
             {"Trader", GameType::Entity::TRADER}, {"Banker", GameType::Entity::BANKER}};
    structures = {{"BoneGuy", GameType::Structure::BONE_GUY}, {"BrokenRipStone", GameType::Structure::BROKEN_RIP_STONE},
            {"Bush", GameType::Structure::BUSH}, {"DeadBush", GameType::Structure::DEAD_BUSH},
            {"DeadGuy", GameType::Structure::DEAD_GUY}, {"DeadTree", GameType::Structure::DEAD_TREE},
            {"FatTree", GameType::Structure::FAT_TREE}, {"HangedGuy", GameType::Structure::HANGED_GUY},
            {"House1", GameType::Structure::HOUSE1}, {"House2", GameType::Structure::HOUSE2},
            {"House3", GameType::Structure::HOUSE3}, {"LongTree", GameType::Structure::LONG_TREE},
            {"PalmTree", GameType::Structure::PALM_TREE}, {"RipStone", GameType::Structure::RIP_STONE},
            {"Tree", GameType::Structure::TREE}, {"VeryDeadGuy", GameType::Structure::VERY_DEAD_GUY},
            {"SunkenColumn", GameType::Structure::SUNKEN_COLUMN}, {"SunkenShip", GameType::Structure::SUNKEN_SHIP},
            {"Nothing", GameType::Structure::NO_STRUCTURE}};

    floors = {{"Grass0", GameType::FloorType::GRASS0}, {"Grass1", GameType::FloorType::GRASS1},
            {"Grass2", GameType::FloorType::GRASS2}, {"Grass3", GameType::FloorType::GRASS3},
            {"Sand", GameType::FloorType::SAND}, {"Water0", GameType::FloorType::WATER0},
            {"Water1", GameType::FloorType::WATER1}, {"Water2", GameType::FloorType::WATER2},
            {"Water3", GameType::FloorType::WATER3}, {"PrettyRoad0", GameType::FloorType::PRETTY_ROAD0},
            {"PrettyRoad1", GameType::FloorType::PRETTY_ROAD1}, {"PrettyRoad2", GameType::FloorType::PRETTY_ROAD2},
            {"PrettyRoad3", GameType::FloorType::PRETTY_ROAD3}, {"PrettyGrass0", GameType::FloorType::PRETTY_GRASS0},
            {"PrettyGrass1", GameType::FloorType::PRETTY_GRASS1}, {"PrettyGrass2", GameType::FloorType::PRETTY_GRASS2},
            {"PrettyGrass3", GameType::FloorType::PRETTY_GRASS3}, {"DeadGrass0", GameType::FloorType::DEAD_GRASS0},
            {"DeadGrass1", GameType::FloorType::DEAD_GRASS1}, {"DeadGrass2", GameType::FloorType::DEAD_GRASS2},
            {"DeadGrass3", GameType::FloorType::DEAD_GRASS3}, {"DarkWater0", GameType::FloorType::DARK_WATER0},
            {"DarkWater1", GameType::FloorType::DARK_WATER1}, {"DarkWater2", GameType::FloorType::DARK_WATER2},
            {"DarkWater3", GameType::FloorType::DARK_WATER3}};
}

void Map::_getTargets(Coordinate center, unsigned int range, std::vector<Coordinate> &targets,
                      bool detectUnreachableTargets) const {
    Coordinate topLeft{}, bottomRight{}, aux{};
    _buildSearchRegion(center, range, topLeft, bottomRight);
    for (int i = topLeft.iPosition; i ≤ bottomRight.iPosition; ++i) {
        for (int j = topLeft.jPosition; j ≤ bottomRight.jPosition; ++j) {
            if (tiles[i][j].hasMonsterTarget() ∧ ¬tiles[i][j].isInCity() ∧
                (_isReachable({i, j}) ∨ detectUnreachableTargets)) {
                aux.iPosition = i;
                aux.jPosition = j;
                targets.push_back(aux);
```

```
            }
        }
    }
}



////////////////////////////PUBLIC///////////////////////////

Map::Map(MapFileReader &mapFile, std::list<Coordinate>& priests) {
    CitizenFactory citizenFactory;
    MapSize mapSize = mapFile.getMapDimensions();
    TileInfo aux{};
    std::shared_ptr<Entity> citizen;
    std::unordered_map<std::string, GameType::Entity> entities;
    std::unordered_map<std::string, GameType::Structure> structures;
    std::unordered_map<std::string, GameType::FloorType> floors;
    _initializeConstructorMaps(entities, structures, floors);
    //tiles.resize(mapSize.height, std::vector<Tile>(mapSize.width));
    GameType::Entity auxEntity{};
    for (unsigned int i = 0; i < mapSize.height; ++i) {
        tiles.emplace_back();
        for (unsigned int j = 0; j < mapSize.width; ++j) {
            aux = mapFile.getTileInfo(i, j);
            if (aux.entityType ≡ "Nothing") {
                citizen.reset();
            } else {
                auxEntity = entities.at(aux.entityType);
                citizenFactory.storeCitizen(citizen, auxEntity,
                        {static_cast<int>(i), static_cast<int>(j)});
                if (auxEntity ≡ GameType::PRIEST) {
                    priests.push_back({static_cast<int>(i), static_cast<int>(j)});
                }
            }
            tiles.at(i).emplace_back(aux.isOccupable, aux.isFromCity, floors.at(aux.tileType), structures
                    .at(aux.structureType), std::move(citizen));
        }
    }
}


std::pair<AttackResult, bool> Map::attackTile(int damage, unsigned int level, bool isAPlayer,
                             Coordinate coordinate) {
    return tiles[coordinate.iPosition][coordinate.jPosition].attacked(damage, level, isAPlayer);
}

void Map::getMoveTargets(Coordinate center, unsigned int range, std::vector<Coordinate>& targets) const {
    _getTargets(center, range, targets, false);
}


void Map::getAttackTargets(Coordinate center, unsigned int range,
                           std::vector<Coordinate> &targets) const {
    _getTargets(center, range, targets, true);
}

bool Map::getPath(Coordinate currentPosition, Coordinate desiredPosition, std::list<Coordinate>& path) const {
    std::priority_queue<PointAndDistance, std::vector<PointAndDistance>, InverseCoordinateDistance> nodes;

    //Key: hijo, Dato: padre
    std::unordered_map<Coordinate, Coordinate> parentsAndChilds;

    //Key:Posicion, Dato: distancia
    std::unordered_map<Coordinate, unsigned int> distances;

    PointAndDistance aux{};
    aux.point = currentPosition;
    aux.distance = 0;
    nodes.push(aux);
    while (¬nodes.empty()) {
        aux = nodes.top();
        nodes.pop();
        if (_areCoordinatesEqual(aux.point, desiredPosition)) {
            _storePath(currentPosition, desiredPosition, parentsAndChilds, path);
            return true;
        }
        _storeAdjacentPositions(aux, distances, parentsAndChilds, nodes, desiredPosition);
    }
    return false;
}

void Map::addEntity(Coordinate position, std::shared_ptr<Entity> ∧entity) {
    if (¬_isCoordinateValid(position)) {
        throw (std::invalid_argument("Out of bounds coordinate"));
    }
    entity→setPosition(position);
    tiles[position.iPosition][position.jPosition].addEntity(std::move(entity));
}

std::shared_ptr<Item> Map::removeItem(Coordinate position) {
    if (¬_isCoordinateValid(position)) {
        throw (std::invalid_argument("Out of bounds coordinate"));
    }
    return tiles[position.iPosition][position.jPosition].removeItem();
}

void Map::removeEntity(Coordinate position) {
    if (¬_isCoordinateValid(position)) {
        throw (std::invalid_argument("Out of bounds coordinate"));
    }
    tiles[position.iPosition][position.jPosition].removeEntity();
}

void Map::moveEntity(Coordinate startingPosition, Coordinate finalPosition) {
    if ((¬_isCoordinateValid(startingPosition)) ∨
        (¬_isCoordinateValid(finalPosition))) {
        return;
    }
    if (¬tiles[finalPosition.iPosition][finalPosition.jPosition].isAvailable()) {
        return;
    }
    Tile& tile = tiles[finalPosition.iPosition][finalPosition.jPosition];
    tile.moveEntity(tiles[startingPosition.iPosition][startingPosition.jPosition],
                        finalPosition);
}

bool Map::isPlaceAvailable(Coordinate position) const {
    return _isCoordinateValid(position) ∧
            tiles[position.iPosition][position.jPosition].isAvailable();
}

```

```
void Map::addItemsToTile(std::list<std::shared_ptr<Item>>∧ items, Coordinate position) {
    if (¬_isCoordinateValid(position)) {
        throw (std::invalid_argument("Out of bounds coordinate"));
    }
    tiles[position.iPosition][position.jPosition].addItem(std::move(items));
}

void Map::addItemsToTile(std::shared_ptr<Item> ∧item, Coordinate position) {
    if (¬_isCoordinateValid(position)) {
        throw (std::invalid_argument("Out of bounds coordinate"));
    }
    tiles[position.iPosition][position.jPosition].addItem(std::move(item));
}


Coordinate Map::getMonsterCoordinate() {
    unsigned int xPosition = Calculator::getRandomInt(0, (int)(tiles.size() − 1));
    unsigned int yPosition = Calculator::getRandomInt(0, (int)(tiles[0].size() − 1));
    while ((¬tiles[xPosition][yPosition].isAvailable()) ∨ (tiles[xPosition][yPosition].isInCity())) {
        xPosition = Calculator::getRandomInt(0, (int)(tiles.size() − 1));
        yPosition = Calculator::getRandomInt(0, (int)(tiles[0].size() − 1));
    }
    return {static_cast<int>(xPosition), static_cast<int>(yPosition)};
}

void Map::list(Player &player, Coordinate coordinate) {
    if (_isCoordinateValid(coordinate)) {
        tiles[coordinate.iPosition][coordinate.jPosition].list(player);
    }
}

void Map::withdraw(Player &player, const std::string &itemName, Coordinate coordinate) {
    if (_isCoordinateValid(coordinate)) {
        tiles[coordinate.iPosition][coordinate.jPosition].withdraw(player, itemName);
    }
}

void Map::deposit(Player &player, const std::string &itemName, Coordinate coordinate) {
    if (_isCoordinateValid(coordinate)) {
        tiles[coordinate.iPosition][coordinate.jPosition].deposit(player, itemName);
    }
}

void Map::buy(Player &player, const std::string &itemName, Coordinate coordinate) {
    if (_isCoordinateValid(coordinate)) {
        tiles[coordinate.iPosition][coordinate.jPosition].buy(player, itemName);
    }
}

void Map::sell(Player &player, const std::string &itemName, Coordinate coordinate) {
    if (_isCoordinateValid(coordinate)) {
        tiles[coordinate.iPosition][coordinate.jPosition].sell(player, itemName);
    }
}

```

```cpp
void Map::operator>>(std::stringstream &mapBuffer) const {
    msgpack::type::tuple<int32_t , int32_t> mapSize(tiles.size(), tiles[0].size(
));
    msgpack::pack(mapBuffer, mapSize);
    for (const auto & row : tiles) {
        for (const auto & tile : row) {
            tile >> mapBuffer;
        }
    }
}

const Item* Map::peekShowedItemData(Coordinate coordinate) {
    if (¬_isCoordinateValid(coordinate)) {
        throw std::invalid_argument("Invalid coordinate in peekShoedItemData");
    }
    return tiles[coordinate.iPosition][coordinate.jPosition].peekShowedItemData(
);
}

Coordinate Map::getSpawnCoordinateArroundPosition(Coordinate refference) {
    Coordinate topLeft{}, bottomRight{};
    _buildSearchRegion(refference, RESPAWN_RANGE, topLeft, bottomRight);
    for (int i = topLeft.iPosition; i ≤ bottomRight.iPosition; ++i) {
        for (int j = topLeft.jPosition; j ≤ bottomRight.jPosition; ++j) {
            if (tiles[i][j].isAvailable() ∧ tiles[i][j].isInCity()) {
                return {i, j};
            }
        }
    }
    return {−1, −1};
}

Coordinate Map::getMonsterRandomPosition(Coordinate refference) const {
    std::vector<Coordinate> positions;
    Coordinate topLeft{}, bottomRight{}, aux{};
    _buildSearchRegion(refference, 1, topLeft, bottomRight);
    for (int i = topLeft.iPosition; i ≤ bottomRight.iPosition; ++i) {
        for (int j = topLeft.jPosition; j ≤ bottomRight.jPosition; ++j) {
            aux = {i, j};
            if ((refference.calculateDistance(aux) ≡ 1) ∧ (tiles[i][j].isAvailab
le()) ∧
                (¬tiles[i][j].isInCity())) {
                positions.push_back(aux);
            }
        }
    }
    if (positions.empty()) {
        return {−1, −1};
    }
    return positions[Calculator::getRandomInt(0, positions.size() − 1)];
}

void Map::requestRestore(Player &player, Coordinate target) {
    if (_isCoordinateValid(target)) {
        tiles[target.iPosition][target.jPosition].requestRestore(player);
    }
}
```

```cpp
//
// Created by agustin on 10/6/20.
//

#ifndef ARGENTUM_INVERSECOORDINATEDISTANCE_H
#define ARGENTUM_INVERSECOORDINATEDISTANCE_H


#include "PointAndDistance.h"

//Functor utilizado para obtener el nodo de menor distancia en la priority_queue
//para el pathfinding
class InverseCoordinateDistance {
public:
    bool operator()(const PointAndDistance& a, const PointAndDistance& b);
};


#endif //ARGENTUM_INVERSECOORDINATEDISTANCE_H
```

```cpp
//
// Created by agustin on 10/6/20.
//

#include "InverseCoordinateDistance.h"

bool InverseCoordinateDistance::operator()(const PointAndDistance &a,
                                           const PointAndDistance &b) {
    return a.distance > b.distance;
}
```

```cpp
//
// Created by agustin on 6/6/20.
//

#ifndef ARGENTUM_COORDINATE_H
#define ARGENTUM_COORDINATE_H

#include <functional>
#include <cstdlib>

//Struct auxiliar para facilitar el paso de coordenadas, debido a esto no
//tiene ningun comportamiento y sus atributos son publicos
struct Coordinate {
    int iPosition;
    int jPosition;

    bool operator≡(const Coordinate& other) {
        return ((iPosition ≡ other.iPosition) ∧ (jPosition ≡ other.jPosition));
    }

    bool operator≠(const Coordinate& other) {
        return ¬((iPosition ≡ other.iPosition) ∧ (jPosition ≡ other.jPosition))
;
    }

    unsigned int calculateDistance(const Coordinate& other) {
        return std::abs(iPosition − other.iPosition) + std::abs(jPosition − othe
r.jPosition);
    }
};

namespace std {
    template <> struct hash<Coordinate> {
        size_t operator()(const Coordinate& x) const {
            return (x.jPosition << 15) + x.jPosition;
        }
    };

    template <> struct equal_to<Coordinate> {
        size_t operator()(const Coordinate& x, const Coordinate& y) const {
            return (x.iPosition ≡ y.iPosition) ∧ (x.jPosition ≡ y.jPosition);
        }
    };
}

#endif //ARGENTUM_COORDINATE_H
```

```
//
// Created by agustin on 3/7/20.
//

#ifndef ARGENTUM_USERETURNDATA_H
#define ARGENTUM_USERETURNDATA_H

#include "Item.h"

/*Este struct es util para cuando mandamos los updates a los players de lo que
 * se equipo otro jugador*/

struct UseReturnData {
    GameType::EquipmentPlace equipmentPlace;
    int32_t id;
};


#endif //ARGENTUM_USERETURNDATA_H
```

```
//
// Created by agustin on 19/6/20.
//

#ifndef ARGENTUM_POTION_H
#define ARGENTUM_POTION_H


#include "../Item.h"
#include "../../Config/Configuration.h"

class ItemTests;

/*Esta clase encapsula el comportamiento global de las pociones que
 * son consumibles por el player (pociones de vida/mana)*/

class Potion: public Item {
protected:
    unsigned int recoveryValue;

    friend ItemTests;

public:
    explicit Potion(GameType::Potion potion);

    /*Indica que una vez que son usadas las pociones son descartadas*/
    GameType::EquipmentPlace use(Player& player) override;

    /*Debe llamar a la funcion de player que restaura el atributo correspondient
e*/
    virtual void restoreStat(Player& player) = 0;

    virtual ~Potion() = default;
};


#endif //ARGENTUM_POTION_H
```

```cpp
//
// Created by agustin on 19/6/20.
//

#include "Potion.h"


GameType::EquipmentPlace Potion::use(Player &player) {
    restoreStat(player);
    return GameType::EQUIPMENT_PLACE_NONE;
}

Potion::Potion(GameType::Potion potion):
                Item(GameType::ITEM_TYPE_POTION, Configuration::getInstance().configPotionData(potion).name) {
    recoveryValue = Configuration::getInstance().configPotionData(potion).recoveryValue;
    id = potion;
}
```

```cpp
//
// Created by agustin on 19/6/20.
//

#ifndef ARGENTUM_MANAPOTION_H
#define ARGENTUM_MANAPOTION_H


#include "Potion.h"

/*Esta clase representa la pocion de mana que puede consumir el player*/

class ManaPotion: public Potion {
private:
    void restoreStat(Player& player) override;

public:
    explicit ManaPotion(): Potion(GameType::MANA_POTION) {}
    ~ManaPotion() override;
};


#endif //ARGENTUM_MANAPOTION_H
```

```cpp
//
// Created by agustin on 19/6/20.
//

#include "ManaPotion.h"
#include "../../Entities/Player.h"

void ManaPotion::restoreStat(Player &player) {
    player.restoreMana(recoveryValue);
}

ManaPotion::~ManaPotion() = default;
```

```cpp
//
// Created by agustin on 19/6/20.
//

#ifndef ARGENTUM_HEALTHPOTION_H
#define ARGENTUM_HEALTHPOTION_H


#include "Potion.h"

/*Esta clase representa la pocion de vida que puede consumir el player*/

class HealthPotion: public Potion {
private:
    void restoreStat(Player& player) override;
public:
    explicit HealthPotion(): Potion(GameType::HEALTH_POTION) {}
    ~HealthPotion() override;
};


#endif //ARGENTUM_HEALTHPOTION_H
```

```cpp
//
// Created by agustin on 19/6/20.
//

#include "HealthPotion.h"
#include "../../Entities/Player.h"

void HealthPotion::restoreStat(Player &player) {
    player.restoreLife(recoveryValue);
}

HealthPotion::~HealthPotion() = default;
```

```cpp
//
// Created by agustin on 6/6/20.
//

#ifndef ARGENTUM_GOLD_H
#define ARGENTUM_GOLD_H


#include "../Item.h"

class ItemTests;

/*Clase que representa un puniado de oro en el piso*/

class Gold: public Item {
private:
    unsigned int amount;

    friend ItemTests;

public:
    explicit Gold(unsigned int amount);

    /*Retorna la posicion donde se equipa (como es oro no se equipa en ningun la
do sino que
     * en realidad le sumamos a la cantidad de oro que guarda el player)*/
    GameType::EquipmentPlace use(Player& player) override;

    /*Retorna la cantidad de oro que representa la instancia Gold*/
    unsigned int getAmount() const;

    /* Retorna true, no es la solucion mas limpia pero era la mas sencilla de
     * implementar. Esto soluciona el tema donde el gold en realidad no va equip
ado en si
     * asi que no lo agregamos al inventario, pero como hereda de item necesitab
amos
     * poder distinguir si era o no oro para sumarle a la variable del player*/
    bool isGold() const override;
};

#endif //ARGENTUM_GOLD_H
```

```
//
// Created by agustin on 6/6/20.
//

#include "Gold.h"
#include "../../Config/Configuration.h"

Gold::Gold(unsigned int amount): Item(GameType::ITEM_TYPE_GOLD,
                    Configuration::getInstance().configGetGoldName()/*, 0*/){
    this→amount = amount;
}

GameType::EquipmentPlace Gold::use(Player &player) {
    return GameType::EQUIPMENT_PLACE_NONE;
}

unsigned int Gold::getAmount() const {
    return amount;
}

bool Gold::isGold() const {
    return true;
}
```

```
//
// Created by agustin on 16/6/20.
//

#ifndef ARGENTUM_ITEMSFACTORY_H
#define ARGENTUM_ITEMSFACTORY_H

#include <unordered_map>
#include <memory>
#include <vector>
#include "../../libs/GameEnums.h"

class Item;
class ItemTests;

/*Esta es una factory de items, para no estar hardcodeando cada vez que necesite
mos
 * crear un item particular (ya que como se crea un item depende del tipo)*/

typedef void (*objectCreator) (std::shared_ptr<Item>&);

class ItemsFactory {
private:
    std::unordered_map<std::string, objectCreator> itemsCreators;
    std::vector<const std::string*> itemsNames;

    friend ItemTests;

private:
    ItemsFactory();

    static void _storeBlueTunic(std::shared_ptr<Item>& item);
    static void _storeLeatherArmor(std::shared_ptr<Item>& item);
    static void _storePlateArmor(std::shared_ptr<Item>& item);
    static void _storeKingArmor(std::shared_ptr<Item> &item);

    static void _storeHood(std::shared_ptr<Item>& item);
    static void _storeIronHelmet(std::shared_ptr<Item>& item);
    static void _storeMagicHat(std::shared_ptr<Item>& item);

    static void _storeIronShield(std::shared_ptr<Item>& item);
    static void _storeTurtleShield(std::shared_ptr<Item>& item);

    static void _storeAshRod(std::shared_ptr<Item>& item);
    static void _storeCompositeBow(std::shared_ptr<Item>& item);
    static void _storeElvenFlute(std::shared_ptr<Item>& item);
    static void _storeGnarledStaff(std::shared_ptr<Item>& item);
    static void _storeLinkedStaff(std::shared_ptr<Item>& item);
    static void _storeLongsword(std::shared_ptr<Item>& item);
    static void _storeSimpleBow(std::shared_ptr<Item>& item);
    static void _storeWarhammer(std::shared_ptr<Item>& item);
    static void _storeAxe(std::shared_ptr<Item>& item);

    static void _storeManaPotion(std::shared_ptr<Item>& item);
    static void _storeHealthPotion(std::shared_ptr<Item>& item);

    static void _storeRandomPotion(std::shared_ptr<Item>& item);

    static void _storeGold(std::shared_ptr<Item>& item, unsigned int amount);

public:

    static ItemsFactory& getInstance();

    /*Guarda una instancia del item pedido en item, si el nombre del item pasado
    no existe entonces tira la exepcion out_of_range*/
    void storeItemInstance(const std::string& itemName, std::shared_ptr<Item>& i
tem);

    /*Guarda una instancia del item pedido en item, si el nombre del item pasado
    no existe entonces tira la exepcion out_of_range*/
    void storeItemInstance(GameType::ItemType type, int32_t instance,
                                         std::shared_ptr<Item> &item);

    /*Almacena un item aleatorio en item, goldMultiplier es el valor por el que
se
    multiplica el porcentaje de oro a generar (del 0 al 20%)*/
    void storeRandomDrop(std::shared_ptr<Item>& item, unsigned int goldMultiplie
r);
};


#endif //ARGENTUM_ITEMSFACTORY_H
```

```
//
// Created by agustin on 6/6/20.
//

#ifndef ARGENTUM_ITEM_H
#define ARGENTUM_ITEM_H

#include <string>
#include <memory>
#include "../../libs/GameEnums.h"
#include "UseReturnData.h"

class ItemTests;
class Player;

//Clase interfaz de la que heredan todos los items
//Los items solo tienen sentido en un inventario de un jugador
class Item {
private:
    //El id esta asocioado al tipo de item que es, estos se repiten entre los distintos
    //tipos de items. Ej: Se puede tener un arma y un escudo con el mismo id, pero se
    //diferencian por ser uno un arma y otro un escudo
    GameType::ItemType type;
    const std::string& name;

    friend ItemTests;

protected:
    int32_t id{};

public:
    Item(GameType::ItemType _type, const std::string& name);

    /*use debe retornar el lugar en el que debera equiparse el item una vez usado desde
    un inventario, si debe ser descartado entonces se tiene que retornar INVENTORY_PLACE_NONE
    Esta funcion retorna en cierta forma el tipo de item que es*/
    virtual GameType::EquipmentPlace use(Player& player);

    /*Retorna el nombre del item (viene asignado desde el archivo de config)*/
    const std::string& getName() const;

    /*Esta implementada como false, el unico que la reimplementa es gold que
     * retorna true*/
    virtual bool isGold() const;

    /*Carga en el buffer la informacion correspondiente al item acorde
     * al protocolo, esta funcion se usa cuando un nuevo player se conecta
     * al juego*/
    void loadDropItemData(std::stringstream& buffer, int32_t i, int32_t j) const;

    /*Carga en el buffer la infromacion corresopndiente al item equipado
     * acorde al protocolo, esta funcion se usa cuando un nuevo player
     * se conecta al juego*/
    void loadEquippedItemData(std::stringstream& buffer);

    /*Carga en el buffer el tipo de item y su id (es decir si es un arma cual seria,
     * o si es ropa cual seria. La usa el inventario para el mensaje individual
     * de cada player (este mensaje es para la UI personal, es decir, no para el mapa)*/
    void loadTypeAndId(std::stringstream& buffer);
```

```

    virtual ~Item() = default;

    /*Retorna el tipo particular del item (si es una weapon retornaria que tipo
     * de weapon, lo mismo si fuera un chest o un shield)*/
    int32_t getId();

    /*Retorna la clase del item (si es una weapon, shield, chest, etc)*/
    GameType::ItemType getType();
};


#endif //ARGENTUM_ITEM_H
```

```
//
// Created by agustin on 5/7/20.
//

#ifndef ARGENTUM_ITEMDATA_H
#define ARGENTUM_ITEMDATA_H

#include <cstdint>
#include "../../libs/GameEnums.h"
#include "../Map/Coordinate.h"

/*Este struct es util para encapsular los datos necesarios del item
 * para el protcolo*/

struct ItemData {
    GameType::ItemType type;
    int32_t id;
    Coordinate coordinate;
};

#endif //ARGENTUM_ITEMDATA_H
```

```
//
// Created by agustin on 9/6/20.
//

#include "Item.h"
#include <msgpack.hpp>

MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::ItemType)

Item::Item(GameType::ItemType _type, const std::string &_name): name(_name) {
    type = _type;
}

GameType::EquipmentPlace Item::use(Player &player) {
    return GameType::EQUIPMENT_PLACE_NONE;
}

const std::string &Item::getName() const {
    return name;
}

bool Item::isGold() const {
    return false;
}

void Item::loadDropItemData(std::stringstream &buffer, int32_t i, int32_t j) con
st {
    msgpack::type::tuple<GameType::EventID> idType(GameType::CREATE_ITEM);
    msgpack::type::tuple<GameType::ItemType, int32_t, int32_t, int32_t>
                                        data(type, id, i, j);
    msgpack::pack(buffer, idType);
    msgpack::pack(buffer, data);
}

void Item::loadEquippedItemData(std::stringstream &buffer) {
    msgpack::type::tuple<int32_t> data(id);
    msgpack::pack(buffer, data);
}

void Item::loadTypeAndId(std::stringstream &buffer) {
    msgpack::type::tuple<GameType::ItemType, int32_t> data(type, id);
    msgpack::pack(buffer, data);
}

int32_t Item::getId() {
    return id;
}

GameType::ItemType Item::getType() {
    return type;
}
```

```
//
// Created by agustin on 8/6/20.
//

#ifndef ARGENTUM_INVENTORY_H
#define ARGENTUM_INVENTORY_H

#include <vector>
#include <memory>
#include <unordered_map>
#include <list>
#include "Item.h"
#include "UseReturnData.h"
#include "../Server/PlayerData.hpp"

class ItemTests;
class EntityTests;
class MapTests;
class Game;
class PlayerStats;
class Weapon;
class Player;
class Clothing;
class Item;
class Minichat;
struct Coordinate;


/*Esta clase representa los items que almacena y tiene equipados el jugador*/
class Inventory {
private:
    unsigned int storedItemsAmount{};
    std::vector<std::shared_ptr<Item>> items;
    std::unordered_map<GameType::EquipmentPlace, std::shared_ptr<Clothing>> clothingEquipment;
    std::shared_ptr<Weapon> equippedWeapon;

    friend ItemTests;
    friend EntityTests;
    friend MapTests;

private:
    UseReturnData _manageItemPlacement(GameType::EquipmentPlace equipmentPlace, unsigned int itemPosition);
    void _dropEquippedItems(std::list<std::shared_ptr<Item>>& droppedItems);
    static void _storeNullItemData(std::stringstream& buffer);
    void _restoreDefaultEquipment();
    void _loadInitialInventory(const PlayerData& data);

public:
    explicit Inventory(const PlayerData& data);

    /*Adquiere el shared pointer recibido y lo guarda si hay espacio y retorna
    true. Si no hay espacio o item es null no adquiere el puntero y retorna false*/
    bool addItem(std::shared_ptr<Item>& item);

    /*Elimina el item del inventario de la posicion recibida  y lo retorna,
    dejando el lugar que ocupaba para un nuevo item que quiera ser guardado
    Si no hay un item en la posicion retorna un shared_ptr que almacena null_ptr*/
    std::shared_ptr<Item> removeItem(unsigned int itemPosition);

    /*Elimina el item con el nombre recibido del inventario y lo retorna,
    dejando el lugar que ocupaba para un nuevo item que quiera ser guardado
    Si no hay un item con el nombre recibido retorna un shared_ptr que almacena null_ptr*/
```

```
    std::shared_ptr<Item> removeItem(const std::string& itemName);

    /*Usa el item en la posicion indicada, si no hay un item en la posicion no
    hace nada*/
    UseReturnData useItem(Player& player, unsigned int itemPosition);

    /*Retorna el danio generado por el arma dentro del rango de ella
     * Si el target esta fuera del rango del arma retorna 0 de danio*/
    int getWeaponDamage(Coordinate currentPosition, Coordinate target, PlayerStats& stats) const;

    /*Retorna la defensa total provista por la armadura equipada (casco, chest, shield)*/
    unsigned int getDefense();

    /*Retorna una lista con todos los items del iventario, quitandolos del mismo*/
    std::list<std::shared_ptr<Item>> dropAllItems();

    /*Desequipa la ropa de la posicion seleccionada en los equipados y la almacena
     * en el inventario. Si no tiene equipado nada (o sea un default) no hace nada*/
    bool unequip(GameType::EquipmentPlace clothing);

    /*Desequipa el arma  y la almacena en el inventario. Si no tiene equipada un arma
     * (o sea un default) no hace nada*/
    bool unequip();

    /*Almacena en el buffer la informacion de los items equipados del jugador
     * acorde al protocolo*/
    void storeEquippedItems(std::stringstream& buffer) const;

    /*Almacena toda la data relevante que tiene en el inventario/equipado el
     * jugador acorde al protocolo. Se usa para los updates individuales
     * que se envian periodicamente a cada player (para que sepa que tiene
     * equipado en la UI)*/
    void storeAllData(std::stringstream& buffer) const;

    /*Retorna el tipo de arma (Axe, Longsword, etc)*/
    int32_t getWeaponId();

    /*Retorna true si se tiene el item almacenado en el inventario (no equipado),
     * false en caso contrario*/
    bool hasItem(const std::string& itemName);

    bool isFull() const;

    /*Almacena en el minichat los nombres de los items que se encuentran en el
     * inventario*/
    void getInventoryNames(Minichat& chat);

    /*Almacena el inventario del player en pData, se usa para el backup del archivo*/
    void getData(PlayerData& pData) const;
};


#endif //ARGENTUM_INVENTORY_H
```

```
//
// Created by agustin on 9/6/20.
//

#ifndef ARGENTUM_SHIELD_H
#define ARGENTUM_SHIELD_H


#include "Clothing.h"

class ItemTests;

/*Esta clase encapsula el comportamiento particular de los escudos*/

class Shield: public Clothing {
private:
    friend ItemTests;

public:
    explicit Shield(GameType::Clothing clothing): Clothing(clothing) {}

    /*Retorna la posicion donde se equipa (como es un shield retornara la posici
on del shield)
    * Esto es util para saber en ejecucion donde se equipa el item ya que lo gua
rdamos
    * en el padre*/
    GameType::EquipmentPlace use(Player& player) override;

    /*Esta funcion existe para que cuando el juegador muera y tire todos sus
    * items no tire los items default*/
    bool isDefault() const override;
};


#endif //ARGENTUM_SHIELD_H
```

```
//
// Created by agustin on 9/6/20.
//

#include "Shield.h"

GameType::EquipmentPlace Shield::use(Player &player) {
    return GameType::EQUIPMENT_PLACE_SHIELD;
}

bool Shield::isDefault() const {
    return (static_cast<GameType::Clothing>(id) ≡ GameType::Clothing::NO_SHIELD)
;
}
```

```
//
// Created by agustin on 9/6/20.
//

#ifndef ARGENTUM_HEAD_H
#define ARGENTUM_HEAD_H


#include "Clothing.h"

class ItemTests;

/*Esta clase encapsula el comportamiento particular de la ropa que se
 * equipa en la cabeza (cascos, sombreros)*/

class Head: public Clothing {
private:
    friend ItemTests;

public:
    explicit Head(GameType::Clothing clothing) : Clothing(clothing) {}

    /*Retorna la posicion donde se equipa (como es un head retornara la posicion del head)
    * Esto es util para saber en ejecucion donde se equipa el item ya que lo guardamos
    * en el padre*/
    GameType::EquipmentPlace use(Player& player) override;

    /*Esta funcion existe para que cuando el juegador muera y tire todos sus
    * items no tire los items default*/
    bool isDefault() const override;
};


#endif //ARGENTUM_HEAD_H
```

```
//
// Created by agustin on 9/6/20.
//

#include "Head.h"

GameType::EquipmentPlace Head::use(Player &player) {
    return GameType::EQUIPMENT_PLACE_HEAD;
}

bool Head::isDefault() const {
    return (static_cast<GameType::Clothing>(id) ≡ GameType::Clothing::NO_HELMET);
}
```

```cpp
//
// Created by agustin on 9/6/20.
//

#ifndef ARGENTUM_CLOTHING_H
#define ARGENTUM_CLOTHING_H

#include "../Item.h"
#include "../../Config/Configuration.h"

class ItemTests;

/*Esta clase encapsula el comportamiento general de la ropa*/

class Clothing : public Item {
private:
    unsigned int minDefense;
    unsigned int maxDefense;

    friend ItemTests;

public:
    explicit Clothing(GameType::Clothing clothing);

    /*Retorna un random entre la minima y maxima defensa del item*/
    unsigned int getDefense() const;

    /*Esta funcion existe para que cuando el player muera y tire todos sus items
    sepa reconocer los default y no los tire*/
    virtual bool isDefault() const = 0;

    virtual ~Clothing() = default;
};


#endif //ARGENTUM_CLOTHING_H
```

```cpp
//
// Created by agustin on 9/6/20.
//

#include "Clothing.h"
#include "../../Config/Calculator.h"

Clothing::Clothing(GameType::Clothing clothing): Item(GameType::ITEM_TYPE_CLOTHI
NG,
                Configuration::getInstance().configClothingData(clothing).name)
{
    Config::ClothingData stats = Configuration::getInstance().configClothingData
(clothing);
    id = static_cast<unsigned int>(clothing);
    minDefense = stats.minDefense;
    maxDefense = stats.maxDefense;
}

unsigned int Clothing::getDefense() const {
    return Calculator::getRandomInt(static_cast<int>(minDefense),
                                    static_cast<int>(maxDefense));
}


```

```
//
// Created by agustin on 9/6/20.
//

#ifndef ARGENTUM_CHEST_H
#define ARGENTUM_CHEST_H

#include "Clothing.h"

class ItemTests;

/*Esta clase encapsula el comportamiento particular de la ropa equipable
 * en el chest*/

class Chest: public Clothing {
private:
    friend ItemTests;

public:
    explicit Chest(GameType::Clothing clothing) : Clothing(clothing) {}

    /*Retorna la posicion donde se equipa (como es un chest retornara la posicio
n del chest)
     * Esto es util para saber en ejecucion donde se equipa el item ya que lo gu
ardamos
     * en el padre*/
    GameType::EquipmentPlace use(Player& player) override;

    /*Esta funcion existe para que cuando el juegador muera y tire todos sus
    items no tire los items default*/
    bool isDefault() const override;
};


#endif //ARGENTUM_CHEST_H
```

```
//
// Created by agustin on 9/6/20.
//

#include "Chest.h"

GameType::EquipmentPlace Chest::use(Player &player) {
    return GameType::EQUIPMENT_PLACE_CHEST;
}

bool Chest::isDefault() const {
    return (id ≡ GameType::COMMON_CLOTHING);
}
```

```cpp
//
// Created by agustin on 6/6/20.
//

#ifndef ARGENTUM_WEAPON_H
#define ARGENTUM_WEAPON_H

#include "../Item.h"
#include "../../../libs/GameEnums.h"

class ItemTests;
class PlayerStats;
struct Coordinate;

/*Esta clase encapsula el comportamiento de las armas*/

class Weapon : public Item {
private:
    int minDamage{};
    int maxDamage{};
    unsigned int attackRange{};
    int manaConsumption{};

    friend ItemTests;

private:
    bool _isTargetReachable(Coordinate attackPosition, Coordinate attackedPositi
on) const;
    void _initializeData(int minDamage, int maxDamage,
                         unsigned int _manaConsumption, unsigned int _range);

public:
    explicit Weapon(GameType::Weapon weapon);

    /*Devuelve el danio que haria el arma por sÃ- sola, es un numero aleatorio
    entre el danio minimo y el maximo
    Recibe las posiciones para determinar si el ataque es realizado o no, si
    no lo realiza retorna 0
    Podria recibir la distancia, pero esto permite encapsular la forma de
    calcular la distancia en la Weapon*/
    int getDamage(Coordinate attackPosition, Coordinate attackedPosition, Player
Stats& stats) const;

    /*Retorna la posicion donde se equipa (como esta es un arma sera en el lugar
 del arma)
     * Esto es util para saber en ejecucion donde se equipa el item ya que lo gu
ardamos
     * en el padre*/
    GameType::EquipmentPlace use(Player& player) override;

    /*Esta funcion existe para que cuando el juegador muera y tire todos sus
    items no tire los items default*/
    bool isDefault();

    ~Weapon() override;
};


#endif //ARGENTUM_WEAPON_H
```

```cpp
//
// Created by agustin on 6/6/20.
//

#include "Weapon.h"
#include "../../Config/Calculator.h"
#include "../../Map/Coordinate.h"
#include "../../Config/Configuration.h"
#include "../../Entities/PlayerStats.h"
#include <ctime>
#include <cstdlib>

/////////////////////////////////////PRIVATE/////////////////////

bool Weapon::_isTargetReachable(Coordinate attackPosition,
                               Coordinate attackedPosition) const {
    unsigned int distance = std::abs(attackPosition.iPosition − attackedPosition
.iPosition) +
                            std::abs(attackPosition.jPosition − attackedPosition
.jPosition);
    return (distance ≠ 0) ∧ (distance ≤ attackRange);
}


void Weapon::_initializeData(int _minDamage, int _maxDamage, unsigned int _manaC
onsumption,
                            unsigned int _range) {
    minDamage = _minDamage;
    maxDamage = _maxDamage;
    manaConsumption = static_cast<int>(_manaConsumption);
    attackRange = _range;
}


/////////////////////////////////////PUBLIC/////////////////////
Weapon::Weapon(GameType::Weapon weapon): Item(GameType::ITEM_TYPE_WEAPON,
                                              Configuration::getInstance().confi
gWeaponData(weapon).name) {
    Config::WeaponData stats = Configuration::getInstance().configWeaponData(wea
pon);
    id = static_cast<unsigned int>(weapon);
    _initializeData(stats.minDmg, stats.maxDmg, stats.manaConsumption, stats.ran
ge);
}

int Weapon::getDamage(Coordinate attackPosition, Coordinate attackedPosition, Pl
ayerStats& stats) const {
    if (¬_isTargetReachable(attackPosition, attackedPosition) ∨ ¬stats.consumeM
ana(manaConsumption)) {
        return 0;
    }
    return Calculator::getRandomInt(minDamage, maxDamage);
}

GameType::EquipmentPlace Weapon::use(Player &player) {
    return GameType::EQUIPMENT_PLACE_WEAPON;
}

Weapon::~Weapon() = default;

bool Weapon::isDefault() {
    return (static_cast<GameType::Weapon>(id) ≡ GameType::Weapon::FIST);
}

```

```
//
// Created by agustin on 8/7/20.
//

#ifndef ARGENTUM_SHOULDPLAYERBEREVIVED_H
#define ARGENTUM_SHOULDPLAYERBEREVIVED_H

#include <sstream>
#include "../Map/Coordinate.h"

struct ResurrectData;
class Map;

//Functor que indica cuando un jugador debe ser eliminado de la lista de jugadores
//a revivir, se debe pasar a remove_if para la ista de monsters
//Almacena en la lista recibida las coordenadas de los monsters que se eliminaron para
//sacarlos despues del mapa
class ShouldPlayerBeRevived {
private:
    Map& map;
    std::stringstream& data;
    double timeStep;

private:
    void _storeResurrectMessage(const ResurrectData& resurrectData);
    void _storeTeleportMessage(const ResurrectData& resurrectData, Coordinate newPosition);

public:
    explicit ShouldPlayerBeRevived(Map& map, std::stringstream& data, double timeStep);

    bool operator()(ResurrectData& resurrectData);
};


#endif //ARGENTUM_SHOULDPLAYERBEREVIVED_H
```

```
//
// Created by agustin on 8/7/20.
//

#include "ShouldPlayerBeRevived.h"

#include "ResurrectData.h"
#include "../../libs/GameEnums.h"
#include "../Entities/Player.h"
#include "../Map/Map.h"
#include <msgpack.hpp>

MSGPACK_ADD_ENUM(GameType::EventID)

ShouldPlayerBeRevived::ShouldPlayerBeRevived(Map& map, std::stringstream &data, double _timeStep)
                                    : map(map), data(data) {
    timeStep = _timeStep;
}

bool ShouldPlayerBeRevived::operator()(ResurrectData &resurrectData) {
    if (¬resurrectData.playerToResurrect→isDead()) {
        return true;
    }
    resurrectData.timeWaited += timeStep;
    Coordinate noFreePositionReturn = {−1, −1};
    Coordinate positionToTeleport{};
    if (resurrectData.timeWaited ≥ resurrectData.timeToWait) {
        positionToTeleport = map.getSpawnCoordinateArroundPosition(resurrectData.resurrectingPriest);
        if (positionToTeleport ≠ noFreePositionReturn) {
            map.moveEntity(resurrectData.playerToResurrect→getPosition(), positionToTeleport);
            resurrectData.playerToResurrect→resetMovement();
            resurrectData.playerToResurrect→restoreStats(true);
            _storeResurrectMessage(resurrectData);
            _storeTeleportMessage(resurrectData, positionToTeleport);
            return true;
        }
    }
    return false;
}

////////////////////////////////PRIVATE//////////////////////////////////

void ShouldPlayerBeRevived::_storeResurrectMessage(const ResurrectData& resurrectData) {
    msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::RESURRECTED);
    msgpack::pack(data, messageTypeData);
    msgpack::type::tuple<std::string> resurrectDataTuple(resurrectData.playerToResurrect→getNickname());
    msgpack::pack(data, resurrectDataTuple);
}

void ShouldPlayerBeRevived::_storeTeleportMessage(const ResurrectData& resurrectData, Coordinate newPosition) {
    msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::TELEPORTED);
    msgpack::pack(data, messageTypeData);
    msgpack::type::tuple<std::string, int32_t, int32_t> resurrectDataTuple
                (resurrectData.playerToResurrect→getNickname(), newPosition.iPosition, newPosition.jPosition);
    msgpack::pack(data, resurrectDataTuple);
}
```

```cpp
//
// Created by agustin on 21/6/20.
//

#ifndef ARGENTUM_SHOULDMONSTERBEREMOVED_H
#define ARGENTUM_SHOULDMONSTERBEREMOVED_H

#include <list>
#include <memory>
#include "../Map/Coordinate.h"

class Monster;

//Functor que indica cuando un monstruo debe ser eliminado de la lista, se debe pasar
//a remove_if para la ista de monsters
//Almacena en la lista recibida las coordenadas de los monsters que se eliminaron para
//sacarlos despues del mapa
class ShouldMonsterBeRemoved {
private:
    std::list<std::pair<Coordinate, const std::string*>>& monstersToRemove;
public:
    explicit ShouldMonsterBeRemoved(std::list<std::pair<Coordinate, const std::string*>>& monstersToRemove);

    //Retorna true si el monstruo esta muerto y guarda su coordenada en la lista
    //almacenada
    bool operator()(const Monster* monster);
};

#endif //ARGENTUM_SHOULDMONSTERBEREMOVED_H
```

```cpp
//
// Created by agustin on 21/6/20.
//

#include "ShouldMonsterBeRemoved.h"

#include "../Entities/Monster.h"

ShouldMonsterBeRemoved::ShouldMonsterBeRemoved(std::list<std::pair<Coordinate, const std::string*>> &monstersToRemove):
                                                monstersToRemove(monstersToRemove) {

}

bool ShouldMonsterBeRemoved::operator()(const Monster* monster) {
    if (monster->isDead()) {
        std::pair<Coordinate, const std::string*> aux(monster->getPosition(), &(monster->getNickname()));
        monstersToRemove.push_back(std::move(aux));
        return true;
    }
    return false;
}
```

```
//
// Created by agustin on 8/7/20.
//

#ifndef ARGENTUM_RESURRECTDATA_H
#define ARGENTUM_RESURRECTDATA_H

#include "../Map/Coordinate.h"

class Player;

//struct auxiliar para guardar la informacion de un player que sera resucitado
struct ResurrectData {
    double timeToWait;
    double timeWaited;
    Coordinate resurrectingPriest;
    Player* playerToResurrect;
};

#endif //ARGENTUM_RESURRECTDATA_H
```

```
//
// Created by agustin on 20/6/20.
//

#ifndef ARGENTUM_MONSTERSFACTORY_H
#define ARGENTUM_MONSTERSFACTORY_H

#include <unordered_map>
#include <vector>
#include <memory>
#include "../../libs/GameEnums.h"
#include "../Map/Coordinate.h"

class GameTests;
class Monster;
class Game;
class Map;

typedef void (*monsterCreator)(Game& game, Coordinate initialPosition,
                               std::shared_ptr<Monster>& monster);

//Esta clase se encarga de crear un monstruo aleatorio, se utiliza para realizar
//los spawns aleatorios de monsters
class MonstersFactory {
private:
    std::unordered_map<GameType::Entity, monsterCreator> monsterCreators;
    std::vector<GameType::Entity> existingMonsters;

    friend GameTests;

private:
    static void _storeSpider(Game& game, Coordinate initialPosition,
                             std::shared_ptr<Monster>& monster);
    static void _storeSkeleton(Game& game, Coordinate initialPosition,
                               std::shared_ptr<Monster>& monster);
    static void _storeZombie(Game& game, Coordinate initialPosition,
                             std::shared_ptr<Monster>& monster);
    static void _storeGoblin(Game& game, Coordinate initialPosition,
                             std::shared_ptr<Monster>& monster);

public:
    MonstersFactory();

    //Guarda en monster un monstruo aleatorio, la coordenada inicial es el {0, 0
}
    void storeRandomMonster(Game& game, std::shared_ptr<Monster>& monster);
};


#endif //ARGENTUM_MONSTERSFACTORY_H
```

```cpp
//
// Created by agustin on 20/6/20.
//



#include "MonstersFactory.h"

#include "../Entities/Monster.h"
#include "../Config/Calculator.h"


/////////////////////////////////////PRIVATE//////////////////////////

void MonstersFactory::_storeSpider(Game& game, Coordinate initialPosition, std::shared_ptr<Monster>& monster) {
    monster = std::make_shared<Monster>(game, initialPosition, GameType::SPIDER, GameType::SPIDER_ATTACK);
}

void MonstersFactory::_storeSkeleton(Game& game, Coordinate initialPosition, std::shared_ptr<Monster>& monster) {
    monster = std::make_shared<Monster>(game, initialPosition, GameType::SKELETON, GameType::SKELETON_ATTACK);
}

void MonstersFactory::_storeZombie(Game& game,Coordinate initialPosition, std::shared_ptr<Monster>& monster) {
        monster = std::make_shared<Monster>(game,initialPosition, GameType::ZOMBIE, GameType::ZOMBIE_ATTACK);
}

void MonstersFactory::_storeGoblin(Game& game, Coordinate initialPosition, std::shared_ptr<Monster>& monster) {
    monster = std::make_shared<Monster>(game, initialPosition, GameType::GOBLIN, GameType::GOBLIN_ATTACK);
}

/////////////////////////////////////PUBLIC//////////////////////////

MonstersFactory::MonstersFactory() {
    monsterCreators[GameType::SPIDER] = _storeSpider;
    monsterCreators[GameType::SKELETON] = _storeSkeleton;
    monsterCreators[GameType::ZOMBIE] = _storeZombie;
    monsterCreators[GameType::GOBLIN] = _storeGoblin;

    for (const auto & creator: monsterCreators) {
        existingMonsters.push_back(creator.first);
    }
}

void MonstersFactory::storeRandomMonster(Game& game, std::shared_ptr<Monster> &monster) {
    //ejecuta la funcion que se almacena en el unordered_map con la key que corresponde a la
    //posicion aleatoria del vector de keys
    monsterCreators[existingMonsters[Calculator::getRandomInt(0, static_cast<int>(existingMonsters.size()) - 1)]]
                                    (game, {0, 0}, monster);
}

```

```cpp
//
// Created by agustin on 7/6/20.
//

#ifndef ARGENTUM_GAME_H
#define ARGENTUM_GAME_H

#include <memory>
#include <queue>
#include "../Map/Map.h"
#include "MonstersFactory.h"
#include "Events/Event.h"
#include "../Items/ItemData.h"
#include "ResurrectData.h"
#include "../../libs/Timer.h"

class EntityTests;
struct PlayerData;

class PlayerShouldBeRemoved {
private:
    Player* playerToRemove;

public:
    explicit PlayerShouldBeRemoved(Player* player) : playerToRemove(player) {}
    bool operator()(const Player* player);
};

struct MoveCommand {
    Coordinate initialPosition;
    Coordinate finalPosition;
    bool isTeleporting;
};

//Esta clase se encarga de manejar en forma general las acciones que quiere realizar
//cada identidad
class Game {
private:
    std::list<Coordinate> priests;
    Map map;
    std::queue<std::unique_ptr<Event>> eventQueue;

    unsigned int monsterCreationRate;
    unsigned int maxNumberOfMonsters;
    unsigned int  spawnInterval;
    Timer monsterSpawnTimer;
    MonstersFactory monstersFactory;

    std::list<Monster*> monsters;
    std::unordered_map<std::string, Player*> players;
    std::unordered_map<Coordinate, const Item*> mapItems;

    std::list<ResurrectData> playersToResurrect;


    friend GameTests;
    friend EntityTests;

private:

    void _removeMonsters(ServerProtocol& protocol);
    void _updateMonsters(double timeStep);
    void _updatePlayers(double timeStep);
    void _executeQueueOperations(ServerProtocol& protocol);
    void _repopulateMap(ServerProtocol& protocol);
    void _updateDeadPlayersTimer(ServerProtocol& protocol, double timestep);

public:

    //Este constructor debe ser utilizado unicamente para las pruebas
    //explicit Game(ClientsMonitor&& clientAux = ClientsMonitor()){};

    explicit Game(MapFileReader∧ mapFile);

    //Delega a map el ataque a la coordenada recibida, retorna una instancia de AttackResult junto
    //con un bool que esta en true si se realizo un ataque, sino retorna false
    std::pair<AttackResult, bool> attackPosition(int damage, unsigned int level, bool isAPlayer,
                                    Coordinate coordinate);

    //Llama a Map para que guarde los items recibidos en el tile que corresponde
    //a la coordenada recibida
    void dropItems(std::list<std::shared_ptr<Item>>∧ items, Coordinate position);

    //Llama a Map para que guarde el item recibido en el tile que corresponde
    //a la coordenada recibida
    void dropItems(std::shared_ptr<Item>∧ item, Coordinate position);

    //Retorna una referencia constante del mapa, util para los monstruos
    const Map& getMap() const;

    //Ejecuta un update del juego, realiza todas las acciones que debe realizar el juego
    //en este tick
    void update(double timeStep, ServerProtocol& protocol);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    //unsigned int list(Player& player, std::list<ProductData>& products, Coordinate coordinate);
    void list(Player& player, Coordinate coordinate);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void withdraw(Player& player, const std::string& itemName, Coordinate coordinate);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void deposit(Player& player, const std::string& itemName, Coordinate coordinate);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void buy(Player& player, const std::string& itemName, Coordinate coordinate);

    //Delega el comportamiento a la entity que guarda, si es que guarda una
    void sell(Player& player, const std::string& itemName, Coordinate coordinate);

    //Delega a map el movimiento de la entidad que se encuentra en initialPosition
    //y debe terminar en finalPosition
    void moveEntity(Coordinate initialPosition, Coordinate finalPosition);

    //Game se apropia del puntero al evento, agregandolo a la cola de enentos que
    //despues sera vaciada para ejecutar las acciones del update. Este metodo es
    //virtual para que Fakeit pueda redefinirlo a que no haga nada porque sino la
    //prueba segmentea (el stub no reserva memoria para la cola de eventos creemos)
```

```
    virtual void pushEvent(std::unique_ptr<Event>∧ event);

    //Crea el player en base al nickname, raza y clase que recibe
    Player& createPlayer(PlayerData& playerData, ServerProtocol& protocol);

    //Carga en el protocolo el estado actual del juego para mandar a un jugador
    //que se conecta y poder mandarle luego solo los cambios en cada update
    const std::vector<char>& getCurrentState(ServerProtocol& protocol);

    //Elimina al jugador del juego, eliminandolo de todos los lugares en los que
    //esta guardado, guarda en el protocolo el mensaje que comunica al resto de los
    //clientes que desaparecio un jugador
    void removePlayer(const std::string& playerNickname, ServerProtocol& protocol);

    //Intenta guardar el item en el inventario del player, retorna el puntero al
    //item que se encuentra al final de la lista de items guardada en el tile del cual
    //se agarro el item, si no hay mas items despues de agarrar uno entonces retorna nullptr
    const Item* storeItemFromTileInPlayer(Player& player);

    //Resusita el player instantaneamente si la coordenada que selecciono contiene un priest, sino
    //guarda su informacion en la lista de players a resucitar para resucitarlo cuando pase la cantidad
    //de tiempo necesaria
    //Si lo resucita instantaneamente retorna true, sino retorna false
    bool requestResurrect(Player& player, Coordinate selectedPosition);

    //Envia un mensaje a otro jugador
    void messagePlayer(Player& playerWhoMessaged, const std::string& playerToMessage, const std::string& message);


    //Retorna true si el player se encuentra en el game, false en caso contrario
    bool playerExists(const std::string& nickname) const;

    //Delega a map el pedido del player al entity de la coordenada target para
    //que se restore su mana y vida
    void requestRestore(Player& player, Coordinate target);

    //Guarda en playerData los items que tiene guardados en el banker el player
    //que tiene el nickname guardado en playerData
    static void getPlayerBank(PlayerData& playerData) ;
};


#endif //ARGENTUM_GAME_H
```

```cpp
//
// Created by agustin on 7/6/20.
//

#include <algorithm>
#include "Game.h"
#include "../Entities/AttackResult.h"
#include "../Entities/Monster.h"
#include "ShouldMonsterBeRemoved.h"
#include "Events/Event.h"
#include "../Server/ServerProtocol.h"
#include "../Entities/Player.h"
#include "ShouldPlayerBeRevived.h"
#include <iostream>
#include "../Server/PlayerData.hpp"
#include "../Entities/Citizens/Banker.h"
#include "../Config/Configuration.h"

MSGPACK_ADD_ENUM(GameType::EventID)

#define WAITING_TIME_MESSAGE "The estimated waiting time to resurrect is "
#define NO_PLAYER_MESSAGE "That player does not exist or is not connected\n"
#define WELCOME_MESSAGE "Welcome to Argentum traveller!\nWe hope you enjoy our work\nSincerely, AIM Team\n"
const Coordinate defaultSpawnPoint = {88,83};


////////////////////////////////PRIVATE////////////////////////


//Carga hasta monsterCreationRate monstruos nuevos cada cierto invervalo de tiempo
//Si la cantidad que se desea crear sobrepasa la cantidad maxima, entonces crea hasta
//conseguir la cantidad maxima
void Game::_repopulateMap(ServerProtocol& protocol) {
    Coordinate aux{};
    std::stringstream data;
    double timePassed = monsterSpawnTimer.getTime();
    if (timePassed ≥ spawnInterval) {
        unsigned int monstersToCreate = monsterCreationRate;
        std::shared_ptr<Monster> monster;
        monsterSpawnTimer.start();
        if ((monstersToCreate + monsters.size()) > maxNumberOfMonsters) {
            monstersToCreate = maxNumberOfMonsters − monsters.size();
        }
        for (unsigned int i = 0; i < monstersToCreate; ++i) {
            monstersFactory.storeRandomMonster(*this, monster);
            aux = map.getMonsterCoordinate();
            monster→setPosition(aux);
            (*monster) >> data;
            monsters.push_back(monster.get());
            map.addEntity(aux, std::static_pointer_cast<Entity>(monster));
        }
        protocol.addToGeneralData(data);
    }
}

//Vacia la cola de operaciones a realizar, ejecutando cada operacion que es
//desencolada
void Game::_executeQueueOperations(ServerProtocol& protocol) {
    while (¬eventQueue.empty()) {
        (*eventQueue.front())(protocol);
        eventQueue.pop();
    }
}
```

```cpp

void Game::moveEntity(Coordinate initialPosition, Coordinate finalPosition) {
    map.moveEntity(initialPosition, finalPosition);
}

//Llama a update de todos los monstruos que se encuentran en el mapa, haciendo
//que tomen una decision
void Game::_updateMonsters(double timeStep) {
    for (const auto & monster: monsters) {
        monster→update(timeStep);
    }
}

//Hace un update de los players conectados, actualizando su vida, mana y distancia
//recorrida
void Game::_updatePlayers(double timeStep) {
    for (const auto & player: players) {
        player.second→update(timeStep);
    }
}

//Elimina de las listas almacenadas y del mapa los monsters que deban ser eliminados
//y guarda en el protocolo el mensaje de que estos deben desaparecer para mandar al cliente
void Game::_removeMonsters(ServerProtocol& protocol) {
    std::stringstream data;
    std::list<std::pair<Coordinate, const std::string*>> monstersToRemove;
    ShouldMonsterBeRemoved sholdBeRemoved(monstersToRemove);
    monsters.erase(std::remove_if(monsters.begin(), monsters.end(), sholdBeRemoved), monsters.end());
    for (const auto & monster: monstersToRemove) {
        msgpack::type::tuple<GameType::EventID> eventIdData(GameType::EventID::REMOVE_ENTITY);
        msgpack::pack(data, eventIdData);
        msgpack::type::tuple<std::string>
                removedMonsterNickname(*monster.second);
        msgpack::pack(data, removedMonsterNickname);
        protocol.addToGeneralData(data);
        map.removeEntity(monster.first);
    }
}

//Itera la lista de players muertos que estan esperando resucitar, aplicando la funcion
//shouldBeRevived y eliminando asi los players que son revividos
void Game::_updateDeadPlayersTimer(ServerProtocol& protocol, double timestep) {
    std::stringstream data;
    ShouldPlayerBeRevived shouldBeRevived(map, data, timestep);
    playersToResurrect.erase(std::remove_if(playersToResurrect.begin(), playersToResurrect.end(), shouldBeRevived),
                             playersToResurrect.end());
    protocol.addToGeneralData(data);
}


////////////////////////////////PUBLIC////////////////////////

std::pair<AttackResult, bool> Game::attackPosition(int damage, unsigned int level, bool isAPlayer,
                                    Coordinate coordinate) {
    return map.attackTile(damage, level, isAPlayer, coordinate);
}

void Game::dropItems(std::list<std::shared_ptr<Item>>∧ items, Coordinate positi
on) {
    if (items.empty()) {
        throw std::invalid_argument("Received empty list in Game::dropItems");
    }
    mapItems[position] = items.back().get();
    map.addItemsToTile(std::move(items), position);
}

void Game::dropItems(std::shared_ptr<Item> ∧item, Coordinate position) {
    if (¬item) {
        throw std::invalid_argument("Received null item in Game::dropItems");
    }
    mapItems[position] = item.get();
    map.addItemsToTile(std::move(item), position);
}

void Game::update(double timeStep, ServerProtocol& protocol) {
    _repopulateMap(protocol);
    _updateMonsters(timeStep);
    _updatePlayers(timeStep);
    _executeQueueOperations(protocol);
    _removeMonsters(protocol);
    _updateDeadPlayersTimer(protocol, timeStep);
}

Game::Game(MapFileReader∧ mapFile): priests(),  map(mapFile, priests) {
    Configuration& config = Configuration::getInstance();
    monsterCreationRate = config.configMonsterSpawnAmount();
    maxNumberOfMonsters = config.configMaxMonsterAmount();
    spawnInterval = config.configTimeBetweenMonsterSpawns();
    monsterSpawnTimer.start();
}

const Map& Game::getMap() const {
    return map;
}

void Game::list(Player &player, Coordinate coordinate) {
    map.list(player, coordinate);
}

void Game::withdraw(Player &player, const std::string &itemName, Coordinate coor
dinate) {
    map.withdraw(player, itemName, coordinate);
}

void Game::deposit(Player &player, const std::string &itemName, Coordinate coord
inate) {
    map.deposit(player, itemName, coordinate);
}

void Game::buy(Player &player, const std::string &itemName, Coordinate coordinat
e) {
    map.buy(player, itemName, coordinate);
}

void Game::sell(Player &player, const std::string &itemName, Coordinate coordina
te) {
    map.sell(player, itemName, coordinate);
}

void Game::pushEvent(std::unique_ptr<Event>∧ event) {
    eventQueue.push(std::move(event));
}

Player& Game::createPlayer(PlayerData& playerData, ServerProtocol& protocol) {
```

```
    Coordinate spawnPosition{};
    if (¬priests.empty()) {
        spawnPosition = map.getSpawnCoordinateArroundPosition(priests.front());
    } else {
        spawnPosition = map.getSpawnCoordinateArroundPosition(defaultSpawnPoint)
;
    }
    Banker::addPlayerItems(playerData);
    auto player = std::make_shared<Player>(*this, spawnPosition, playerData);
    Player* playerAux = player.get();
    players.emplace(playerAux→getNickname(), playerAux);
    player→addMessage(WELCOME_MESSAGE);
    map.addEntity(spawnPosition, std::move(player));
    std::stringstream data;
    (*playerAux) >> data;
    protocol.addToGeneralData(data);
    return *playerAux;
}

const std::vector<char>& Game::getCurrentState(ServerProtocol& protocol) {
    return protocol.buildCurrentState(players, monsters,  mapItems);
}

void Game::removePlayer(const std::string& playerNickname, ServerProtocol& proto
col) {
    std::stringstream data;
    msgpack::type::tuple<GameType::EventID> eventIdData(GameType::EventID::REMOV
E_ENTITY);
    msgpack::pack(data, eventIdData);
    msgpack::type::tuple<std::string>
            removedPlayerNickname(playerNickname);
    msgpack::pack(data, removedPlayerNickname);
    protocol.addToGeneralData(data);
    Coordinate playerPosition = players.at(playerNickname)→getPosition();
    players.erase(playerNickname);
    Banker::erasePlayerItems(playerNickname);
    map.removeEntity(playerPosition);
}

const Item* Game::storeItemFromTileInPlayer(Player& player) {
    Coordinate playerPosition = player.getPosition();
    std::shared_ptr<Item> retreivedItem = map.removeItem(playerPosition);
    const Item* returnData = nullptr;
    if (retreivedItem) {
        if (¬player.storeItem(retreivedItem)) {
            returnData = retreivedItem.get();
            map.addItemsToTile(std::move(retreivedItem), playerPosition);
        } else {
            returnData = map.peekShowedItemData(playerPosition);
            if (returnData) {
                mapItems[{playerPosition.iPosition, playerPosition.jPosition}] =
 returnData;
            } else {
                mapItems.erase({playerPosition.iPosition, playerPosition.jPositi
on});
            }
        }
    }
    return returnData;
}

bool Game::requestResurrect(Player &player, Coordinate selectedPosition) {
    if (priests.empty() ∨ ¬player.isDead()) {
        return false;
    }
    Coordinate playerPosition = player.getPosition();
```

```cpp
    Coordinate nearestPriest = priests.front();
    for (const auto & priestPosition: priests) {
        if (selectedPosition ≡ priestPosition) {
            player.restoreStats(true);
            return true;
        }
        if (playerPosition.calculateDistance(priestPosition) <
                playerPosition.calculateDistance(nearestPriest)) {
            nearestPriest = priestPosition;
        }
    }
    //Por cada tile de distancia espera 200ms
    auto waitingTime = static_cast<double>(playerPosition.calculateDistance(nearestPriest) * 200);
    player.addMessage(WAITING_TIME_MESSAGE);
    player.addMessage(std::to_string(static_cast<int>(waitingTime/1000)) + " seconds\n");
    playersToResurrect.push_back({waitingTime, 0, nearestPriest, &player});
    return false;
}

void Game::messagePlayer(Player& playerWhoMessaged, const std::string &playerToMessage, const std::string &message) {
    if (players.count(playerToMessage) ≡ 1) {
        Player* player = players.at(playerToMessage);
        player→addMessage(message);
    } else {
        playerWhoMessaged.addMessage(NO_PLAYER_MESSAGE);
    }
}

bool Game::playerExists(const std::string &nickname) const {
    return (players.count(nickname) ≡ 1);
}

void Game::requestRestore(Player& player, Coordinate target) {
    map.requestRestore(player, target);
}

void Game::getPlayerBank(PlayerData &playerData) {
    Banker::getPlayerItems(playerData);
}

bool PlayerShouldBeRemoved::operator()(const Player* player) {
    return (playerToRemove ≡ player);
}
```

```cpp
//
// Created by agustin on 23/6/20.
//

#ifndef ARGENTUM_WITHDRAW_H
#define ARGENTUM_WITHDRAW_H


#include "Event.h"
#include <string>
#include "../../Map/Coordinate.h"

class Player;

//Clase que se almacena en la cola de eventos cuando un player quiere ejecutar
//el comando withdraw
class Withdraw : public Event {
private:
    Player& player;
    std::string itemName;
    Coordinate npcPosition{};

public:
    Withdraw(Player& player, std::string∧ _itemName, Coordinate _npcPosition);

    //Intenta llamar a withdraw en la posicion guardada en el constructor, pidiendole
    //a map que realice esa accion
    void operator()(ServerProtocol& protocol) override;
};

#endif //ARGENTUM_WITHDRAW_H
```

```
//
// Created by agustin on 23/6/20.
//

#include "Withdraw.h"
#include "../../Entities/Player.h"

Withdraw::Withdraw(Player &_player, std::string^ _itemName, Coordinate _npcPosition) : player(_player) {
    itemName = std::move(_itemName);
    npcPosition = _npcPosition;
}

void Withdraw::operator()(ServerProtocol& protocol) {
    player.withdrawFrom(itemName, npcPosition);
}
```

```
//
// Created by agustin on 23/6/20.
//

#ifndef ARGENTUM_USEITEM_H
#define ARGENTUM_USEITEM_H


#include "Event.h"

class Player;

//Clase que se almacena en la cola de eventos cuando un player quiere usar
//un item
class UseItem : public Event {
private:
    Player& player;
    int position;

public:
    UseItem(Player& player, int position);

    //Intenta usar el item del lugar recibido en el constructor, en caso de ser
    //necesario, le comuinica al resto de los clientes si algun (y cual) item fue equipado
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_USEITEM_H
```

```cpp
//
// Created by agustin on 23/6/20.
//

#include "UseItem.h"
#include "../../Entities/Player.h"
#include <msgpack.hpp>
#include "../../Server/ServerProtocol.h"

MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::EquipmentPlace)

UseItem::UseItem(Player &player, int _position): player(player) {
    position = _position;
}

void UseItem::operator()(ServerProtocol& protocol) {
    UseReturnData useData = player.useItem(position);
    if (useData.equipmentPlace ≠ GameType::EQUIPMENT_PLACE_NONE) {
        std::stringstream data;
        msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::EQUIPPED);
        msgpack::pack(data, messageTypeData);
        msgpack::type::tuple<std::string, GameType::EquipmentPlace, int32_t> useDataTuple
                                (player.getNickname(), useData.equipmentPlace, useData.id);
        msgpack::pack(data, useDataTuple);
        protocol.addToGeneralData(data);
    }
}
```

```cpp
//
// Created by agustin on 23/6/20.
//

#ifndef ARGENTUM_UNEQUIP_H
#define ARGENTUM_UNEQUIP_H


#include "Event.h"
#include "../../Items/Defense/Clothing.h"
#include "../../../libs/GameEnums.h"

//Clase que se almacena en la cola de eventos cuando un player quiere desequipar se
//un item
class Unequip : public Event {
private:
    Player& player;
    GameType::EquipmentPlace equipment;

public:
    Unequip(Player& player, GameType::EquipmentPlace equipment);

    //Intenta desequipar el item del lugar recibido en el constructor, si logra
    //hacerlo entonces le comunica a los clientes que ese item ya no se encuentra
    //equipado, agregandolo al protocolo, si el equipment place es el del pecho,
    //entonces notifica tambien que el player se equipo ropa default
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_UNEQUIP_H
```

```
//
// Created by agustin on 23/6/20.
//

#include "Unequip.h"
#include "../../Entities/Player.h"
#include "../../Items/Item.h"
#include "../../Server/ServerProtocol.h"
#include "msgpack.hpp"

MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::EquipmentPlace)

Unequip::Unequip(Player &player, GameType::EquipmentPlace _equipment): player(pl
ayer) {
    equipment = _equipment;
}

void Unequip::operator()(ServerProtocol& protocol) {
    bool hasAppearanceChanged;
    if (equipment ≡ GameType::EQUIPMENT_PLACE_WEAPON) {
        hasAppearanceChanged = player.unequip();
    } else {
        hasAppearanceChanged = player.unequip(equipment);
    }
    if (hasAppearanceChanged) {
        std::stringstream data;
        msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::UNEQUI
P);
        msgpack::pack(data, messageTypeData);
        msgpack::type::tuple<std::string, GameType::EquipmentPlace>
                unequipData(player.getNickname(), equipment);
        msgpack::pack(data, unequipData);
        if (equipment ≡ GameType::EQUIPMENT_PLACE_CHEST) {
            msgpack::type::tuple<GameType::EventID> messageTypeEquipData(GameTyp
e::EQUIPPED);
            msgpack::pack(data, messageTypeEquipData);
            msgpack::type::tuple<std::string, GameType::EquipmentPlace, int32_t>
 useDataTuple
                    (player.getNickname(), GameType::EQUIPMENT_PLACE_CHEST, Game
Type::COMMON_CLOTHING);
            msgpack::pack(data, useDataTuple);
        }
        protocol.addToGeneralData(data);
    }
}
```

```
//
// Created by agustin on 23/6/20.
//

#ifndef ARGENTUM_SELL_H
#define ARGENTUM_SELL_H


#include "Event.h"
#include "../../Map/Coordinate.h"


class Player;

//Clase que se almacena en la cola de eventos cuando un player quiere ejecutar
//el comando sell
class Sell: public Event {
private:
    Player& player;
    Coordinate position{};
    std::string itemName;
public:
    Sell(Player& player, std::string∧ itemName, Coordinate position);

    //Intenta llamar a sell en la posicion guardada en el constructor, pidiendol
e
    //a map que realice esa accion
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_SELL_H
```

```cpp
//
// Created by agustin on 23/6/20.
//

#include "Sell.h"
#include "../../Entities/Player.h"

Sell::Sell(Player &player, std::string^ _itemName, Coordinate _position):
            player(player), itemName(std::move(_itemName)) {
    position = _position;
}

void Sell::operator()(ServerProtocol& protocol) {
    player.sellTo(itemName, position);
}
```

```cpp
//
// Created by agustin on 10/7/20.
//

#ifndef ARGENTUM_RESTORESTATS_H
#define ARGENTUM_RESTORESTATS_H


#include "Event.h"
#include "../../Map/Coordinate.h"

class Player;
class Game;

//Clase que se almacena en la cola de eventos cuando un player quiere ejecutar
//el comando heal
class RestoreStats: public Event {
private:
    Game& game;
    Player& player;
    Coordinate target;

public:
    RestoreStats(Game& game, Player& player, Coordinate target);

    //Pide a game que llame a restore para el mapa en la coordenada recibida
    void operator()(ServerProtocol& serverProtocol) override;
};


#endif //ARGENTUM_RESTORESTATS_H
```

```cpp
//
// Created by agustin on 10/7/20.
//

#include "RestoreStats.h"

#include "../../Entities/Player.h"
#include "../Game.h"

RestoreStats::RestoreStats(Game &game, Player &player, Coordinate _target): game
(game), player(player) {
    target = _target;
}

void RestoreStats::operator()(ServerProtocol &serverProtocol) {
    game.requestRestore(player, target);
}
```

```cpp
//
// Created by agustin on 8/7/20.
//

#ifndef ARGENTUM_REQUESTRESURRECT_H
#define ARGENTUM_REQUESTRESURRECT_H


#include "Event.h"
#include "../../Map/Coordinate.h"

class Player;
class Game;

//Clase que se almacena en la cola de eventos cuando un player quiere ejecutar
//el comando resurrect
class RequestResurrect: public Event {
private:
    Player& player;
    Game& game;
    Coordinate selectedPosition{};

public:
    RequestResurrect(Game& game, Player& player, Coordinate selectedPosition);

    //Pide a game que reviva al player, mandandole la coordenada a la que player
    //le hace el pedido, si es resucitado al realizarlo entonces se agrega esta
    //informacion al protocolo para que le llegue a todos los clientes y sepan
    //que el player esta vivo
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_REQUESTRESURRECT_H
```

```
//
// Created by agustin on 8/7/20.
//

#include "RequestResurrect.h"

#include "../../Entities/Player.h"
#include "../Game.h"
#include "../../Server/ServerProtocol.h"
#include <msgpack.hpp>
#include "../../Server/ServerProtocol.h"

MSGPACK_ADD_ENUM(GameType::EventID)

RequestResurrect::RequestResurrect(Game& game, Player& player, Coordinate _selec
tedPosition)
                                        : player(player), game(game) {
    selectedPosition = _selectedPosition;
}

void RequestResurrect::operator()(ServerProtocol &protocol) {
    if (game.requestResurrect(player, selectedPosition)) {
        std::stringstream data;
        msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::RESURR
ECTED);
        msgpack::pack(data, messageTypeData);
        msgpack::type::tuple<std::string> playerData(player.getNickname());
        msgpack::pack(data, playerData);
        protocol.addToGeneralData(data);
    }
}
```

```
//
// Created by agustin on 20/7/20.
//

#ifndef ARGENTUM_PLAYERLEVELEDUP_H
#define ARGENTUM_PLAYERLEVELEDUP_H


#include "Event.h"

#include <string>

class PlayerLeveledUp: public Event {
private:
    const std::string& playerNickname;
    int32_t level;

public:
    explicit PlayerLeveledUp(const std::string& playerNickname, int32_t level);

    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_PLAYERLEVELEDUP_H
```

```
//
// Created by agustin on 20/7/20.
//

#include "PlayerLeveledUp.h"

#include "../../../libs/GameEnums.h"
#include "../../Server/ServerProtocol.h"
#include <msgpack.hpp>

MSGPACK_ADD_ENUM(GameType::EventID)

PlayerLeveledUp::PlayerLeveledUp(const std::string &playerNickname, int32_t _level):
                                 playerNickname(playerNickname) {
    level = _level;
}

void PlayerLeveledUp::operator()(ServerProtocol &protocol) {
    std::stringstream data;
    msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::PLAYER_LEVEL_UP);
    msgpack::pack(data, messageTypeData);
    msgpack::type::tuple<std::string, int32_t> nicknameData(playerNickname, level);
    msgpack::pack(data, nicknameData);
    protocol.addToGeneralData(data);
}
```

```
//
// Created by agustin on 6/7/20.
//

#ifndef ARGENTUM_PICKUPITEM_H
#define ARGENTUM_PICKUPITEM_H


#include "Event.h"

class Game;
class Player;
class ServerProtocol;

//Clase que se almacena en la cola de eventos cuando un player quiere ejecutar
//el comando pickUp
class PickUpItem: public Event {
private:
    Game& game;
    Player& player;

public:
    PickUpItem(Game& game, Player& player);

    //Le dice a game que el player quiere agarrar el item que se encuentra en
    //su posicion, si lo logra entonces manda al cliente el mensaje necesario:
    //Manda que se debe destruir el item de esa posicion si es que ya no quedan
    //mas items, sino manda que se debe crear el item que se quiere mostrar en
    //esa posicion
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_PICKUPITEM_H
```

```cpp
//
// Created by agustin on 6/7/20.
//

#include "PickUpItem.h"
#include "../../Entities/Player.h"
#include "../Game.h"
#include "../../Server/ServerProtocol.h"
#include <msgpack.hpp>

MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::ItemType)

PickUpItem::PickUpItem(Game &game, Player &player): game(game), player(player) {

}

void PickUpItem::operator()(ServerProtocol& protocol) {
    const Item* itemPtr = game.storeItemFromTileInPlayer(player);
    std::stringstream data;
    Coordinate pickUpPosition = player.getPosition();
    if (itemPtr) {
        itemPtr->loadDropItemData(data, pickUpPosition.iPosition, pickUpPosition.jPosition);
        protocol.addToGeneralData(data);
    } else {
        msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::DESTROY_ITEM);
        msgpack::pack(data, messageTypeData);
        msgpack::type::tuple<int32_t, int32_t> itemDataTuple
                (pickUpPosition.iPosition, pickUpPosition.jPosition);
        msgpack::pack(data, itemDataTuple);
        protocol.addToGeneralData(data);
    }
}
```

```cpp
//
// Created by agustin on 5/7/20.
//

#ifndef ARGENTUM_NOTIFYDEATH_H
#define ARGENTUM_NOTIFYDEATH_H


#include "Event.h"

#include <sstream>

class Player;

//Clase que se almacena en la cola de eventos para notificarle a los clientes que
//un player murio, por lo que ahora es un fantasma
class NotifyDeath: public Event {
private:
    const Player& player;

private:
    void _appendUnequipMessages(std::stringstream& data);

public:
    explicit NotifyDeath(const Player& player);

    //Guarda en el protocolo los mensajes de muerte y desequipamiento de items,
    //ademas del equipamiento de los items default, para que se envie a todos los
    //clientes
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_NOTIFYDEATH_H
```

```
//
// Created by agustin on 5/7/20.
//

#include "NotifyDeath.h"

#include "../../Entities/Player.h"
#include "../../../libs/GameEnums.h"
#include "../../Server/ServerProtocol.h"
#include <msgpack.hpp>

MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::EquipmentPlace)

NotifyDeath::NotifyDeath(const Player &player): player(player) {

}

void NotifyDeath::operator()(ServerProtocol &protocol) {
    std::stringstream data;
    msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::PLAYER_DEATH);
    msgpack::pack(data, messageTypeData);
    msgpack::type::tuple<std::string> nicknameData(player.getNickname());
    msgpack::pack(data, nicknameData);
    _appendUnequipMessages(data);
    protocol.addToGeneralData(data);
}

////////////////////////////////PRIVATE/////////////////////////

void NotifyDeath::_appendUnequipMessages(std::stringstream& data) {
    std::vector<GameType::EquipmentPlace> equipment = {GameType::EQUIPMENT_PLACE_WEAPON,
          GameType::EQUIPMENT_PLACE_SHIELD, GameType::EQUIPMENT_PLACE_HEAD,
          GameType::EQUIPMENT_PLACE_CHEST};

    for (const auto & place: equipment) {
        msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::UNEQUIP);
        msgpack::pack(data, messageTypeData);
        msgpack::type::tuple<std::string, GameType::EquipmentPlace>
                unequipmentData(player.getNickname(), place);
        msgpack::pack(data, unequipmentData);
    }
    msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::EQUIPPED);
    msgpack::pack(data, messageTypeData);
    msgpack::type::tuple<std::string, GameType::EquipmentPlace, int32_t>
            equippedData(player.getNickname(), GameType::EQUIPMENT_PLACE_CHEST,
                    GameType::COMMON_CLOTHING);
    msgpack::pack(data, equippedData);
}
```

```
//
// Created by agustin on 23/6/20.
//

#ifndef ARGENTUM_MOVE_H
#define ARGENTUM_MOVE_H


#include "Event.h"
#include "../../Map/Coordinate.h"
#include "../../../libs/GameEnums.h"

class Game;
class Entity;

//Clase que se almacena en la cola de eventos cuando un entity quiere concretar su
//desplazamiento en alguna direccion
class Move: public Event {
private:
    Game& game;
    Entity& entity;
    GameType::Direction moveDirection;

public:
    Move(Game& _game, Entity& _entity, GameType::Direction moveDirection);

    //Intenta mover el entity guardado en la direccion recibida en el constructor,
    //pasandola inmediatamente al tile si es que esta disponible para el entity,
    //empezando asi su desplazamiento "visual" hacia este
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_MOVE_H
```

```cpp
//
// Created by agustin on 29/6/20.
//

#ifndef ARGENTUM_MOVED_H
#define ARGENTUM_MOVED_H


#include "Event.h"
#include "../../../libs/GameEnums.h"

#include <cstdint>

class Entity;
class ServerProtocol;

//Clase que se almacena en la cola de eventos para notificarle a los clientes que
//un entity se desplazo
class Moved: public Event {
private:
    Entity& entity;
    GameType::Direction direction;
    int32_t displacement;

public:
    Moved(Entity& entity, GameType::Direction direction, int32_t displacement);

    //Almacena en el protocolo el mensaje del desplazamiento de un entity para
    //comunicarselo a todos los clientes
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_MOVED_H
```

```cpp
//
// Created by agustin on 29/6/20.
//

#include "Moved.h"
#include "../../Entities/Entity.h"
#include "../../Server/ServerProtocol.h"
#include <msgpack.hpp>

MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::Direction)

Moved::Moved(Entity &entity, GameType::Direction _direction, int32_t _displacement):
             entity(entity) {
    direction = _direction;
    displacement = _displacement;
}

void Moved::operator()(ServerProtocol &protocol) {
    std::stringstream data;
    msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::MOVED);
    msgpack::pack(data, messageTypeData);
    bool hasFinishedMoving;
    int32_t realDisplacement = entity.executeDisplacement(displacement, hasFinishedMoving);
    msgpack::type::tuple<GameType::Direction, int32_t, std::string, bool>
                        eventData(direction, realDisplacement, entity.getNickname(), hasFinishedMoving);
    msgpack::pack(data, eventData);
    protocol.addToGeneralData(data);
}
```

```
//
// Created by agustin on 23/6/20.
//

#include "Move.h"
#include "../../Entities/Entity.h"
#include "../Game.h"

Move::Move(Game &_game, Entity &_entity, GameType::Direction _moveDirection) :
                                                  game(_game), entity(_entity) {
    moveDirection = _moveDirection;
}

void Move::operator()(ServerProtocol& protocol) {
    game.moveEntity(entity.getPosition(), entity.getFinalCoordinate(moveDirection));
}
```

```
//
// Created by agustin on 14/7/20.
//

#ifndef ARGENTUM_MODIFYPLAYERMOVEMENT_H
#define ARGENTUM_MODIFYPLAYERMOVEMENT_H


#include "Event.h"
#include "../../../libs/GameEnums.h"

class Player;

//Clase que se almacena en la cola de eventos cuando un player quiere desplazarse
//en alguna direccion
class ModifyPlayerMovement: public Event {
private:
    Player& player;
    GameType::Direction direction{};
    bool continuesMovement;

public:
    //Constructor que le indica a movement que se debe empezar a mover en la direccion
    //recibida
    explicit ModifyPlayerMovement(Player& player, GameType::Direction direction);

    //constructor que le indica a movement que debe dejar de moverse
    explicit ModifyPlayerMovement(Player& player);

    //Guarda el proximo estado de movimiento para que sea ejecutado una vez que
    //termine de realizar el movimiento que se este realizando en el momento
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_MODIFYPLAYERMOVEMENT_H
```

```
//
// Created by agustin on 14/7/20.
//

#include "ModifyPlayerMovement.h"

#include "../../Entities/Player.h"

ModifyPlayerMovement::ModifyPlayerMovement(Player& player, GameType::Direction _
direction): player(player) {
    direction = _direction;
    continuesMovement = true;
}

ModifyPlayerMovement::ModifyPlayerMovement(Player& player): player(player) {
    continuesMovement = false;
}

void ModifyPlayerMovement::operator()(ServerProtocol& protocol) {
    if (continuesMovement) {
        player.startMovement(direction);
    } else {
        player.stopMovement();
    }
}
```

```
//
// Created by marcos on 7/10/20.
//

#ifndef ARGENTUM_MESSAGE_H
#define ARGENTUM_MESSAGE_H

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

class Game;
class Player;

//Clase que se almacena en la cola de eventos cuando un player quiere ejecutar
//el comando message
class Message : public Event {
private:
    Game& game;
    Player& playerWhoMessaged;
    std::string playerToMessage;
    std::string message;

public:
    Message(Game& _game, Player& _playerWhoMessaged, std::string∧ _playerToMess
age,
            std::string∧ _message);

    //Llama a la funcion de game que manda el mensaje a otro jugador, agregandos
elo
    //a su minichat
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_MESSAGE_H
```

```cpp
//
// Created by marcos on 7/10/20.
//

#include "Message.h"
#include "../../Entities/Player.h"
#include "../Game.h"

void Message::operator()(ServerProtocol &protocol) {
    game.messagePlayer(playerWhoMessaged, playerToMessage, message);
}

Message::Message(Game &_game, Player &_playerWhoMessaged,
                 std::string ^_playerToMessage, std::string ^_message)
                 : game(_game), playerWhoMessaged(_playerWhoMessaged), playerToMessage(std::move(_playerToMessage)) {

    message = _playerWhoMessaged.getNickname() + ":" + _message;
}
```

```cpp
//
// Created by agustin on 23/6/20.
//

#ifndef ARGENTUM_LIST_H
#define ARGENTUM_LIST_H


#include "Event.h"
#include <string>
#include "../../Map/Coordinate.h"

class Player;

//Clase que se almacena en la cola de eventos cuando un player quiere ejecutar
//el comando list
class List : public Event {
private:
    Player& player;
    Coordinate npcPosition{};

public:
    List(Player& player, Coordinate _npcPosition);

    //Intenta llamar a list en la posicion guardada en el constructor, pidiendole
    //a map que realice esa accion
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_LIST_H
```

```cpp
//
// Created by agustin on 23/6/20.
//

#include "List.h"
#include "../../Entities/Player.h"

List::List(Player &_player, Coordinate _npcPosition) : player(_player) {
    npcPosition = _npcPosition;
}

void List::operator()(ServerProtocol& protocol) {
    player.listFrom(npcPosition);
}
```

```cpp
//
// Created by agustin on 13/7/20.
//

#ifndef ARGENTUM_GETINVENTORYNAMES_H
#define ARGENTUM_GETINVENTORYNAMES_H


#include "Event.h"

class Player;

//Clase que se almacena en la cola de eventos cuando un player quiere ejecutar
//el comando inventory
class GetInventoryNames: public Event {
private:
    Player& player;

public:
    explicit GetInventoryNames(Player& player);

    //Guarda en el minichat del player almacenado mensajes que indican el nombre
    //de cada item del inventario junto a la posicion del inventario en el que
    //este se encuentra. De la forma posicion: Nombre
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_GETINVENTORYNAMES_H
```

```cpp
//
// Created by agustin on 13/7/20.
//

#include "GetInventoryNames.h"

#include "../../Entities/Player.h"

GetInventoryNames::GetInventoryNames(Player &player): player(player) {

}

void GetInventoryNames::operator()(ServerProtocol &protocol) {
    player.getInventoryNames();
}
```

```cpp
//
// Created by agustin on 23/6/20.
//

#ifndef ARGENTUM_EVENT_H
#define ARGENTUM_EVENT_H

class ServerProtocol;

//Interfaz para encolar eventos para game
class Event {
public:

    //Funcion a implementar para cada clase que herede de esta, debe realizar
    //la accion del juegoq que se quiere ejecutar
    virtual void operator()(ServerProtocol& protocol) = 0;
    virtual ~Event() = default;
};


#endif //ARGENTUM_EVENT_H
```

jul 21, 20 15:47 **Event.cpp** Page 1/1

```
//
// Created by agustin on 23/6/20.
//

#include "Event.h"
```

jul 21, 20 15:47 **Drop.h** Page 1/1

```
//
// Created by agustin on 23/6/20.
//

#ifndef ARGENTUM_DROP_H
#define ARGENTUM_DROP_H


#include "Event.h"

#include <string>
#include <memory>
#include <list>
#include "../../Items/Item.h"
#include "../../Map/Coordinate.h"

class ServerProtocol;
class Player;
class Game;

//Clase que se almacena en la cola de eventos cuando un player quiere ejecutar
//el comando drop
class Drop: public Event {
private:
    Player* player{nullptr};
    int inventoryPosition{−1};

    std::list<std::shared_ptr<Item>> items;
    Game* game{nullptr};
    Coordinate dropPosition{−1, −1};

public:
    Drop(Player& player, int position);

    //La lista debe contener al menos 1 elemento si se utiliza este contrsuctor, sino
    //se tirara la excepcion std::invalid_argument
    Drop(Game& game, std::list<std::shared_ptr<Item>>∧ items, Coordinate dropPosition);
    Drop(Game& game, std::shared_ptr<Item>∧ item, Coordinate dropPosition);

    //Si se llamo al primer constructor entonces se intenta sacar el item de la posicion
    //del inventario guardada y, si hay un item en esta, se envia al cliente que item debe mostrar
    //el item en el piso, ademas de guardarlo en el tile en el que se encuentra parado el player
    //Si se llamo al segundo o tercer constructor, se agregan todos los items recibidos
    //en el tile en el que se desean dejar y se manda al cliente el item que debe mostrar
    //en ese tile
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_DROP_H
```

```cpp
//
// Created by agustin on 23/6/20.
//

#include "Drop.h"
#include "../../Entities/Player.h"
#include "../../Items/ItemData.h"
#include "../../Server/ServerProtocol.h"
#include "../Game.h"
#include "../../Items/Item.h"
#include <msgpack.hpp>

MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::ItemType)


Drop::Drop(Player &_player, int _inventoryPosition) {
    player = &_player;
    inventoryPosition = _inventoryPosition;
    dropPosition = player->getPosition();
}

Drop::Drop(Game& _game, std::list<std::shared_ptr<Item>>&& _items, Coordinate _d
ropPosition) {
    game = &_game;
    dropPosition = _dropPosition;
    if (_items.empty()) {
        throw std::invalid_argument("List without elements in Drop");
    }
    items = std::move(_items);
}

Drop::Drop(Game &_game, std::shared_ptr<Item> &&item, Coordinate _dropPosition)
{
    game = &_game;
    dropPosition = _dropPosition;
    items.push_back(std::move(item));
}

void Drop::operator()(ServerProtocol& protocol) {
    const Item* itemPtr = nullptr;
    if (player) {
        itemPtr = player->dropItem(inventoryPosition);
    } else {
        itemPtr = items.back().get();
        game->dropItems(std::move(items), dropPosition);
    }
    if (itemPtr) {
        std::stringstream data;
        itemPtr->loadDropItemData(data, dropPosition.iPosition, dropPosition.jPo
sition);
        protocol.addToGeneralData(data);
    }
}
```

```cpp
//
// Created by agustin on 23/6/20.
//

#ifndef ARGENTUM_DEPOSIT_H
#define ARGENTUM_DEPOSIT_H


#include "Event.h"
#include <string>
#include "../../Map/Coordinate.h"

class Player;

//Clase que se almacena en la cola de eventos cuando un player quiere ejecutar
//el comando de deposit en un tile
class Deposit : public Event {
private:
    Player& player;
    std::string itemName;
    Coordinate npcPosition{};

public:
    Deposit(Player& player, std::string&& _itemName, Coordinate _npcPosition);

    //Intenta depositar el item con el nombre en el entity que se encuentre
    //en la coordenada guardada, pidiendole a map que realice esa accion
    void operator()(ServerProtocol& protocol) override;

};


#endif //ARGENTUM_DEPOSIT_H
```

```cpp
//
// Created by agustin on 23/6/20.
//

#include "Deposit.h"
#include "../../Entities/Player.h"

Deposit::Deposit(Player &_player, std::string^ _itemName, Coordinate _npcPositi
on) : player(_player) {
    itemName = std::move(_itemName);
    npcPosition = _npcPosition;
}

void Deposit::operator()(ServerProtocol& protocol) {
    player.depositTo(itemName, npcPosition);
}
```

```cpp
//
// Created by agustin on 23/6/20.
//

#ifndef ARGENTUM_BUY_H
#define ARGENTUM_BUY_H


#include "Event.h"
#include <string>
#include "../../Map/Coordinate.h"

class Player;

//Clase que se almacena en la cola de eventos cuando un player quiere ejecutar
//el comando de comprar en un tile
class Buy : public Event {
private:
    Player& player;
    std::string itemName;
    Coordinate npcPosition{};

public:
    Buy(Player& player, std::string^ _itemName, Coordinate _npcPosition);

    //Intenta comprar el item con el nombre guardado al entity que se encuentre
    //en la coordenada guardada, pidiendole a map que realice esa accion
    void operator()(ServerProtocol& protocol) override;
};


#endif //ARGENTUM_BUY_H
```

jul 21, 20 15:47 **Buy.cpp** Page 1/1

```
//
// Created by agustin on 23/6/20.
//

#include "Buy.h"
#include "../../Entities/Player.h"

Buy::Buy(Player &_player, std::string^ _itemName, Coordinate _npcPosition) : pl
ayer(_player) {
    itemName = std::move(_itemName);
    npcPosition = _npcPosition;
}

void Buy::operator()(ServerProtocol& protocol) {
    player.buyFrom(itemName, npcPosition);
}
```

jul 21, 20 15:47 **Attack.h** Page 1/1

```
//
// Created by agustin on 23/6/20.
//

#ifndef ARGENTUM_ATTACK_H
#define ARGENTUM_ATTACK_H


#include "Event.h"
#include "../../Map/Coordinate.h"
#include "../../../libs/GameEnums.h"

class Entity;

//Clase que se almacena en la cola de eventos cuando se quiere que un entity
//ataque a un tile
class Attack : public Event {
private:
    Entity& entity;
    Coordinate target{};

public:
    Attack(Entity& _entity, Coordinate _target);

    //Realiza el ataque y de ser necesario almacena el evento en el protocolo
    //para comunicarselo a los clientes, pidiendole a map que realice esa accion
    void operator()(ServerProtocol& protocol) override;

private:
    GameType::Direction _attackDirection(Coordinate attackerPosition) const;
};


#endif //ARGENTUM_ATTACK_H
```

```
//
// Created by agustin on 23/6/20.
//

#include "Attack.h"
#include "../../Entities/Entity.h"
#include "../../Server/ServerProtocol.h"
#include "msgpack.hpp"

MSGPACK_ADD_ENUM(GameType::Direction)
MSGPACK_ADD_ENUM(GameType::EventID)

Attack::Attack(Entity& _entity, Coordinate _target) : entity(_entity) {
    target = _target;
}

void Attack::operator()(ServerProtocol& protocol) {
    int32_t usedWeapon = entity.attack(target);
    if (usedWeapon ≠ −1) {
        GameType::Direction attackDir = _attackDirection(entity.getPosition());
        std::stringstream data;
        msgpack::type::tuple<GameType::EventID> messageTypeData(GameType::ATTACK
);
        msgpack::pack(data, messageTypeData);
        msgpack::type::tuple<std::string, int32_t, int32_t, int32_t,
                                            GameType::Direction> attackCoordinateData
                                        (entity.getNickname(), target.iPosition, target.
jPosition,
                                                usedWeapon, attackDir);
        msgpack::pack(data, attackCoordinateData);
        protocol.addToGeneralData(data);
    }
}

GameType::Direction Attack::_attackDirection(Coordinate attackerPosition) const
{
    if (attackerPosition.iPosition ≡ target.iPosition) {
        if (attackerPosition.jPosition > target.jPosition) {
            return GameType::DIRECTION_LEFT;
        } else {
            return GameType::DIRECTION_RIGHT;
        }
    } else {
        if (attackerPosition.iPosition < target.iPosition) {
            return GameType::DIRECTION_DOWN;
        } else {
            return GameType::DIRECTION_UP;
        }
    }
}
```

```
//
// Created by marcos on 17/7/20.
//

#ifndef ARGENTUM_UNAVAILABLEPLAYEREXCEPTION_H
#define ARGENTUM_UNAVAILABLEPLAYEREXCEPTION_H

#include <exception>

class UnavailablePlayerException : std::exception {

};


#endif //ARGENTUM_UNAVAILABLEPLAYEREXCEPTION_H
```

```
//
// Created by marcos on 17/7/20.
//

#ifndef ARGENTUM_INEXISTENTPLAYEREXCEPTION_H
#define ARGENTUM_INEXISTENTPLAYEREXCEPTION_H

#include <exception>

class InexistentPlayerException : std::exception {

};


#endif //ARGENTUM_INEXISTENTPLAYEREXCEPTION_H
```

```
//
// Created by agustin on 14/6/20.
//

#ifndef ARGENTUM_PLAYERSTATS_H
#define ARGENTUM_PLAYERSTATS_H

#include <sstream>
#include <cstdint>
#include "../../libs/GameEnums.h"
#include "../Server/PlayerData.hpp"
#include "../Config/ConfigFileReader.h"
#include "../Config/Configuration.h"

class EntityTests;
class MapTests;
class Minichat;

class PlayerStats {
private:
    bool isMeditating{};
    double timeElapsedLife{};
    double timeElapsedMana{};

    int32_t constitution{};
    int32_t  intelligence{};
    int32_t  agility{};
    int32_t  strength{};

    int32_t  classLifeMultiplier{};
    int32_t  raceLifeMultiplier{};
    int32_t  classManaMultiplier{};
    int32_t  raceManaMultiplier{};
    int32_t  recoveryRate{};
    int32_t  meditationRate{};

    int32_t  experience{};
    int32_t  nextLevelExperience{};
    int32_t  level{};
    int32_t  currentMana{};
    int32_t  currentLife{};
    int32_t  maxMana{};
    int32_t  maxLife{};

    const double TIME_FOR_RECOVERY{Configuration::getInstance().configPlayerRecoveryTime()*1000};

    friend EntityTests;
    friend MapTests;

public:
    //Construye el PlayerStats utilizando los datos almacenados en la instancia de PlayerData
    //Esta pensado para cargar los stats que tuvo un jugador antes de desconectarse
    explicit PlayerStats(const PlayerData& data);

    //Retorna el danio base que logro hacer el arma del player para el ataque
    int getTotalDamage(int weaponDamage) const;

    //Retorna el level actual del player
    unsigned int getLevel() const;

    //Aumenta la xp del player, retorna true si subio de nivel, false en otro caso
    //Un player nunca puede subir de a mas de un nivel ya que la experiencia de
    //sobra es descartada
    bool increaseExperience(unsigned int _experience);

    //Modifica la vida del player acorde al danio/curacion ocasionados
    //Retorna el pair(danio total recibido, pudo esquivar)
    //Concatena en attackedMessage prefijos para el mensaje de ataque segun el
    //resultado
    std::pair<int, bool> modifyLife(int damage, unsigned int attackerLevel, unsi
gned int defense,
                    bool isAPlayer, std::string& attackedMessage);

    //Retorna la maxima vida que puede tener el player dados sus stats actuales
    int getMaxLife() const;

    //Retorna la vida actual del player
    int getCurrentLife() const;

    //Restaura hasta amount cantidad de vida, sin pasarse de la cantidad maxima
    void restoreLife(unsigned int amount);

    //Restaura hasta amount cantidad de mana, sin pasarse de la cantidad maxima
    void restoreMana(unsigned int amount);

    //Retorna si el player esta o no muerto
    bool isDead() const;

    //Actualiza el estado de la vida y el mana del player
    void update(double timeStep);

    //Setea el player a modo meditacion
    void startMeditating(Minichat& minichat);

    //Hace que el player deje de estar en modo meditacion
    void stopMeditating(Minichat& minichat);

    int32_t& getCurrentMana();

    //Setea el mana y la vida actual en sus valores maximos
    void restore();

    //Guarda experiencia, experiencia para proximo nivel, nivel, mana actual, ma
na maximo,
    //vida, vida maxima y si esta vivo (true) o muerto (false) en buffer
    void storeAllRelevantData(std::stringstream& buffer) const;

    //Guarda true si esta vivo, sino guarda false en buffer
    void storeLifeStatus(std::stringstream& buffer) const;

    //Almacena las stats del player en pData, se usa para el backup del archivo
    void getData(PlayerData& pData) const;

    //Intenta consumir amount cantidad de mana, retorna true si lo pudo hacer,
    //sino retorna false
    bool consumeMana(unsigned int amount);

private:
    void _increaseStats();
    void _loadInitialStats(Config::Modifiers& classM, Config::Modifiers& raceM,
                                        const PlayerData& data);
    void _loadGenericStats(Config::Modifiers& classM, Config::Modifiers& raceM,
                                        const PlayerData& data);
};


#endif //ARGENTUM_PLAYERSTATS_H
```

```
//
// Created by agustin on 14/6/20.
//

#include "PlayerStats.h"
#include "../Config/Calculator.h"
#include "Minichat.h"
#include <algorithm>
#include <msgpack.hpp>

#define MUCH_LEVEL_DIFF_MESSAGE "I think the level gap between us is a tad much, I’m level "
#define CRITICAL_MESSAGE "That must have hurt! Critical! "
#define DODGE_MESSAGE "Too weak, too slow. "
#define STARTED_MEDITATING_MESSAGE "Started meditating\n"
#define STOPPED_MEDITATING_MESSAGE "Stopped meditating\n"

using namespace GameType;

///////////////////////////////////PUBLIC////////////////////////////////////

//Funcion auxiliar para inicializar el resto de los atributos de la clase
void PlayerStats::_loadGenericStats(Config::Modifiers& classM, Config::Modifiers
& raceM,
                                    const PlayerData& data) {
    isMeditating = false;
    timeElapsedLife = 0;
    timeElapsedMana = 0;
    experience = data.experience;
    level = data.level;
    maxLife = Calculator::calculateMaxLife(constitution, classLifeMultiplier, ra
ceLifeMultiplier, level);
    maxMana = Calculator::calculateMaxMana(intelligence, classManaMultiplier, ra
ceManaMultiplier, level);
    currentLife = maxLife;
    currentMana = maxMana;
    nextLevelExperience = Calculator::calculateNextLevelXP(level);
    recoveryRate = raceM.recoveryRate;
    meditationRate = classM.meditationRate;
}

//Setea los valores de los stats del player segun sea un jugador nuevo o uno ya
existente
void PlayerStats::_loadInitialStats(Config::Modifiers& classM, Config::Modifiers
& raceM,
                                    const PlayerData& data) {
    if (data.isNewPlayer) {
        constitution += classM.constitution + raceM.constitution;
        intelligence += classM.intelligence + raceM.intelligence;
        agility += classM.agility + raceM.agility;
        strength += classM.strength + raceM.strength;
    } else {
        constitution = data.constitution;
        intelligence = data.intelligence;
        agility = data.agility;
        strength = data.strength;
    }
}

//Aumenta en 1 el nivel de los atributos guardados y recalcula los nuevos valore
s
//maximos de mana y vida, asignandole tambien estos a los valores actuales
void PlayerStats::_increaseStats() {
    ++strength;
    ++agility;
    ++intelligence;
    ++constitution;
```

```cpp
    maxLife = Calculator::calculateMaxLife(constitution, classLifeMultiplier, raceLifeMultiplier,
                                           level);
    maxMana = Calculator::calculateMaxMana(intelligence, classManaMultiplier, raceManaMultiplier,
                                           level);
    currentLife = maxLife;
    currentMana = maxMana;
}

/////////////////////////////////PUBLIC/////////////////////////////////

PlayerStats::PlayerStats(const PlayerData& data) {
    Configuration& config = Configuration::getInstance();
    Config::Modifiers classModifier = config.configClassModifiers(data.pClass);
    Config::Modifiers raceModifier = config.configRaceModifiers(data.pRace);
    classLifeMultiplier = classModifier.lifeMultiplier;
    classManaMultiplier = classModifier.manaMultiplier;
    raceLifeMultiplier = raceModifier.lifeMultiplier;
    raceManaMultiplier = raceModifier.manaMultiplier;
    _loadInitialStats(classModifier, raceModifier, data);
    _loadGenericStats(classModifier, raceModifier, data);
}

int PlayerStats::getTotalDamage(int weaponDamage) const {
    return Calculator::calculateDamage(strength, weaponDamage);
}

unsigned int PlayerStats::getLevel() const {
    return level;
}

bool PlayerStats::increaseExperience(unsigned int _experience) {
    experience += _experience;
    if (experience ≥ nextLevelExperience) {
        ++level;
        _increaseStats();
        experience = 0;
        nextLevelExperience = Calculator::calculateNextLevelXP(level);
        return true;
    }
    return false;
}

std::pair<int, bool> PlayerStats::modifyLife(int damage, unsigned int attackerLevel, unsigned int defense,
                              bool isAPlayer, std::string& attackedMessage) {
    if (damage < 0) {
        currentLife += −damage;
        if (currentLife > maxLife) currentLife = maxLife;
        currentMana = 0;
        return {damage, false};
    } else {
        Configuration& config = Configuration::getInstance();
        if (isAPlayer ∧ std::abs(static_cast<int32_t>(attackerLevel) − level) >
                                    static_cast<int32_t>(config.configMaxLevelDiff())) {
            attackedMessage += MUCH_LEVEL_DIFF_MESSAGE + std::to_string(level) + "\n";
            return {0, false};
        }
        if (Calculator::isCritical()) {
            attackedMessage += CRITICAL_MESSAGE;
            damage = damage * 2;
        } else if (Calculator::canDodge(agility)) {
            attackedMessage += DODGE_MESSAGE;
```

```cpp
            return {0, true};
        }
        int totalDamage = std::max(damage − static_cast<int>(defense), 0);
        currentLife −= totalDamage;
        if (currentLife ≤ 0) {
            currentLife = 0;
            currentMana = 0;
        }
        if (totalDamage > 0) {
            timeElapsedLife = 0.0;
        }
        return {totalDamage, false};
    }
}

int PlayerStats::getMaxLife() const {
    return maxLife;
}

int PlayerStats::getCurrentLife() const {
    return currentLife;
}

void PlayerStats::restoreLife(unsigned int amount) {
    currentLife += static_cast<int>(amount);
    if (currentLife > maxLife) {
        currentLife = maxLife;
    }
}

void PlayerStats::restoreMana(unsigned int amount) {
    currentMana += amount;
    if (currentMana > maxMana) {
        currentMana = maxMana;
    }
}

bool PlayerStats::isDead() const {
    return (getCurrentLife() ≡ 0);
}

void PlayerStats::update(double timeStep) {
    if (isDead()) {
        return;
    }
    timeElapsedLife += timeStep;
    timeElapsedMana += timeStep;
    if (timeElapsedLife ≥ TIME_FOR_RECOVERY) {
        currentLife += Calculator::lifeRecovered(recoveryRate, timeElapsedLife/1000);
        if (currentLife > maxLife) {
            currentLife = maxLife;
        }
        timeElapsedLife = 0.0;
    }
    if (timeElapsedMana ≥ TIME_FOR_RECOVERY) {
        if (isMeditating) {
            currentMana += Calculator::manaRecoveredWithMeditation(meditationRate,
                                                                  intelligence, timeElapsedMana/1000);
        } else {
            currentMana += Calculator::manaRecoveredNoMeditation(recoveryRate,
                                                                timeElapsedMana/1000);
        }
        if (currentMana ≥ maxMana) {
            currentMana = maxMana;
        }
        timeElapsedMana = 0.0;
    }
}

void PlayerStats::startMeditating(Minichat& minichat) {
    if (¬isMeditating) {
        isMeditating = true;
        minichat.addMessage(STARTED_MEDITATING_MESSAGE);
    }
}

void PlayerStats::stopMeditating(Minichat& minichat) {
    if (isMeditating) {
        isMeditating = false;
        minichat.addMessage(STOPPED_MEDITATING_MESSAGE);
    }
}

int32_t& PlayerStats::getCurrentMana() {
    return currentMana;
}

void PlayerStats::restore() {
    currentMana = maxMana;
    currentLife = maxLife;
}

void PlayerStats::storeAllRelevantData(std::stringstream& buffer) const {
    msgpack::type::tuple<int32_t, int32_t, int32_t> xpData(experience, nextLevel
Experience, level);
    msgpack::pack(buffer, xpData);
    msgpack::type::tuple<int32_t, int32_t> manaData(currentMana, maxMana);
    msgpack::pack(buffer, manaData);
    msgpack::type::tuple<int32_t, int32_t> lifeData(currentLife, maxLife);
    msgpack::pack(buffer, lifeData);
    msgpack::type::tuple<int32_t, int32_t, int32_t, int32_t> statsData(strength,
                            constitution, intelligence, agility);
    msgpack::pack(buffer, statsData);
}

void PlayerStats::storeLifeStatus(std::stringstream& buffer) const {
    msgpack::type::tuple<bool> isAlive(¬isDead());
    msgpack::pack(buffer, isAlive);
}

void PlayerStats::getData(PlayerData &pData) const {
    pData.level = level;
    pData.experience = experience;
    pData.constitution = constitution;
    pData.strength = strength;
    pData.agility = agility;
    pData.intelligence = intelligence;
}

bool PlayerStats::consumeMana(unsigned int amount) {
    if (currentMana < static_cast<int>(amount)) {
        return false;
    }
    if (amount ≠ 0) {
        currentMana −= amount;
        timeElapsedMana = 0;
    }
    return true;
```

```
}
```

```cpp
//
// Created by marcos on 23/6/20.
//

#ifndef ARGENTUM_PLAYERPROXY_H
#define ARGENTUM_PLAYERPROXY_H

#include <string>
#include <queue>
#include "../Items/Inventory.h"
#include "../Game/Events/Event.h"

class Player;
class Game;
struct Coordinate;

class PlayerProxy {
private:
    Game* game{nullptr};
    Player* player{nullptr};
    std::queue<std::unique_ptr<Event>> storedEvents;

public:
    PlayerProxy() = default;
    PlayerProxy(PlayerProxy^ other) noexcept;
    PlayerProxy& operator=(PlayerProxy^ other) noexcept;
    PlayerProxy(const PlayerProxy& other) = delete;
    PlayerProxy& operator=(const PlayerProxy& other) = delete;
    explicit PlayerProxy(Game* _game, Player* _player);

    /*Encola un ataque*/
    void attack(Coordinate target);

    /*Encola un use item (equipar para los equipables, consumir para los consumi
bles, etc)*/
    void useItem(int32_t itemPosition);

    /*Setea al player en modo meditacion (es el unico evento que no encolamos
     * porque no valia la pena*/
    void meditate();

    /*Encola un buy*/
    void buyFrom(std::string^ itemName, Coordinate npcPosition);

    /*Encola un sell*/
    void sellTo(std::string^ itemName, Coordinate npcPosition);

    /*Encola un withdraw*/
    void withdrawFrom(std::string^ itemName, Coordinate npcPosition);

    /*Encola un mensaje a otro player*/
    void messageOtherPlayer(std::string^ playerToMessage, std::string^ message)
;

    /*Encola un list*/
    void listFrom(Coordinate npcPosition);

    /*Encola un deposit*/
    void depositTo(std::string^ itemName, Coordinate npcPosition);

    /*Encola un unequip*/
    void unequip(GameType::EquipmentPlace clothing);

    /*Encola un drop item*/
    void dropItem(int32_t itemPosition);

```

```cpp
    /*Encola un pickup de un item*/
    void pickUpItem();

    /*Encola un resurrect*/
    void requesResurrect(Coordinate selectedPosition);

    /*Encola un heal*/
    void requestHeal(Coordinate selectedPosition);

    /*Encola un request para ver el nombre de los items del inventario*/
    void getInventoryNames();

    /*Guarda en data los datos del player administrado por este PlayerProxy*/
    void storeAllRelevantData(std::stringstream& data) const;

    /*Borra el minichat del player*/
    void clearMinichat();

    /*Mergea la cola del PlayerProxy con la del Game*/
    void giveEventsToGame();

    /*Encola el comando para empezar a moverse en una direccion*/
    void startMoving(GameType::Direction direction);

    /*Encola la directiva para dejar de moverse*/
    void stopMoving();

    /*Retorna los datos actuales del player*/
    PlayerData getData() const;
};


#endif //ARGENTUM_PLAYERPROXY_H
```

```cpp
//
// Created by marcos on 23/6/20.
//

#include "PlayerProxy.h"
#include "../Map/Coordinate.h"
#include "../Game/Game.h"
#include "../Game/Events/Attack.h"
#include "../Game/Events/Buy.h"
#include "../Game/Events/Sell.h"
#include "../Game/Events/Withdraw.h"
#include "../Game/Events/List.h"
#include "Player.h"
#include "../Game/Events/Deposit.h"
#include "../Game/Events/Drop.h"
#include "../Game/Events/Unequip.h"
#include "../Game/Events/UseItem.h"
#include "../Game/Events/PickUpItem.h"
#include "../Game/Events/RequestResurrect.h"
#include "../Game/Events/Message.h"
#include "../Game/Events/RestoreStats.h"
#include "../Game/Events/GetInventoryNames.h"
#include "../Game/Events/ModifyPlayerMovement.h"

const unsigned int MAX_EVENTS_STORED = 3;

PlayerProxy::PlayerProxy(PlayerProxy ∧other) noexcept {
    game = other.game;
    other.game = nullptr;
    player = other.player;
    other.player = nullptr;
    storedEvents = std::move(other.storedEvents);
}

PlayerProxy &PlayerProxy::operator=(PlayerProxy ∧other) noexcept {
    game = other.game;
    other.game = nullptr;
    player = other.player;
    other.player = nullptr;
    storedEvents = std::move(other.storedEvents);
    return *this;
}


PlayerProxy::PlayerProxy(Game *_game, Player *_player) {
    game = _game;
    player = _player;
}

void PlayerProxy::attack(Coordinate target) {
    if (player→getPosition() ≠ target) {
        if (storedEvents.size() < MAX_EVENTS_STORED) {
            storedEvents.emplace(new Attack(*player, target));
        }
    }
}

void PlayerProxy::useItem(int32_t itemPosition) {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new UseItem(*player, itemPosition));
    }
}

void PlayerProxy::meditate() {
    player→meditate();
}
```

```cpp

void PlayerProxy::buyFrom(std::string ∧itemName, Coordinate npcPosition) {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new Buy(*player, std::move(itemName), npcPosition))
;
    }
}

void PlayerProxy::sellTo(std::string ∧itemName, Coordinate npcPosition) {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new Sell(*player, std::move(itemName), npcPosition)
);
    }
}

void PlayerProxy::withdrawFrom(std::string ∧itemName, Coordinate npcPosition) {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new Withdraw(*player, std::move(itemName), npcPosit
ion));
    }
}

void PlayerProxy::listFrom(Coordinate npcPosition) {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new List(*player, npcPosition));
    }
}

void PlayerProxy::depositTo(std::string ∧itemName, Coordinate npcPosition) {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new Deposit(*player, std::move(itemName), npcPositi
on));
    }
}

void PlayerProxy::unequip(GameType::EquipmentPlace place) {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new Unequip(*player, place));
    }
}

void PlayerProxy::dropItem(int32_t itemPosition) {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new Drop(*player, itemPosition));
    }
}

void PlayerProxy::pickUpItem() {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new PickUpItem(*game, *player));
    }
}

void PlayerProxy::requesResurrect(Coordinate selectedPosition) {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new RequestResurrect(*game, *player, selectedPositi
on));
    }
}

void PlayerProxy::messageOtherPlayer(std::string ∧playerToMessage, std::string
∧message) {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new Message(*game, *player, std::move(playerToMessa
ge),
                                         std::move(message)));
    }
}

void PlayerProxy::requestHeal(Coordinate selectedPosition) {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new RestoreStats(*game, *player, selectedPosition))
;
    }
}

void PlayerProxy::getInventoryNames() {
    if (storedEvents.size() < MAX_EVENTS_STORED) {
        storedEvents.emplace(new GetInventoryNames(*player));
    }
}

/*Aca no lo limitamos al tamanio de la cola porque es un evento que no
 * deberiamos ignorar nunca*/
void PlayerProxy::startMoving(GameType::Direction direction) {
    storedEvents.emplace(new ModifyPlayerMovement(*player, direction));
}

/*Aca no lo limitamos al tamanio de la cola porque es un evento que no
 * deberiamos ignorar nunca*/
void PlayerProxy::stopMoving() {
    storedEvents.emplace(new ModifyPlayerMovement(*player));
}

void PlayerProxy::giveEventsToGame() {
    while (¬storedEvents.empty()) {
        game→pushEvent(std::move(storedEvents.front()));
        storedEvents.pop();
    }
}

void PlayerProxy::clearMinichat() {
    player→clearMinichat();
}

void PlayerProxy::storeAllRelevantData(std::stringstream& data) const {
    player→storeAllRelevantData(data);
}

PlayerData PlayerProxy::getData() const {
    PlayerData data = player→getData();
    Game::getPlayerBank(data);
    return data;
}


```

```cpp
//
// Created by agustin on 8/6/20.
//

#ifndef ARGENTUM_PLAYER_H
#define ARGENTUM_PLAYER_H


#include "../Items/Inventory.h"
#include "Entity.h"
#include "PlayerStats.h"
#include "Minichat.h"
#include "../Items/ItemData.h"
#include "MovementBackup.h"

class Game;
class EntityTests;
class MapTests;

class Player: public Entity {
private:
    GameType::Race race;
    GameType::Class pClass;
    Inventory inventory;
    PlayerStats stats;
    int32_t gold;
    Minichat chat;
    Game& game;
    MovementBackup movementBackup{};

    friend EntityTests;
    friend MapTests;

public:
    Player(Game& _game, Coordinate _initialPosition, const PlayerData& data);

    /*Indica si el jugador es target de un monster, un jugador es un target si e
sta vivo, si
    esta muerto no lo es*/
    bool isMonsterTarget() override;

    /*Ataca el lugar especificado en target*/
    int32_t attack(Coordinate target) override;

    /*Si hay lugar en el inventario del player entonces se apropia del item y
    retorna true, sino no se apropia de el y retorna false*/
    bool storeItem(std::shared_ptr<Item>& item);

    /*Retorna el item que almacene el inventario al pedirle el item con nombre
    itemName*/
    std::shared_ptr<Item> removeItem(const std::string& itemName);

    /*Intenta reducir en amount la cantidad de oro que guarda, si esta es menor
    que amount entonces no la reduce y retorna false, sino la reduce y retorna
    true*/
    bool spendGold(int amount);

    /*Incrementa el oro del Player en cantidad amount*/
    void receiveGold(unsigned int amount);

    /*Usa el item en la posicion indicada, si no hay un item en la posicion no
    hace nada*/
    UseReturnData useItem(int itemPosition);

    /*Ataca al player, retorna el danio ocasionado y el xp ganado*/
    AttackResult attacked(int damage, unsigned int attackerLevel, bool isAPlayer
) override;

    /*Restaura hasta amount cantidad de vida, sin pasarse de la cantidad maxima*
/
    void restoreLife(unsigned int amount);

    /*Restaura hasta amount cantidad de mana, sin pasarse de la cantidad maxima*
/
    void restoreMana(unsigned int amount);

    /*Actualiza al player acorde a su estado actual (moviendo, meditando, etc)*/
    void update(double timeStep);

    /*Hace que el player comience a meditar (el Warrior no puede meditar)*/
    void meditate();

    /*Desequipa la ropa de equipment place*/
    bool unequip(GameType::EquipmentPlace clothing);

    /*Compra el item de nombre itemName del npc en la posicion npcPosition, si n
o hay
    uno no hace nada*/
    void buyFrom(const std::string& itemName, Coordinate npcPosition);

    /*Vende el item de nombre itemName al npc en la posicion npcPosition, si no
hay
    uno no hace nada*/
    void sellTo(const std::string& itemName, Coordinate npcPosition);

    /*Pide recuperar un item de una posicion, en el caso que haya un banker
     * se retornara el item pedido (si existiera), si no hay un banker no pasa
     * nada*/
    void withdrawFrom(const std::string& itemName, Coordinate npcPosition);

    /*Pide que le listen los items a un NPC ubicado en npcPosition, si no hay
     * un npc valido no pasa nada*/
    void listFrom(Coordinate npcPosition);

    /*Intenta depositar un item en el banker ubicado en npcPosition, si no hay
    * un npc valido no pasa nada*/
    void depositTo(const std::string& itemName, Coordinate npcPosition);

    /*Desequipa el arma*/
    bool unequip();

    /*Retorna una instancia de un ItemData que guarda el tipo e id de item y la
    posicion en la que el item fue dejado, si no se encuentra un item en la posi
cion
    recibida entonces se almacena -1 en el id del item*/
    const Item* dropItem(unsigned int itemPosition);

    /*Carga los datos generales del player de acuerdo al protocolo, se utiliza
     * para la info que se le envia a un cliente recien conectado*/
    void operator>>(std::stringstream& buffer) const override;

    /*Almacena los datos personales del player acorde al protocolo, estos
     * datos solo son enviados al cliente que controla a ese player
     * (seria la UI externa al mapa)*/
    void storeAllRelevantData(std::stringstream& buffer) const;

    /*Limpia el minichat del player, se llama despues de haber enviado el update
     * al cliente que controla a ese player. Es para no mandar los mensajes repe
tidos*/
    void clearMinichat();

    /*Agrega un mensaje al minichat del player*/
```

```
    void addMessage(const std::string& message);

    /*Le confirma a entity el request de movimiento para comenzar la interpolaci
on*/
    void move(Coordinate newPosition) override;

    /*Setea el mana y la vida del player al maximo*/
    void restoreStats(bool isBeingRevived);

    /*Retorna true si el player esta muerto, false en caso contrario*/
    bool isDead();

    /*Resetea la interpolacion del player*/
    void resetMovement();

    /*Retorna true si el player tiene dicho item, false en caso contrario*/
    bool hasItem(const std::string& itemName);

    /*Agrega al minichat del player en nombre de los items que tiene en el
     * inventario para que el cliente sepa como se llaman*/
    void getInventoryNames();

    /*Setea la direccion de movimiento del player en direction*/
    void startMovement(GameType::Direction direction);

    /*Deja de mover al player*/
    void stopMovement();

    int32_t getLevel() const override;

    bool hasFullInventory() const;

    /*Retorna su data actual (las cosas guardadas, stats, etc)*/
    PlayerData getData() const;

private:
    void _dropItems();
    void _storeAttackedResultMessage(std::string& resultMessage, std::pair<int,
bool> attackResult,
                                     unsigned int experience);
    AttackResult _receiveDamage(int damage, unsigned int attackerLevel, bool isA
Player);
};


#endif //ARGENTUM_PLAYER_H
```

```cpp
//
// Created by agustin on 8/6/20.
//

#include "Player.h"
#include "../Config/Calculator.h"
#include "AttackResult.h"
#include "../Game/Game.h"
#include "../Items/Miscellaneous/Gold.h"
#include "../Game/Events/Drop.h"
#include "../Game/Events/NotifyDeath.h"
#include "../Game/Events/Move.h"
#include "../Game/Events/PlayerLeveledUp.h"
#include <msgpack.hpp>

#define ATTACKER_IS_NEWBIE_MESSAGE "I won’t lose my time on a low level newbie like you!\n"
#define PLAYER_IS_A_NEWBIE_MESSAGE "Surely you have better things to do than attack a low level newbie
 like me...\n"
#define PLAYER_IS_DEAD_MESSAGE "You can’t kill a ghost, you know?\n"
#define DODGED_ATTACK_MESSAGE "You dodged an attack\n"
#define GHOSTS_CANT_RESTORE_STATS_MESSAGE "Ghosts can’t restore stats\n"
#define WARRIOR_CANT_MEDITATE_MESSAGE "You are a Warrior and Warriors cannot meditate!\n"

using namespace GameType;

MSGPACK_ADD_ENUM(GameType::Race)
MSGPACK_ADD_ENUM(GameType::EventID)

//////////////////////////////////PUBLIC//////////////////////////////

Player::Player(Game& _game, Coordinate _initialPosition, const PlayerData& data)
:
               Entity(GameType::Entity::PLAYER, _initialPosition, data.nickname,
 true),
               inventory(data),
               stats(data),
               game(_game) {

    speed = Configuration::getInstance().configPlayerSpeed();
    pClass = data.pClass;
    race = data.pRace;
    gold = data.gold;
    movementBackup = {false, GameType::DIRECTION_STILL};
}

int32_t Player::attack(Coordinate target) {
    int32_t returnValue = −1;
    if (¬stats.isDead()) {
        stats.stopMeditating(chat);
        int weaponDamage;
        weaponDamage = inventory.getWeaponDamage(currentPosition, target, stats)
;
        int totalDamage = stats.getTotalDamage(weaponDamage);
        if (totalDamage ≠ 0) {
            std::pair<AttackResult, bool> result = game.attackPosition(totalDama
ge, stats.getLevel(),
                                                       true, target);
            if (stats.increaseExperience(result.first.experience)) {
                game.pushEvent(std::unique_ptr<Event>(new PlayerLeveledUp(getNic
kname(), stats.getLevel())));
            }
            chat.addMessage(std::move(result.first.resultMessage));
            if (result.second) {
                returnValue = inventory.getWeaponId();
            }
        }
```

```cpp
    }
    return returnValue;
}


AttackResult Player::attacked(int damage, unsigned int attackerLevel, bool isAPl
ayer) {
    stats.stopMeditating(chat);
    unsigned int newbieLevel = Configuration::getInstance().configNewbieLevel();
    if (¬stats.isDead()) {
        if (isAPlayer ∧ damage > 0) {
            if (stats.getLevel() ≤ newbieLevel) {
                return {0, 0, PLAYER_IS_A_NEWBIE_MESSAGE};
            } else if (attackerLevel ≤ newbieLevel) {
                return {0, 0, ATTACKER_IS_NEWBIE_MESSAGE};
            }
        }
        return _receiveDamage(damage, attackerLevel, isAPlayer);
    } else {
        return {0, 0, PLAYER_IS_DEAD_MESSAGE};
    }
}

bool Player::isMonsterTarget() {
    return ¬stats.isDead();
}

bool Player::spendGold(int amount) {
    stats.stopMeditating(chat);
    if ((¬stats.isDead()) ∧ (amount ≤ gold)) {
        gold −= amount;
        return true;
    }
    return false;
}

void Player::receiveGold(unsigned int amount) {
    unsigned int maxGold = Calculator::calculateMaxSafeGold(stats.getLevel());
    maxGold += maxGold / 2;
    if (¬stats.isDead() ∧ (gold + amount) ≤ maxGold) {
        gold += amount;
    }
}

bool Player::storeItem(std::shared_ptr<Item> &item) {
    if ((¬stats.isDead()) ∧ (item)) {
        stats.stopMeditating(chat);
        if (item→isGold()) {
            std::shared_ptr<Gold> aux = std::dynamic_pointer_cast<Gold>(item);
            receiveGold(aux→getAmount());
            return true;
        } else {
            return inventory.addItem(item);
        }
    }
    return false;
}

std::shared_ptr<Item> Player::removeItem(const std::string &itemName) {
    if (¬stats.isDead()) {
        stats.stopMeditating(chat);
        return inventory.removeItem(itemName);
    }
    return nullptr;
}

```

```
UseReturnData Player::useItem(int itemPosition) {
    if (¬stats.isDead()) {
        stats.stopMeditating(chat);
        return inventory.useItem(*this, itemPosition);
    }
    return {GameType::EQUIPMENT_PLACE_NONE, −1};
}

void Player::restoreLife(unsigned int amount) {
    if (¬stats.isDead()) {
        stats.restoreLife(amount);
    } else {
        chat.addMessage(GHOSTS_CANT_RESTORE_STATS_MESSAGE);
    }
}

void Player::restoreMana(unsigned int amount) {
    if (¬stats.isDead()) {
        stats.restoreMana(amount);
    } else {
        chat.addMessage(GHOSTS_CANT_RESTORE_STATS_MESSAGE);
    }
}

void Player::meditate() {
    if (¬stats.isDead()) {
        if (pClass ≠ GameType::WARRIOR) {
            stats.startMeditating(chat);
        } else {
            addMessage(WARRIOR_CANT_MEDITATE_MESSAGE);
        }
    }
}

void Player::update(double timeStep) {
    Entity::update(timeStep, game); /*actualiza movimiento*/
    if (¬movement.isMoving ∧ movementBackup.isFollowingRoad) {
        game.pushEvent(std::unique_ptr<Event>(new Move(game, *this,
                        movementBackup.direction)));
    }
    stats.update(timeStep); /*actualiza la vida y manda en base al tiempo/medita
cion*/
}

bool Player::unequip(EquipmentPlace clothing) {
    if (¬stats.isDead()) {
        stats.stopMeditating(chat);
        return inventory.unequip(clothing);
    }
    return false;
}

bool Player::unequip() {
    if (¬stats.isDead()) {
        stats.stopMeditating(chat);
        return inventory.unequip();
    }
    return false;
}

const Item* Player::dropItem(unsigned int itemPosition) {
    std::shared_ptr<Item> aux = inventory.removeItem(itemPosition);
    const Item* returnData = aux.get();
    if (aux) {
        game.dropItems(std::move(aux), currentPosition);
    }
```

```
    return returnData;
}

void Player::buyFrom(const std::string &itemName, Coordinate npcPosition) {
    game.buy(*this, itemName, npcPosition);
}

void Player::sellTo(const std::string &itemName, Coordinate npcPosition) {
    game.sell(*this, itemName, npcPosition);
}

void Player::withdrawFrom(const std::string &itemName, Coordinate npcPosition) {
    game.withdraw(*this, itemName, npcPosition);
}

void Player::listFrom(Coordinate npcPosition) {
    game.list(*this, npcPosition);
}

void Player::depositTo(const std::string &itemName, Coordinate npcPosition) {
    game.deposit(*this, itemName, npcPosition);
}

void Player::operator>>(std::stringstream &buffer) const {
    Entity::operator>>(buffer);
    msgpack::type::tuple<Race> data(race);
    msgpack::pack(buffer, data);
    stats.storeLifeStatus(buffer);
    inventory.storeEquippedItems(buffer);
}

void Player::storeAllRelevantData(std::stringstream& buffer) const {
    msgpack::type::tuple<int32_t, int32_t> data(gold, Calculator::calculateMaxSa
feGold(stats.getLevel()));
    msgpack::pack(buffer, data);
    inventory.storeAllData(buffer);
    stats.storeAllRelevantData(buffer);
    msgpack::type::tuple<int32_t, int32_t> position(currentPosition.iPosition,
                                                    currentPosition.jPosition);
    msgpack::pack(buffer, position);
    msgpack::type::tuple<std::string> minichat(chat.getMessages());
    msgpack::pack(buffer, minichat);
    msgpack::type::tuple<std::string> nick(Entity::getNickname());
    msgpack::pack(buffer, nick);
}

void Player::clearMinichat() {
    chat.clear();
}

void Player::addMessage(const std::string &message) {
    chat.addMessage(message);
}

void Player::move(Coordinate newPosition) {
    stats.stopMeditating(chat);
    Entity::move(newPosition);
}

void Player::restoreStats(bool isBeingRevived) {
    //if ((esta vivo y no pide que lo resuciten) || (pide que lo resuciten y est
a muerto)))
    //Esta funcion se usa para curar y para cuando se resucite, se chequea para
que no se cure
    //cuando llame a resucitar si es que esta vivo y para que no resucite si est
a vivo
```

```
    if ((¬stats.isDead() ∧ ¬isBeingRevived) ∨ (isBeingRevived ∧ stats.isDead())
) {
        stats.restore();
    }
    if (stats.isDead() ∧ ¬isBeingRevived) {
        chat.addMessage(GHOSTS_CANT_RESTORE_STATS_MESSAGE);
    }
}

bool Player::isDead() {
    return stats.isDead();
}

void Player::resetMovement() {
    movement.movedDistance = 0;
    movement.direction = DIRECTION_STILL;
    movement.isMoving = false;
}

bool Player::hasItem(const std::string& itemName) {
    return inventory.hasItem(itemName);
}

void Player::getInventoryNames() {
    inventory.getInventoryNames(chat);
}

void Player::startMovement(GameType::Direction direction) {
    movementBackup.isFollowingRoad = true;
    movementBackup.direction = direction;
}

void Player::stopMovement() {
    movementBackup.isFollowingRoad = false;
}

PlayerData Player::getData() const {
    PlayerData pData;
    pData.nickname = getNickname();
    pData.pRace = race;
    pData.pClass = pClass;
    pData.gold = gold;
    stats.getData(pData);
    inventory.getData(pData);
    return pData;
}

////////////////////////////////////PRIVATE///////////////////////////////

void Player::_dropItems() {
    std::list<std::shared_ptr<Item>> items = inventory.dropAllItems();
    int goldDropped = static_cast<int>(gold −
                                       Calculator::calculateMaxSafeGold(stats.ge
tLevel()));
    goldDropped = std::max(goldDropped, 0);
    gold −= goldDropped;
    if (goldDropped > 0) {
        items.emplace_back(std::make_shared<Gold>(goldDropped));
    }
    if (¬items.empty()) {
        //game.dropItems(std::move(items), currentPosition);
        game.pushEvent(std::unique_ptr<Event>(new Drop(game, std::move(items), c
urrentPosition)));
    }
}

```

```
void Player::_storeAttackedResultMessage(std::string& resultMessage, std::pair<i
nt, bool> attackResult,
                                  unsigned int experience) {
    std::string damageString = std::to_string(std::abs(attackResult.first));
    if (attackResult.second) {
        resultMessage += getNickname() + " dodged your attack\n";
    } else if (attackResult.first ≥ 0) {
        resultMessage += "You damaged " + getNickname() + " by " + damageString;
        resultMessage += " (Remaining Life: " + std::to_string(stats.getCurrentLife()
) +
                           ", XP Gained: " + std::to_string(experience) + ")\n";
    } else {
        resultMessage += "You healed " + getNickname() + " by " + damageString;
        resultMessage += " (Remaining Life: " + std::to_string(stats.getCurrentLife()
) + ")\n";
    }

    //Se agrega el mensaje del ataque al minichat del atacado
    if (attackResult.second) {
        chat.addMessage(DODGED_ATTACK_MESSAGE);
    } else if (attackResult.first ≥ 0) {
        chat.addMessage("You lost " + damageString + " health points\n");
    } else if (attackResult.first < 0) {
        chat.addMessage("You healed " + damageString + " health points\n");
    }
}

AttackResult Player::_receiveDamage(int damage, unsigned int attackerLevel, bool
 isAPlayer) {
    std::string attackedMessage;
    unsigned int defense = inventory.getDefense();
    std::pair<int, bool> result = stats.modifyLife(damage, attackerLevel, defens
e, isAPlayer, attackedMessage);
    unsigned int experience = Calculator::calculateAttackXP(result.first,
                                                            attackerLevel, stats
.getLevel());
    if (stats.isDead() ∧ result.first > 0) {
        _dropItems();
        experience += Calculator::calculateKillXP(attackerLevel,
                                                  stats.getLevel(), stats.getMax
Life());
        game.pushEvent(std::unique_ptr<Event>(new NotifyDeath(*this)));
    }
    _storeAttackedResultMessage(attackedMessage, result, experience);
    return {result.first, experience, std::move(attackedMessage)};
}

int32_t Player::getLevel() const {
    return stats.getLevel();
}

bool Player::hasFullInventory() const {
    return inventory.isFull();
}
```

```cpp
//
// Created by agustin on 14/7/20.
//

#ifndef ARGENTUM_MOVEMENTBACKUP_H
#define ARGENTUM_MOVEMENTBACKUP_H

#include "../../libs/GameEnums.h"

struct MovementBackup {
    bool isFollowingRoad;
    GameType::Direction direction;
};

#endif //ARGENTUM_MOVEMENTBACKUP_H
```

```cpp
//
// Created by marcos on 20/6/20.
//

#ifndef ARGENTUM_MONSTERSTATS_H
#define ARGENTUM_MONSTERSTATS_H

#include "../../libs/GameEnums.h"
#include <string>

class EntityTests;
struct AttackResult;

class MonsterStats {
private:
    unsigned int constitution;
    unsigned int agility;
    unsigned int strength;
    int damage;
    unsigned int speed;

    unsigned int level;
    int currentLife;
    int maxLife;
    unsigned int rangeOfVision;

    friend EntityTests;

public:
    explicit MonsterStats(GameType::Entity type);

    /*Retorna el rango de vision del monster*/
    unsigned int getRangeOfVision() const;

    /*Retorna el danio del monster*/
    int getDamage() const;

    /*Retorna el nivel del monster*/
    unsigned int getLevel() const;

    /*Recibe el danio y modifica la vida del monster acorde a este, retorna lueg
o
     * un pair con el danio neto recibido y un bool en true en caso de que haya
     * logrado esquivar el ataque (esto es asi ya que queda extendible por si el
     * monstruo pudiera tener defensa y el danio fuera 0 sin haber esquivado)
     * Almacena en attackedMessage prefijos para el mensaje a mostrar segun el
     * resultado del ataque*/
    std::pair<int, bool> modifyLife(int _damage, std::string& attackedMessage);

    /*Retorna la vida actual del monstruo*/
    int getCurrentLife() const;

    /*Retorna la vida maxima del monstruo*/
    int getMaxLife() const;

    /*Retorna la agilidad del monstruo*/
    unsigned int getAgility() const;
};


#endif //ARGENTUM_MONSTERSTATS_H
```

```cpp
//
// Created by marcos on 20/6/20.
//

#include "MonsterStats.h"
#include "../Config/Configuration.h"
#include "../Config/Calculator.h"

#define CRITICAL_ATTACK_MESSAGE "Critical attack. "
#define DODGED_ATTACK_MESSAGE "Fear runs through your spine. "

MonsterStats::MonsterStats(GameType::Entity type) {
    Configuration& config = Configuration::getInstance();
    Config::MonsterStats stats = config.configMonsterStats(type);
    level = Calculator::getRandomInt(static_cast<int>(stats.minLevel),
                                     static_cast<int>(stats.maxLevel));
    constitution = stats.constitution + level;
    strength = stats.strength + level;
    agility = stats.agility + level;
    damage = static_cast<int>(stats.damage) + 10 * strength;
    rangeOfVision = stats.rangeOfVision;
    maxLife = stats.life + constitution * 10;
    currentLife = maxLife;
    speed = stats.speed;
}

unsigned int MonsterStats::getRangeOfVision() const {
    return rangeOfVision;
}

int MonsterStats::getDamage() const {
    return damage;
}

unsigned int MonsterStats::getLevel() const {
    return level;
}

std::pair<int, bool> MonsterStats::modifyLife(int _damage, std::string& attacked
Message) {
    if (Calculator::isCritical()) {
        _damage *= 2;
        attackedMessage += CRITICAL_ATTACK_MESSAGE;
    } else if (Calculator::canDodge(getAgility())) {
        attackedMessage += DODGED_ATTACK_MESSAGE;
        return {0, true};
    }
    currentLife -= _damage;
    if (currentLife < 0) {
        currentLife = 0;
    }
    return {_damage, false};
}

int MonsterStats::getCurrentLife() const {
    return currentLife;
}

int MonsterStats::getMaxLife() const {
    return maxLife;
}

unsigned int MonsterStats::getAgility() const {
    return agility;
}
```

```cpp
//
// Created by agustin on 7/6/20.
//

#ifndef ARGENTUM_MONSTER_H
#define ARGENTUM_MONSTER_H


#include "Entity.h"
#include <memory>
#include "MonsterStats.h"
#include "../../libs/GameEnums.h"

class Game;
class Map;
class Item;
class EntityTests;

class Monster: public Entity {
private:
    const unsigned int timeBetweenActions;
    double elapsedTime;
    int inactiveCycles;
    MonsterStats stats;
    GameType::Weapon monsterWeapon;

    /*Guarda parte del camino para no llamar al pathfinding cada vez que se
    quiera mover*/
    std::list<Coordinate> pathCache;

    /*Guarda el mapa para ver sus alrededores, el juego lo guarda para poder
    modificarlo*/
    const Map& map;
    Game& game;

    friend EntityTests;

private:
    static unsigned int _getDistance(Coordinate a, Coordinate b);

    void _storeNearestPlayerPathCache();

    bool _tryToAttack();

    void _move();

public:
    Monster(Game& _game, Coordinate initialPositionunsigned,
            GameType::Entity _type, GameType::Weapon _weapon);


    /*DaÃ±a el monstruo, retorna la cantidad de danio recibido
    Recibe level por un tema de herencia y logica del juego, esto igualmente
    hace que el ataque sea extendible, podria aplicarse la logica del fair
    play tambien a los monstruos. El isAPlayer no lo usa pero hacia falta
    para el polimorfismo de Player*/
    AttackResult attacked(int damage, unsigned int attackerLevel, bool isAPlayer
) override;

    /*Actualiza al monstruo acorde a su estado (por ejemplo si esta moviendose
     * actualiza la interpolacion, o si tiene a un playera la vista lo
     * ataca*/
    void update(double timeStep);

    /*Indica si el monstruo esta muerto, retorna true si lo esta, sino retorna f
alse*/
```

```
    bool isDead() const;

    /*Ataca una posicion (lo hace cuando tiene un player a la vista*/
    int32_t attack(Coordinate attackedPosition) override;

    int32_t getLevel() const override;
};


#endif //ARGENTUM_MONSTER_H
```

```
//
// Created by agustin on 7/6/20.
//

#include "Monster.h"
#include "../Items/ItemsFactory.h"
#include "../Game/Game.h"
#include "AttackResult.h"
#include "../Config/Configuration.h"
#include "../Game/Events/Attack.h"
#include "../Game/Events/Move.h"
#include "../Game/Events/Drop.h"
#include "../Config/Calculator.h"

#define MAX_NUMBER_OF_CACHED_NODES 4

///////////////////////PRIVATE///////////////////////

//Retorna la distancia (siempre positiva) entre las dos coordenadas
unsigned int Monster::_getDistance(Coordinate a, Coordinate b) {
    return std::abs(a.iPosition − b.iPosition) + std::abs(a.jPosition − b.jPosit
ion);
}

/*Guarda parte del camino al jugador al cual tiene que moverse la menor cantidad
de veces para alcanzarlo*/
void Monster::_storeNearestPlayerPathCache() {
    unsigned int nearestTargetIndex = 0;
    std::vector<Coordinate> positions;
    map.getMoveTargets(currentPosition, stats.getRangeOfVision(), positions);
    if (¬positions.empty()) {
        std::vector<std::list<Coordinate>> allPaths/*(positions.size())*/;
        std::list<Coordinate> aux;
        for (auto & position : positions) {
            if (map.getPath(currentPosition, position, aux)) {
                allPaths.push_back(std::move(aux));
                if (allPaths[allPaths.size() − 1].size() < allPaths[nearestTarge
tIndex].size()) {
                    nearestTargetIndex = allPaths.size() − 1;
                }
                aux.clear();
            }
        }
        if (¬allPaths.empty()) {
            pathCache = std::move(allPaths[nearestTargetIndex]);
            if (pathCache.size() > MAX_NUMBER_OF_CACHED_NODES) {
                pathCache.resize(MAX_NUMBER_OF_CACHED_NODES);
            }
        }
    }
}

/*Intenta atacar en sus alrededores, si no encuentra un jugador a quien atacar
no hace nada y retorna false, sino vacia pathCache, ataca y retorna true*/
bool Monster::_tryToAttack() {
    std::vector<Coordinate> targets;
    map.getAttackTargets(currentPosition, stats.getRangeOfVision(), targets);
    for (auto & target : targets) {
        if (_getDistance(currentPosition, target) ≡ 1) {
            std::unique_ptr<Attack> attackFunction(new Attack(*this, target));
            game.pushEvent(std::move(attackFunction));
            pathCache.clear();
            inactiveCycles = 0;
            return true;
        }
    }
    return false;
}

/*Pide al game que lo mueva a la siguiente posicion en pathCache, si pathCache
esta vacio entonces busca el jugador mas cercano en su rango de vision y le
pide al mapa un camino a este
Si la proxima posicion a la que se va a mover esta ocupada entonces vuelve a
calcular el camino al jugador mas cercano (esto puede pasar si un monstruo se
pone en su camino)*/
void Monster::_move() {
    if (¬map.isPlaceAvailable(pathCache.front())) {
        pathCache.clear();
    }
    if (pathCache.empty()) {
        _storeNearestPlayerPathCache();
    }
    if (¬pathCache.empty()) {
        game.pushEvent(std::unique_ptr<Move>(new Move(game, *this,
                _getMoveDirection(pathCache.front()))));
        pathCache.pop_front();
        inactiveCycles = 0;
    } else if (inactiveCycles ≥ 10) {
        Coordinate newPosition = map.getMonsterRandomPosition(currentPosition);
        Coordinate noPositions = {−1, −1};
        if (newPosition ≠ noPositions) {
            game.pushEvent(std::unique_ptr<Move>(new Move(game, *this,
                    _getMoveDirection(newPosition))));
        }
        inactiveCycles = 0;
    }
}


///////////////////////PUBLIC////////////////////////

Monster::Monster(Game &_game, Coordinate initialPosition,
                 GameType::Entity _type, GameType::Weapon _weapon):
                 Entity(_type, initialPosition, "Monster"),
                 timeBetweenActions(Configuration::getInstance().configMonsterSt
ats(_type).reactionSpeed * 200),
                 stats(_type), map(_game.getMap()), game(_game) {
    monsterWeapon = _weapon;
    elapsedTime = 0;
    inactiveCycles = 0;
    type = _type;
    speed = Configuration::getInstance().configMonsterStats(_type).speed;
}

AttackResult Monster::attacked(int _damage, unsigned int attackerLevel, bool isA
Player) {
    AttackResult result{0, 0, ""};
    if (_damage ≤ 0) return result;
    if (¬isDead()) {
        std::pair<int, bool> realAttackResult = stats.modifyLife(_damage, result
.resultMessage);
        if (realAttackResult.second) {
            result.resultMessage += "The monster dodged your attack\n";
        } else {
            unsigned int experience = Calculator::calculateKillXP(attackerLevel,
                        stats.getLevel(), stats.getMaxLife());
            result.damage = realAttackResult.first;
            result.experience = experience;
            result.resultMessage += "You damaged the Monster by " +
                                    std::to_string(result.damage) + " (Remaining Lif
e: " +
                                    std::to_string(stats.getCurrentLife()) +
```

```
                                    ", XP Gained: " + std::to_string(result.experie
nce) + ")\n";
        }
        if (isDead()) {
            std::shared_ptr<Item> drop;
            ItemsFactory::getInstance().storeRandomDrop(drop, stats.getMaxLife()
);
            if (drop) {
                game.pushEvent(std::unique_ptr<Event>(new Drop(game,
                        std::move(drop), currentPosition)));
            }
        }
    }
    return result;
}

void Monster::update(double timeStep) {
    Entity::update(timeStep, game);
    elapsedTime += timeStep;
    if (elapsedTime ≥ timeBetweenActions) {
        elapsedTime = 0;
        ++inactiveCycles;
        if (¬_tryToAttack() ∧ ¬isMoving()) {
            _move();
        }
    }
}

bool Monster::isDead() const {
    return (stats.getCurrentLife() ≡ 0);
}

int32_t Monster::attack(Coordinate attackedPosition) {
    game.attackPosition(stats.getDamage(), stats.getLevel(), false, attackedPosi
tion);
    return monsterWeapon;
}

int32_t Monster::getLevel() const {
    return stats.getLevel();
}
```

```
//
// Created by marcos on 7/3/20.
//

#ifndef ARGENTUM_MINICHAT_H
#define ARGENTUM_MINICHAT_H

#include <string>

class Minichat {
private:
    std::string message;

public:
    /*Agrega un mensaje al minichat del player*/
    void addMessage(std::string^ msg);
    void addMessage(const std::string& msg);

    /*Retorna el string del minichat (el minichat en si digamos)*/
    std::string getMessages() const;

    /*Limpia el minichat (elimina el string)*/
    void clear();
};


#endif //ARGENTUM_MINICHAT_H
```

```
//
// Created by marcos on 7/3/20.
//

#include "Minichat.h"

void Minichat::addMessage(const std::string &msg) {
    message += msg;
}

void Minichat::addMessage(std::string^ msg) {
    message += msg;
}

std::string Minichat::getMessages() const {
    return message;
}

void Minichat::clear() {
    message.clear();
}
```

```cpp
//
// Created by agustin on 6/6/20.
//

#ifndef ARGENTUM_ENTITY_H
#define ARGENTUM_ENTITY_H


#include "../Map/Coordinate.h"
#include <list>
#include <chrono>
#include "../../libs/GameEnums.h"

struct ProductData;
struct AttackResult;
class Game;

struct Movement {
    bool isMoving{false};
    unsigned int movedDistance{0};

    /*Esta direccion solo tiene sentido si se setea que se esta moviendo el entity*/
    GameType::Direction direction{GameType::DIRECTION_STILL};
};

class Player;

class Entity {
private:
    static unsigned int availableId;
    std::string nickname;

protected:
    Coordinate currentPosition{};
    GameType::Entity type;
    Movement movement{};
    unsigned int speed{};

private:
    Coordinate _calculatePreviousPosition() const;

protected:
    GameType::Direction _getMoveDirection(Coordinate destination);

public:
    Entity(GameType::Entity _type, Coordinate initialPosition, const std::string& _nicknamePrefix,
           bool isPrefixUnique = false);

    /*Implementa el comportamiento realizado al atacar,
    debe ser modificado en las clases hijas de ser necesario*/
    virtual int32_t attack(Coordinate target);

    /*Implementa el comportamiento realizado al ser atacado, por default
    retorna struct nulo, debe ser modificado en las clases hijas de ser necesario*/
    virtual AttackResult attacked(int damage, unsigned int level, bool isAPlayer);

    /*Implementa el comportamiento realizado al pedirle una lista de los items
    que tiene en venta, por default no hace nada, debe ser reimplementada
    si la clase hija tiene objetos en venta para listar*/
    virtual void list(Player &player);

    /*Implementa el comportamiento realizado al pedirle uno de los items
```

```cpp
    que tiene guardados, por default no hace nada, debe ser reimplementada
    si la clase hija puede almacenar y devolver items*/
    virtual void withdraw(Player& player, const std::string& itemName);

    /*Implementa el comportamiento realizado al pedirle que guarde el item que se
    encuentra en la posicion pasada, por default no hace nada, debe ser reimplementada
    si la clase hija puede almacenar y devolver items*/
    virtual void deposit(Player& player, const std::string& itemName);

    /*Implementa el comportamiento realizado al intentar comprar un item
    con el nombre pasado, por default no hace nada, debe ser reimplementada
    si la clase hija puede comprar y vender items*/
    virtual void buy(Player& player, const std::string& itemName);

    /*Implementa el comportamiento realizado al intentar vender un item que se
    encuentra en la posicion pasada, por default no hace nada, debe ser reimplementada
    si la clase hija puede comprar y vender items*/
    virtual void sell(Player& player, const std::string& itemName);

    /*Indica si el Entity va a ser atacado por un monstruo, por default retorna
    false, las clases que hereden deben reimplementarla si son consideradas
    targets*/
    virtual bool isMonsterTarget();

    /*Le asigna al jugador la posicion recibida*/
    void setPosition(Coordinate coordinate);

    /*Retorna la posicion en la que quiere estar el jugadoras*/
    Coordinate getPosition() const;

    /*Le confirma a entity el request de movimiento para comenzar la interpolacion*/
    virtual void move(Coordinate newPosition);

    /*Por default no hace nada, las clases hijas deben reimplementarlo de ser
     * necesario (por ejemplo el Priest)*/
    virtual void requestHeal(Player& player);

    /*Actualiza el estado de la entity, por ejemplo si se esta moviendo le
    actualiza la interpolacion*/
    void update(double timeStep, Game& game);

    /*Retorna si el entity esta o no en movimiento (lo uso en monster)*/
    bool isMoving() const;

    /*Retorna el tipo de entity (tipo de monstruo, npc o si es un player)*/
    GameType::Entity getType() const;

    /*Carga los datos generales del entity de acuerdo al protocolo, se utiliza
     * para la info que se le envia a un cliente recien conectado*/
    virtual void operator>>(std::stringstream& buffer) const;

    virtual ~Entity() = default;

    /*Retorna el nickname del entity (los monsters/npc tiene asignado uno tambien*/
    virtual const std::string& getNickname() const;

    /*Actualiza la interpolacion de la entity cuando se esta moviendo
     * de un tile al otro*/
    int32_t executeDisplacement(int32_t displacement, bool& hasFinished);

    /*Retorna la coordenada del tile al que se esta desplazando la entity*/
    virtual Coordinate getFinalCoordinate(GameType::Direction direction);

    virtual int32_t getLevel() const;
};


#endif //ARGENTUM_ENTITY_H
```

```
//
// Created by agustin on 8/6/20.
//

#include "Entity.h"
#include "AttackResult.h"
#include "../Game/Game.h"
#include "../Game/Events/Moved.h"
#include <msgpack.hpp>
#include "../../libs/SharedConstants.h"

MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::Entity)
MSGPACK_ADD_ENUM(GameType::Direction)

unsigned int Entity::availableId = 0;

Entity::Entity(GameType::Entity _type, Coordinate initialPosition,
        const std::string& _nicknamePrefix, bool isPrefixUnique /*= false*/) {
    currentPosition.iPosition = initialPosition.iPosition;
    currentPosition.jPosition = initialPosition.jPosition;
    movement.movedDistance = 0;
    movement.isMoving = false;
    movement.direction = GameType::DIRECTION_STILL;
    type = _type;
    nickname = std::move(_nicknamePrefix);
    if (¬isPrefixUnique) {
        nickname += std::to_string(availableId);
        availableId++;
    }
}

void Entity::setPosition(Coordinate coordinate) {
    currentPosition = coordinate;
}

Coordinate Entity::getPosition() const {
    return currentPosition;
}

bool Entity::isMonsterTarget() {
    return false;
}

AttackResult Entity::attacked(int damage, unsigned int level, bool isAPlayer) {
    return {0, 0};
}

void Entity::list(Player &player) {
    //DO NOTHING
}

void Entity::withdraw(Player &player, const std::string& itemName) {
    //DO NOTHING
}

void Entity::deposit(Player &player, const std::string& itemName) {
    //DO NOTHING
}

void Entity::buy(Player &player, const std::string &itemName) {
    //DO NOTHING
}

void Entity::sell(Player &player, const std::string& itemName) {
    //DO NOTHING
```

```
}


void Entity::requestHeal(Player &player) {
    //DO NOTHING
}

void Entity::move(Coordinate newPosition) {
    movement.direction = _getMoveDirection(newPosition);
    currentPosition = newPosition;
    movement.isMoving = true;
}

void Entity::update(double timeStep, Game& game) {
    if (movement.isMoving) {
        std::unique_ptr<Moved> event(new Moved(*this, movement.direction,
                                        static_cast<unsigned int>(timeStep) * spe
ed));
        game.pushEvent(std::move(event));
    }
}

bool Entity::isMoving() const {
    return movement.isMoving;
}

int32_t Entity::attack(Coordinate target) {
    return −1;
}

GameType::Entity Entity::getType() const {
    return type;
}

void Entity::operator>>(std::stringstream& buffer) const {
    msgpack::type::tuple<GameType::EventID> idType(GameType::EventID::CREATE_ENT
ITY);
    msgpack::pack(buffer, idType);
    msgpack::type::tuple<GameType::Entity, std::string, int32_t>
                                        idData(type, nickname, getLevel());
    msgpack::pack(buffer, idData);

    Coordinate previousPosition = _calculatePreviousPosition();
    msgpack::type::tuple<int32_t, int32_t, GameType::Direction, int32_t> current
MovementData(previousPosition.iPosition,
            previousPosition.jPosition, movement.direction, movement.movedDistan
ce);

    msgpack::pack(buffer, currentMovementData);
}

const std::string &Entity::getNickname() const {
    return nickname;
}

int32_t Entity::executeDisplacement(int32_t displacement, bool& hasFinished) {
    int32_t realDisplacement = displacement;
    movement.movedDistance += displacement;
    hasFinished = false;
    if (movement.movedDistance ≥ TILE_DISTANCE_IN_METERS) {
        realDisplacement = displacement − (movement.movedDistance − TILE_DISTANC
E_IN_METERS);
        movement.movedDistance = 0;
        movement.direction = GameType::DIRECTION_STILL;
        movement.isMoving = false;
        hasFinished = true;
```

```
    }
    return realDisplacement;
}

Coordinate Entity::getFinalCoordinate(GameType::Direction moveDirection) {
    if (¬isMoving()) {
        switch (moveDirection) {
            case GameType::DIRECTION_UP:
                return {currentPosition.iPosition − 1, currentPosition.jPosition
};
            case GameType::DIRECTION_DOWN:
                return {currentPosition.iPosition + 1, currentPosition.jPosition
};
            case GameType::DIRECTION_RIGHT:
                return {currentPosition.iPosition, currentPosition.jPosition + 1
};
            case GameType::DIRECTION_LEFT:
                return {currentPosition.iPosition, currentPosition.jPosition − 1
};
            case GameType::DIRECTION_STILL:
                //do nothing
                break;
        }
    }
    return {−1, −1};
}


int32_t Entity::getLevel() const {
    return −1;
}

///////////////////////////////////PRIVATE//////////////////////////////////
//Calcula la posicion previa tomando en cuenta la direccion de movimiento y la
//posicion actual
Coordinate Entity::_calculatePreviousPosition() const {
    Coordinate previous = currentPosition;
    switch (movement.direction) {
        case GameType::DIRECTION_UP:
            previous.iPosition++;
            break;
        case GameType::DIRECTION_DOWN:
            previous.iPosition−−;
            break;
        case GameType::DIRECTION_LEFT:
            previous.jPosition++;
            break;
        case GameType::DIRECTION_RIGHT:
            previous.jPosition−−;
            break;
        case GameType::DIRECTION_STILL:
            //DO NOTHING
            break;
    }
    return previous;
}

///////////////////////////////////PROTECTED////////////////////////////////
GameType::Direction Entity::_getMoveDirection(Coordinate destination) {
    Coordinate difference = {destination.iPosition − currentPosition.iPosition,
                             destination.jPosition − currentPosition.jPosition};
    if (difference.iPosition  ≡ 1) {
        return GameType::DIRECTION_DOWN;
    } else if (difference.iPosition ≡ −1) {
        return GameType::DIRECTION_UP;
    } else if (difference.jPosition ≡ −1) {
```

```cpp
        return GameType::DIRECTION_LEFT;
    } else {
        return GameType::DIRECTION_RIGHT;
    }
}

```

```cpp
//
// Created by agustin on 15/6/20.
//

#ifndef ARGENTUM_TRADER_H
#define ARGENTUM_TRADER_H


#include "../Entity.h"
#include "Shop.h"

class EntityTests;

class Trader: public Entity {
private:
    Shop shop;

    friend EntityTests;

public:
    explicit Trader(Coordinate initialPosition);

    /*Retorna por le minichat del player los items que tiene el Priest a la vent
a junto con su precio*/
    void list(Player &player) override;

    /*Le vende al player el item pedido en caso de tenerlo, caso contrario no su
cede nada*/
    void buy(Player& player, const std::string& itemName) override;

    /*Le compra al player el item pedido en caso de tenerlo, caso contrario no s
ucede nada*/
    void sell(Player& player, const std::string& itemName) override;
};


#endif //ARGENTUM_TRADER_H
```

```
//
// Created by agustin on 15/6/20.
//

#include "Trader.h"
#include "../../Config/Configuration.h"

#define INITIAL_ITEMS_AMOUNT 10
#define BUYING_PRICE_MULTIPLIER 1.1
#define SELLING_PRICE_MULTIPLIER 0.9

using namespace GameType;

Trader::Trader(Coordinate initialPosition) : Entity(GameType::TRADER, initialPosition, "Trader") {
    std::unordered_set<std::string> acceptedProducts;
    Configuration& config = Configuration::getInstance();
    std::unordered_map<std::string, unsigned int> initialItemsAmounts;

    initialItemsAmounts[config.configWeaponData(LONGSWORD).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configWeaponData(AXE).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configWeaponData(WARHAMMER).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configWeaponData(SIMPLE_BOW).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configWeaponData(COMPOSITE_BOW).name] = INITIAL_ITEMS_AMOUNT;

    initialItemsAmounts[config.configClothingData(LEATHER_ARMOR).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configClothingData(PLATE_ARMOR).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configClothingData(KING_ARMOR).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configClothingData(BLUE_TUNIC).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configClothingData(HOOD).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configClothingData(IRON_HELMET).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configClothingData(TURTLE_SHIELD).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configClothingData(IRON_SHIELD).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configClothingData(MAGIC_HAT).name] = INITIAL_ITEMS_AMOUNT;

    initialItemsAmounts[config.configPotionData(MANA_POTION).name] = INITIAL_ITEMS_AMOUNT;
    initialItemsAmounts[config.configPotionData(HEALTH_POTION).name] = INITIAL_ITEMS_AMOUNT;

    for (const auto & item: initialItemsAmounts) {
        acceptedProducts.emplace(item.first);
    }

    Shop aux(initialItemsAmounts, std::move(acceptedProducts), BUYING_PRICE_MULTIPLIER, SELLING_PRICE_MULTIPLIER);
    shop = std::move(aux);
}

void Trader::list(Player &player) {
    shop.list(player);
}
```

```


void Trader::buy(Player &player, const std::string &itemName) {
    shop.buy(player, itemName);
}

void Trader::sell(Player &player, const std::string& itemName) {
    shop.sell(player, itemName);
}

```

```cpp
//
// Created by agustin on 15/6/20.
//

#ifndef ARGENTUM_STORAGE_H
#define ARGENTUM_STORAGE_H

#include <string>
#include <unordered_map>
#include <list>
#include <utility>
#include <memory>
#include "../../Server/PlayerData.hpp"

class Item;
class Player;

class EntityTests;

/*Clase utilizada para guardar instancias de items, se guardan las instancias
y no cantidades de cada item ya que de esa forma no se debe estar creando y
copiando cada vez que se retira algun item. Esto permite ademas extensibilidad
ya que se podria agregar a cada item por separado modificadores y estos quedarian
guardados en la instancia*/

/*Cualquier tipo de item que sea guardado 0 veces no tiene un lugar en el unordered_map*/

class Storage {
private:
    int32_t storedGold;
    std::unordered_map<std::string, std::list<std::shared_ptr<Item>>> storedItems;

    friend EntityTests;

private:
    static void _addAmmountMessageToPlayer(Player& player, const std::string& itemName,
                                    int concatenatedNumber);

public:
    Storage();

    Storage(Storage∧ other) noexcept;

    Storage& operator=(Storage∧ other) noexcept;

    explicit Storage(const std::unordered_map<std::string, unsigned int>&
                     initialItemsAmounts, unsigned int gold);

    /*Almacena el item, apropiandose de el.
    Retorna true si cambio la cantidad de items guardados, sino retorna false
    La cantidad cambia si se recibe algo que no sea oro*/
    bool storeItem(std::shared_ptr<Item>∧ item);

    /*Intenta pasar el item pedido de Storage a Player, si el item no existe o
    el player no tiene espacio entonces no hace nada
    Retorna true si cambio la cantidad de items que guarda, sino retorna false
    La cantidad cambia si se saca algo que no sea oro*/
    bool retreiveItem(const std::string& itemName, Player& player);

    /*Agrega en el minichat del player la lista con los items disponibles y su
     * precio, la usan los vendedores*/
    void getStorageData(Player& player, const std::unordered_map<std::string,
```

```cpp
                         unsigned int>& prices, float priceMultiplier) const;

    /*Agrega en el minichat del player la lista con los items disponibles,
     * la usa el banker*/
    void getStorageData(Player& player) const;

    /*Indica si el item con el nombre indicado se encuentra guardado*/
    bool isItemAvailable(const std::string& itemName) const;

    /*Aumenta la cantidad de oro guardado en el storage*/
    void increaseGoldReserves(int amount);

    /*Disminuye la cantidad de oro guardado en el storage*/
    bool decreaseGoldReserves(int amount);

    /*Retorna en el struct playerData los datos de los items que el player
     * tiene guardados*/
    void getPlayerData(PlayerData& playerData) const;
};


#endif //ARGENTUM_STORAGE_H
```

```cpp
//
// Created by agustin on 15/6/20.
//

#include "Storage.h"
#include "../../Items/ItemsFactory.h"
#include <utility>
#include "../../Items/Item.h"
#include "../Player.h"
#include "msgpack.hpp"
#include "../../Config/Configuration.h"

MSGPACK_ADD_ENUM(GameType::EventID)

#define UNEXISTING_ITEM_MESSAGE "The requested item is not available\n"

Storage &Storage::operator=(Storage ∧other) noexcept {
    storedGold = other.storedGold;
    other.storedGold = 0;
    storedItems = std::move(other.storedItems);
    return *this;
}

Storage::Storage(Storage ∧other) noexcept {
    storedGold = other.storedGold;
    other.storedGold = 0;
    storedItems = std::move(other.storedItems);
}

Storage::Storage(const std::unordered_map<std::string, unsigned int>&
                 initialItemsAmounts, unsigned int gold) {
    storedGold = gold;
    ItemsFactory& factory = ItemsFactory::getInstance();
    std::shared_ptr<Item> aux;
    for (const auto & initialItemAmount: initialItemsAmounts) {
        for (unsigned int i = 0; i < initialItemAmount.second; ++i) {
            factory.storeItemInstance(initialItemAmount.first, aux);
            storedItems[initialItemAmount.first].push_back(std::move(aux));
        }
    }
}

bool Storage::storeItem(std::shared_ptr<Item> ∧item) {
    if (item) {
        storedItems[item→getName()].push_back(std::move(item));
        return true;
    }
    return false;
}

bool Storage::retreiveItem(const std::string& itemName, Player &player) {
    std::shared_ptr<Item> item;
    if (storedItems.count(itemName) ≡ 1) {
        item = storedItems.at(itemName).front();
        if (¬player.storeItem(item)) {
            return false;
        }
        storedItems[itemName].pop_front();
        if (storedItems[itemName].empty()) {
            storedItems.erase(itemName);
        }
    } else {
        player.addMessage(UNEXISTING_ITEM_MESSAGE);
        return false;
    }
    return true;
```

```cpp
}

void Storage::getStorageData(Player& player, const std::unordered_map<std::string, unsigned int> &prices,
                             float priceMultiplier) const {
    _addAmmountMessageToPlayer(player, Configuration::getInstance().configGetGoldName(), storedGold);
    for (const auto & storedItem : storedItems) {
        _addAmmountMessageToPlayer(player, storedItem.second.front()→getName(),
                                   prices.at(storedItem.first) * priceMultiplier);
    }
}

void Storage::getStorageData(Player& player) const {
    _addAmmountMessageToPlayer(player, Configuration::getInstance().configGetGoldName(), storedGold);
    for (const auto & storedItem : storedItems) {
        _addAmmountMessageToPlayer(player, storedItem.second.front()→getName(),
                                   storedItem.second.size());
    }
}


bool Storage::isItemAvailable(const std::string &itemName) const {
    return storedItems.count(itemName) ≡ 1;
}

void Storage::increaseGoldReserves(int amount) {
    storedGold += amount;
}

bool Storage::decreaseGoldReserves(int amount) {
    if (amount ≤ storedGold) {
        storedGold -= amount;
        return true;
    }
    return false;
}

Storage::Storage() {
    storedGold = 0;
}

void Storage::getPlayerData(PlayerData &playerData) const {
    int stored = 0;
    for (auto & item : storedItems) {
        GameType::ItemType type = item.second.front()→getType();
        int32_t id = item.second.front()→getId();
        for (std::size_t i = 0; i < item.second.size(); ++i) {
            playerData.bankerItems.at(stored) = std::make_tuple(type, id);
            ++stored;
        }
    }
    playerData.bankerGold = storedGold;
}

/////////////////////////////PRIVATE//////////////////////////

void Storage::_addAmmountMessageToPlayer(Player &player, const std::string &itemName,
                                         int concatenatedNumber) {
    player.addMessage(itemName);
    player.addMessage(": ");
    player.addMessage(std::to_string(concatenatedNumber));
    player.addMessage("\n");
```

```
}
```

```
//
// Created by agustin on 16/6/20.
//

#ifndef ARGENTUM_SHOP_H
#define ARGENTUM_SHOP_H

#include "Storage.h"

class Player;

class EntityTests;


class Shop {
private:
    float buyingMultiplier{};
    float sellingMultiplier{};
    Storage storage;
    std::unordered_map<std::string, unsigned int> prices;
    std::unordered_set<std::string> acceptedProducts;

    friend EntityTests;

public:
    Shop();

    Shop(const std::unordered_map<std::string, unsigned int>&
         initialItemsAmounts, std::unordered_set<std::string>^ acceptedProducts
,
                                float buyingMultiplier, float sellingMultiplier)
;

    Shop(Shop^ other) noexcept;

    Shop& operator=(Shop^ other) noexcept;

    /*Almacena en la lista los items que tiene en venta, los precios
    almacenados para cada producto fueron modificados utilizando el
    multiplicador recibido en el constructor*/
    void list(Player &player) const;

    /*Funcion utilizada cuando el jugador quiere comprar un item, si el jugador
no
    tiene oro suficiente o el item pedido no esta guardado entonces no hace nada
*/
    void buy(Player& player, const std::string& itemName);

    /*Funcion utilizada cuando el jugador quiere vender un item, si el NPC compr
ador no
    tiene oro suficiente o el item pedido no esta guardado entonces no hace nada
*/
    void sell(Player& player, const std::string& itemName);
};


#endif //ARGENTUM_SHOP_H
```

```cpp
//
// Created by agustin on 16/6/20.
//

#include "Shop.h"
#include "../Player.h"
#include "../../Config/Configuration.h"


#define PRODUCT_NOT_IN_STORAGE_MESSAGE "I don't have a "
#define NOT_ACCEPTED_PRODUCT_MESSAGE "I don't buy "
#define NOT_ENOUGH_GOLD_STORED_MESSAGE "I don't have enough gold\n"
#define PLAYER_CANT_AFFORD_MESSAGE "You don't have enough gold\n"
#define PLAYER_DOES_NOT_HAVE_SPACE_MESSAGE "You don't have enough space for that item\n"

Shop::Shop() {
    sellingMultiplier = 1;
    buyingMultiplier = 1;
}

Shop::Shop(const std::unordered_map<std::string, unsigned int> &initialItemsAmou
nts,
           std::unordered_set<std::string>∧ _acceptedProducts, float _buyingMul
tiplier, float _sellingMultiplier):
           storage(initialItemsAmounts, Configuration::getInstance().configIniti
alMerchantGold()) {
    acceptedProducts = std::move(_acceptedProducts);
    Configuration& config = Configuration::getInstance();
    const auto & weaponsData = config.configAllWeaponsData();
    const auto & clothesData = config.configAllClothingData();
    const auto & potionsData = config.configAllPotionsData();

    for (const auto & weaponData: weaponsData) {
        prices[weaponData.second.name] = weaponData.second.price;
    }
    for (const auto & clothingData: clothesData) {
        prices[clothingData.second.name] = clothingData.second.price;
    }
    for (const auto & potionData: potionsData) {
        prices[potionData.second.name] = potionData.second.price;
    }
    buyingMultiplier = _buyingMultiplier;
    sellingMultiplier = _sellingMultiplier;
}

Shop &Shop::operator=(Shop ∧other) noexcept {
    storage = std::move(other.storage);
    buyingMultiplier = other.buyingMultiplier;
    sellingMultiplier = other.sellingMultiplier;
    prices = std::move(other.prices);
    acceptedProducts = std::move(other.acceptedProducts);
    return *this;
}

Shop::Shop(Shop ∧other) noexcept {
    storage = std::move(other.storage);
    buyingMultiplier = other.buyingMultiplier;
    sellingMultiplier = other.sellingMultiplier;
    prices = std::move(other.prices);
    acceptedProducts = std::move(other.acceptedProducts);
}

void Shop::list(Player &player) const {
    storage.getStorageData(player, prices, buyingMultiplier);
}

```

```cpp
void Shop::buy(Player &player, const std::string &itemName) {
    unsigned int price;
    if (storage.isItemAvailable(itemName)) {
        price = static_cast<unsigned int>(static_cast<float>(prices[itemName])
                                                    * buyingMultiplier);
        if (¬player.hasFullInventory()) {
            if (player.spendGold(static_cast<int>(price))) {
                storage.increaseGoldReserves(static_cast<int>(price));
                storage.retreiveItem(itemName, player);
            } else {
                player.addMessage(PLAYER_CANT_AFFORD_MESSAGE);
            }
        } else {
            player.addMessage(PLAYER_DOES_NOT_HAVE_SPACE_MESSAGE);
        }
    } else {
        player.addMessage(PRODUCT_NOT_IN_STORAGE_MESSAGE + itemName + "\n");
    }
}

void Shop::sell(Player &player, const std::string& itemName) {
    if (acceptedProducts.count(itemName) ≡ 0) {
        player.addMessage(NOT_ACCEPTED_PRODUCT_MESSAGE + itemName + "s\n");
        return;
    }
    unsigned int price;
    price = static_cast<unsigned int>(static_cast<float>(prices.at(itemName))
                                  * sellingMultiplier);
    if (player.hasItem(itemName)) {
        if (storage.decreaseGoldReserves(static_cast<int>(price))) {
            player.receiveGold(price);
            storage.storeItem(player.removeItem(itemName));
            player.addMessage("You got " + std::to_string(price) + " gold coins\n");
        } else {
            player.addMessage(NOT_ENOUGH_GOLD_STORED_MESSAGE);
        }
    }
}
```

```cpp
//
// Created by agustin on 15/6/20.
//

#ifndef ARGENTUM_PRIEST_H
#define ARGENTUM_PRIEST_H


#include "Shop.h"
#include "../Entity.h"

class EntityTests;

class Priest: public Entity {
private:
    Shop shop;

    friend EntityTests;

public:
    explicit Priest(Coordinate initialPosition);

    /*Retorna por le minichat del player los items que tiene el Priest a la vent
a junto con su precio*/
    void list(Player &player) override;

    /*Le vende al player el item pedido en caso de tenerlo, caso contrario no su
cede nada*/
    void buy(Player& player, const std::string& itemName) override;

    /*Le compra al player el item pedido en caso de tenerlo, caso contrario no s
ucede nada*/
    void sell(Player& player, const std::string& itemName) override;

    /*Cura al Player toda su vida y le recupera la totalidad del mana tambien (n
o le carga mana al Warrior)*/
    void requestHeal(Player& player) override;
};


#endif //ARGENTUM_PRIEST_H
```

```cpp
//
// Created by agustin on 15/6/20.
//

#include "Priest.h"
#include "../../Config/Configuration.h"
#include "../Player.h"

using namespace GameType;

#define INITIAL_WEAPONS_AMOUNT 10
#define INITIAL_POTIONS_AMOUNT 40
#define BUYING_PRICE_MULTIPLIER 1.25
#define SELLING_PRICE_MULTIPLIER 0.75

Priest::Priest(Coordinate initialPosition): Entity(GameType::PRIEST, initialPosi
tion, "Priest") {
    std::unordered_set<std::string> acceptedProducts;
    Configuration& config = Configuration::getInstance();
    std::unordered_map<std::string, unsigned int> initialItemsAmounts;
    initialItemsAmounts[config.configWeaponData(LINKED_STAFF).name] = INITIAL_WE
APONS_AMOUNT;
    initialItemsAmounts[config.configWeaponData(GNARLED_STAFF).name] = INITIAL_W
EAPONS_AMOUNT;
    initialItemsAmounts[config.configWeaponData(ELVEN_FLUTE).name] = INITIAL_WEA
PONS_AMOUNT;
    initialItemsAmounts[config.configWeaponData(ASH_ROD).name] = INITIAL_WEAPONS
_AMOUNT;

    initialItemsAmounts[config.configPotionData(HEALTH_POTION).name] = INITIAL_P
OTIONS_AMOUNT;
    initialItemsAmounts[config.configPotionData(MANA_POTION).name] = INITIAL_POT
IONS_AMOUNT;

    for (const auto & item: initialItemsAmounts) {
        acceptedProducts.emplace(item.first);
    }

    Shop aux(initialItemsAmounts, std::move(acceptedProducts), BUYING_PRICE_MULT
IPLIER, SELLING_PRICE_MULTIPLIER);
    shop = std::move(aux);
}

void Priest::list(Player &player) {
    shop.list(player);
}

void Priest::buy(Player &player, const std::string &itemName) {
    shop.buy(player, itemName);
}

void Priest::sell(Player &player, const std::string& itemName) {
    shop.sell(player, itemName);
}

void Priest::requestHeal(Player &player) {
    player.restoreStats(false);
}

```

```
//
// Created by agustin on 25/6/20.
//

#ifndef ARGENTUM_CITIZENFACTORY_H
#define ARGENTUM_CITIZENFACTORY_H

#include <unordered_map>
#include <memory>
#include "../../../libs/GameEnums.h"
#include "../../Map/Coordinate.h"

class Entity;

typedef void (*CitizenCreator)(std::shared_ptr<Entity>& citizen, Coordinate initialPosition);

/*Factory de citizens*/

class CitizenFactory {
private:
    std::unordered_map<GameType::Entity, CitizenCreator> citizensCreators;

private:
    static void _storeTrader(std::shared_ptr<Entity>& citizen, Coordinate initialPosition);
    static void _storePriest(std::shared_ptr<Entity>& citizen, Coordinate initialPosition);
    static void _storeBanker(std::shared_ptr<Entity>& citizen, Coordinate initialPosition);

public:
    CitizenFactory();

    /*Crea un citizen y lo almacena en el shared_ptr*/
    void storeCitizen(std::shared_ptr<Entity>& citizen, GameType::Entity _type, Coordinate initialPosition);
};


#endif //ARGENTUM_CITIZENFACTORY_H
```

```
//
// Created by agustin on 25/6/20.
//

#include "CitizenFactory.h"
#include "Trader.h"
#include "Priest.h"
#include "Banker.h"

////////////////////////////PRIVATE/////////////////////////////////

void CitizenFactory::_storeTrader(std::shared_ptr<Entity>& citizen, Coordinate initialPosition) {
    citizen = std::make_shared<Trader>(initialPosition);
}


void CitizenFactory::_storePriest(std::shared_ptr<Entity>& citizen, Coordinate initialPosition) {
    citizen = std::make_shared<Priest>(initialPosition);
}

void CitizenFactory::_storeBanker(std::shared_ptr<Entity> &citizen, Coordinate initialPosition) {
    citizen = std::make_shared<Banker>(initialPosition);
}

////////////////////////////PUBLIC/////////////////////////////////

CitizenFactory::CitizenFactory() {
    citizensCreators[GameType::Entity::TRADER] = _storeTrader;
    citizensCreators[GameType::Entity::PRIEST] = _storePriest;
    citizensCreators[GameType::Entity::BANKER] = _storeBanker;
}

void CitizenFactory::storeCitizen(std::shared_ptr<Entity> &citizen, GameType::Entity _type,
                                  Coordinate initialPosition) {
    citizensCreators.at(_type)(citizen, initialPosition);
}
```

```
//
// Created by agustin on 15/6/20.
//

#ifndef ARGENTUM_BANKER_H
#define ARGENTUM_BANKER_H

#include <unordered_map>
#include "../Entity.h"
#include "../../Server/PlayerData.hpp"

class Storage;

class EntityTests;

class Banker;
typedef void (*ModifyGold)(Storage &playerStorage, Player &player, int goldAmount);

/*Clase que se encarga de guardar todos los items que le otorgue el jugador
El jugador puede darle oro o items nuevos para que guarde o sacar oro o items*/
class Banker: public Entity {
private:
    static std::unordered_map<std::string, std::pair<unsigned int, Storage>> playersStorages;

    friend EntityTests;

private:
    static int32_t _getNumberOfItemsStored(const std::unordered_map<std::string, unsigned int>&
                                                       initialItemsAmounts) ;
    static void _storeAvailableRoomMessage(Player &player, unsigned int storedItemsAmount);

    static void _modifyGoldReserves(Storage& playerStorage, Player &player,
                        const std::string& itemName, ModifyGold modifier);

    static std::string _translateItemTypeToName(std::tuple<GameType::ItemType, int32_t> item);

    static void _depositGold(Storage &playerStorage, Player &player, int goldAmount);

    static void _withdrawGold(Storage &playerStorage, Player &player, int goldAmount);

public:
    explicit Banker(Coordinate initialPosition);

    /*Almacena una lista con los datos de los items y oro almacenados*/
    void list(Player &player) override;

    /*Intenta sacar el item del banco y almacenarlo en el inventario del jugador
    Si el item no esta guardado en el banco o el judador no tiene espacio en
    su inventario entonces no hace nada
    Si se intenta sacar mas oro del guardado, entonces saca el oro disponible*/
    void withdraw(Player& player, const std::string& itemName) override;

    /*Intenta guardar el item en el banco, sacandolo del inventario del jugador
    Si el item no esta en el inventario del judador entonces no hace nada
    No se pueden guardar items equipados
    Si se intenta depositar mas oro del guardado, entonces deposita la totalidad del oro*/
    void deposit(Player& player, const std::string& itemName) override;

```

```
    /*Carga en el banker los items provenientes del player provenientes del archivo,
     * esto se hace cuando un player se conecta*/
    static void addPlayerItems(const PlayerData& playerData);

    /*Borra del banker los items almacenados por el player, esto se hace cuando un player se desconecta*/
    static void erasePlayerItems(const std::string& playerNickname);

    /*Retorna los items del player almacenados para ser guardados en el archivo de persistencia*/
    static void getPlayerItems(PlayerData& playerData);
};


#endif //ARGENTUM_BANKER_H
```

```cpp
//
// Created by agustin on 15/6/20.
//

#include "Banker.h"

#include "../Player.h"
#include "Storage.h"
#include "../../../libs/TPException.h"
#include "../../Config/Configuration.h"
#include "../../Server/NonModifiableConstants.h"

#define NO_ROOM_AVAILABLE_MESSAGE "You don’t have more storage room, the limit is "
#define ROOM_AVAILABLE_MESSAGE "Items stored: "
#define NO_ITEM_MESSAGE "You don’t have that item in you inventory\n"
#define INVALID_GOLD_PARAMETERS "Invalid parameters for gold deposit/withdrawal\n"
#define INSUFFICIENT_GOLD_MESSAGE "Insufficient gold\n"
#define NEGATIVE_GOLD_MESSAGE "Negative gold does not exist\n"
#define GOLD_AMOUNT_SEPARATOR ' '

std::unordered_map<std::string, std::pair<unsigned int, Storage>> Banker::playersStorages;

Banker::Banker(Coordinate initialPosition): Entity(GameType::BANKER,
                                                   initialPosition, "Banker") {

}

void Banker::list(Player &player) {
    const std::pair<unsigned int, Storage>& aux = playersStorages.at(player.getNickname());
    aux.second.getStorageData(player);
    _storeAvailableRoomMessage(player, aux.first);
}


void Banker::withdraw(Player &player, const std::string &itemName) {
    try {
        std::pair<unsigned int, Storage>& aux = playersStorages.at(player.getNickname());
        if (itemName.find(Configuration::getInstance().configGetGoldName()) ≠ std::string::npos) {
            _modifyGoldReserves(aux.second, player, itemName, _withdrawGold);
        } else if (aux.second.retreiveItem(itemName, player)) {
            aux.first--;
            _storeAvailableRoomMessage(player, aux.first);
        }
    } catch (...) {
        throw TPException("Tried to withdraw an item from a player that"
                          "didnt exist!");
    }
}

void Banker::deposit(Player &player, const std::string& itemName) {
    try {
        std::pair<unsigned int, Storage>& aux = playersStorages.at(player.getNickname());
        if (itemName.find(Configuration::getInstance().configGetGoldName()) ≠ std::string::npos) {
            _modifyGoldReserves(aux.second, player, itemName, _depositGold);
        } else if (aux.first < BANK_SIZE) {
            if (playersStorages.at(player.getNickname()).second.storeItem(player.removeItem(itemName))) {
                aux.first++;
                _storeAvailableRoomMessage(player, aux.first);
            } else {
```

```cpp
                player.addMessage(NO_ITEM_MESSAGE);
            }
        } else {
            player.addMessage(NO_ROOM_AVAILABLE_MESSAGE + std::to_string(BANK_SIZE) + "\n");
        }
    } catch(...) {
        throw TPException("Tried to deposit an item of an inexistent player!");
    }
}

void Banker::getPlayerItems(PlayerData &playerData) {
    auto & playerStorageData = playersStorages.at(playerData.nickname);
    playerStorageData.second.getPlayerData(playerData);
}

void Banker::addPlayerItems(const PlayerData &playerData) {
    std::unordered_map<std::string, unsigned int> initialItemsAmounts;
    unsigned int itemAmount = 0;
    std::tuple<GameType::ItemType, int32_t> currItemType = playerData.bankerItems.at(0);
    for (auto & item : playerData.bankerItems) {
        if (currItemType ≠ item) {
            std::string itemName = _translateItemTypeToName(currItemType);
            initialItemsAmounts.emplace(itemName, itemAmount);
            itemAmount = 0;
            currItemType = item;

        }
        ++itemAmount;
    }
    unsigned int gold = playerData.bankerGold;
    playersStorages.emplace(playerData.nickname, std::pair<int32_t, Storage>
            (_getNumberOfItemsStored(initialItemsAmounts), Storage(initialItemsAmounts, gold)));
}


void Banker::erasePlayerItems(const std::string& playerNickname) {
    playersStorages.erase(playerNickname);
}


/////////////////////////////////PRIVATE/////////////////////////////////////////

int32_t Banker::_getNumberOfItemsStored(const std::unordered_map<std::string, unsigned int> &
                                        initialItemsAmounts) {
    int32_t storedItemsAmount = 0;
    for (const auto & itemList: initialItemsAmounts) {
        storedItemsAmount += itemList.second;
    }
    return storedItemsAmount;
}

void Banker::_storeAvailableRoomMessage(Player &player, unsigned int storedItemsAmount) {
    player.addMessage(ROOM_AVAILABLE_MESSAGE + std::to_string(storedItemsAmount) + "/"
                      + std::to_string(BANK_SIZE) + "\n");
}

void Banker::_modifyGoldReserves(Storage& playerStorage, Player &player,
                        const std::string& itemName, ModifyGold modifyGold) {
    int goldAmount = 0;
    size_t separatorPosition = itemName.find(GOLD_AMOUNT_SEPARATOR);
    if ((separatorPosition ≠ std::string::npos) ∧
        (separatorPosition ≠ itemName.size() − 1)) {
        try {
            goldAmount = std::stoi(itemName.substr(separatorPosition + 1));
            modifyGold(playerStorage, player, goldAmount);
        } catch (std::invalid_argument &e) {
            player.addMessage(INVALID_GOLD_PARAMETERS);
        } catch (std::out_of_range &e) {
            player.addMessage(INVALID_GOLD_PARAMETERS);
        }
    } else {
        player.addMessage(INVALID_GOLD_PARAMETERS);
    }
}

void Banker::_depositGold(Storage &playerStorage, Player &player, int goldAmount
) {
    if (goldAmount ≥ 0) {
        if (player.spendGold(goldAmount)) {
            playerStorage.increaseGoldReserves(goldAmount);
        } else {
            player.addMessage(INSUFFICIENT_GOLD_MESSAGE);
        }
    } else {
        player.addMessage(NEGATIVE_GOLD_MESSAGE);
    }
}

void Banker::_withdrawGold(Storage &playerStorage, Player &player, int goldAmoun
t) {
    if (goldAmount ≥ 0) {
        if (playerStorage.decreaseGoldReserves(goldAmount)) {
            player.receiveGold(goldAmount);
        } else {
            player.addMessage(INSUFFICIENT_GOLD_MESSAGE);
        }
    } else {
        player.addMessage(NEGATIVE_GOLD_MESSAGE);
    }
}

std::string Banker::_translateItemTypeToName(std::tuple<GameType::ItemType, int3
2_t> item) {
    Configuration& config = Configuration::getInstance();
    switch (std::get<0>(item)) {
        case GameType::ITEM_TYPE_CLOTHING:
            return config.configClothingData(
                    static_cast<GameType::Clothing>(std::get<1>(item))).name;
        case GameType::ITEM_TYPE_WEAPON:
            return config.configWeaponData(
                    static_cast<GameType::Weapon>(std::get<1>(item))).name;
        case GameType::ITEM_TYPE_POTION:
            return config.configPotionData(
                    static_cast<GameType::Potion>(std::get<1>(item))).name;
        default:
            //do nothing
            throw TPException("Player was storing invalid banker items!");
    }
}
```

```
//
// Created by marcos on 18/6/20.
//

#ifndef ARGENTUM_ATTACKRESULT_H
#define ARGENTUM_ATTACKRESULT_H

struct AttackResult {
    int damage;
    unsigned int experience;
    std::string resultMessage;
};

#endif //ARGENTUM_ATTACKRESULT_H
```

```cpp
//
// Created by marcos on 6/15/20.
//

#ifndef ARGENTUM_MAPFILEREADER_H
#define ARGENTUM_MAPFILEREADER_H

#include <fstream>
#include "json.hpp"
#include <unordered_map>

struct TileInfo {
    std::string tileType;
    std::string structureType;
    std::string entityType;
    bool isOccupable;
    bool isFromCity;
};

struct MapSize {
    unsigned int width;
    unsigned int height;
};

class MapFileReader {
private:
    nlohmann::json obj;
    std::unordered_map<int, std::string> mapElements;
    MapSize mapDimensions{};

public:
    explicit MapFileReader(const std::string& path);

    /*Retorna los datos del tile (si es ocupable, pertence a una ciudad, tiene u
n NPC/estructura*/
    TileInfo getTileInfo(unsigned int x, unsigned int y);

    /*Retorna las dimensiones MxN del mapa*/
    MapSize getMapDimensions() const;

private:
    void _readMapSize();
    void _readIDs();
};


#endif //ARGENTUM_MAPFILEREADER_H
```

```cpp
//
// Created by marcos on 6/15/20.
//

#include "MapFileReader.h"
#include "../../libs/TPException.h"
#include <memory>

using json = nlohmann::json;

MapFileReader::MapFileReader(const std::string& path) {
    mapDimensions.width = 0;
    mapDimensions.height = 0;
    std::ifstream file(path);
    if (¬file.is_open()) {
        throw TPException("Could not open Map File, check whether"
                          " it exists or not!");
    }
    try {
        file >> obj;
    } catch (...) {
        throw TPException("Map file parsing failed!");
    }
    _readMapSize();
    _readIDs();
}

void MapFileReader::_readMapSize() {
    mapDimensions.width = obj["width"].get<int>();
    mapDimensions.height = obj["height"].get<int>();
}

void MapFileReader::_readIDs() {
    json& tilesets = obj["tilesets"];
    mapElements.emplace(0, "Nothing");
    for (auto & tileset : tilesets) {
        std::string name = tileset["name"].get<std::string>();
        int id = tileset["firstgid"].get<int>();
        int tilecount = tileset["tilecount"].get<int>();
        if (tilecount > 1) {
            for (int i = 0; i < tilecount; ++i) {
                mapElements.emplace(id + i, name + std::to_string(i));
            }
        } else {
            mapElements.emplace(id, name);
        }
    }
}

TileInfo MapFileReader::getTileInfo(unsigned int row, unsigned int column) {
    json& layers = obj["layers"];
    json& tileData = layers[0]["data"];
    TileInfo tile;
    tile.tileType = mapElements.at(tileData[row*mapDimensions.width + column].ge
t<int>());
    json& sData = layers[1]["data"];
    tile.structureType = mapElements.at(sData[row*mapDimensions.width + column].
get<int>());
    json& eData = layers[2]["data"];
    tile.entityType = mapElements.at(eData[row*mapDimensions.width + column].get
<int>());
    json& oData = layers[3]["data"]; /*isOccupable*/
    tile.isOccupable = (oData[row*mapDimensions.width + column].get<int>() ≡ 0);
    json& cData = layers[4]["data"]; /*isFromCity*/
    tile.isFromCity = (cData[row*mapDimensions.width + column].get<int>() ≠ 0);
    return tile;
}

MapSize MapFileReader::getMapDimensions() const {
    return mapDimensions;
}
```

```
//
// Created by ivan on 8/6/20.
//

#ifndef ARGENTUM_CONFIGURATION_H
#define ARGENTUM_CONFIGURATION_H

#include "ConfigFileReader.h"
#include <unordered_map>

/*La siguiente clase toma los valores que lee el ConfigFileReader y los guarda e
n memoria para que el resto del
 * juego pueda acceder a ellos rapidamente*/

class Configuration {

private:
    std::unordered_map<GameType::Race, Config::Modifiers> raceModifiers{};
    std::unordered_map<GameType::Class, Config::Modifiers> classModifiers{};
    std::unordered_map<GameType::Entity, Config::MonsterStats> monsterStats{};
    std::unordered_map<GameType::Weapon, Config::WeaponData> weaponData{};
    std::unordered_map<GameType::Clothing, Config::ClothingData> clothingData{};
    std::unordered_map<GameType::Potion, Config::PotionData> potionData{};

    Config::GoldModifiers goldModifiers{};

    Config::XPModifiers xpModifiers{};

    float criticalAttackChance;
    float dodgeChance;

    unsigned int newbieLevel;
    unsigned int maxLevelDif;

    unsigned int timeBetweenMonsterSpawns{};
    unsigned int monsterSpawnAmount{};
    unsigned int maxMonsterAmount{};

    unsigned int initialMerchantGold;
    std::string goldName;

    unsigned int playerSpeed;
    double timeForPlayerRecovery;

    std::string port;
    std::string mapPath;
    std::string savePath;
    std::string indexPath;

public:
    Configuration(Configuration const&) = delete;
    void operator=(Configuration const&) = delete;

    /*Retorna la instancia Singleton del Configuration*/
    static Configuration& getInstance();

    /*Retorna los modificadores de clase*/
    const Config::Modifiers& configClassModifiers(GameType::Class _class) const;

    /*Retorna los modificadores de clase*/
    const Config::Modifiers& configRaceModifiers(GameType::Race race) const;

    /*Retorna los modificadores de clase*/
    const Config::MonsterStats& configMonsterStats(GameType::Entity monster) con
st;

```

```
    /*Retorna los modificadores de clase*/
    const Config::WeaponData& configWeaponData(GameType::Weapon weapon) const;

    /*Retorna los modificadores de clase*/
    const Config::ClothingData& configClothingData(GameType::Clothing clothes) const;

    /*Retorna los modificadores de clase*/
    const Config::PotionData& configPotionData(GameType::Potion potion) const;

    /*Retorna los modificadores de clase*/
    const Config::GoldModifiers& configGoldModifiers() const;

    /*Retorna los modificadores de clase*/
    const Config::XPModifiers& configXPModifiers() const;

    /*Retorna los modificadores de clase*/
    const std::unordered_map<GameType::Weapon, Config::WeaponData>& configAllWeaponsData();

    /*Retorna los modificadores de clase*/
    const std::unordered_map<GameType::Clothing, Config::ClothingData>& configAllClothingData();

    /*Retorna los modificadores de clase*/
    const std::unordered_map<GameType::Potion, Config::PotionData>& configAllPotionsData();

    /*Retorna la probabilidad de un ataque critico*/
    float configCriticalAttackChance() const;

    /*Retorna un coeficiente que afecta la probabilidad de esquivar un ataque*/
    float configDodgeChance() const;

    /*Retorna el newbie level*/
    unsigned int configNewbieLevel() const;

    /*Retorna la maxima diferencia que puede haber entre 2 players para que puedan atacarse*/
    unsigned int configMaxLevelDif() const;

    /*Retorna el tiempo entre spawns de monstruos*/
    unsigned int configTimeBetweenMonsterSpawns() const;

    /*Retorna la maxima cantidad de monstruos que puede haber en simultaneo vivos en la partida*/
    unsigned int configMaxMonsterAmount() const;

    /*Retorna la cantidad de monstruos que spawnean a la vez en cada spawn*/
    unsigned int configMonsterSpawnAmount() const;

    /*Retorna la cantidad inicial de oro de los mercaderes*/
    unsigned int configInitialMerchantGold() const;

    /*Retorna el nombre a mostrar del oro (mostramos Gold pero podria ponerse un nombre mas tuneado*/
    const std::string &configGetGoldName() const;

    /*Retorna la velocidad del player para moverse*/
    unsigned int configPlayerSpeed() const;

    /*Retorna el tiempo (en segundos) para que el player recupere vida/mana*/
    double configPlayerRecoveryTime() const;

    const std::string& configPort() const;

```

```
    const std::string& configMapPath() const;

    const std::string& configSavePath() const;

    const std::string& configIndexPath() const;

private:
    Configuration();
};


#endif //ARGENTUM_CONFIGURATION_H
```

```cpp
//
// Created by ivan on 8/6/20.
//

#include "Configuration.h"

using namespace GameType;

Configuration& Configuration::getInstance() {
    static Configuration instance;
    return instance;
}

Configuration::Configuration() {
    Config::ConfigFileReader fileReader("/etc/Argentum/config.json");

    fileReader.loadClassModifiers(classModifiers);
    fileReader.loadRaceModifiers(raceModifiers);
    fileReader.loadMonsterStats(monsterStats);

    fileReader.loadWeaponData(weaponData);
    fileReader.loadClothingData(clothingData);

    fileReader.loadGoldModifiers(goldModifiers);
    fileReader.loadXPModifiers(xpModifiers);

    fileReader.loadPotionData(potionData);

    fileReader.loadMonsterSpawnData(maxMonsterAmount, timeBetweenMonsterSpawns,
                                    monsterSpawnAmount);

    initialMerchantGold = fileReader.loadInitialMerchantGold();

    criticalAttackChance = fileReader.loadCritAttackChance();
    dodgeChance = fileReader.loadDodgeChance();

    newbieLevel = fileReader.loadNewbieLevel();
    maxLevelDif = fileReader.loadmaxLevelDif();
    playerSpeed = fileReader.loadPlayerSpeed();
    timeForPlayerRecovery = fileReader.loadTimeForPlayerRecovery();
    port = fileReader.loadPort();
    mapPath = fileReader.loadMapPath();
    savePath = fileReader.loadSavePath();
    indexPath = fileReader.loadIndexPath();
    goldName = "Gold";
}

const Config::Modifiers& Configuration::configClassModifiers(Class _class) const
 {
    return classModifiers.at(_class);
}

const Config::Modifiers& Configuration::configRaceModifiers(Race race) const{
    return raceModifiers.at(race);
}

const Config::MonsterStats& Configuration::configMonsterStats(GameType::Entity m
onster) const{
    return monsterStats.at(monster);
}

const Config::WeaponData& Configuration::configWeaponData(Weapon weapon) const{
    return weaponData.at(weapon);
}

const Config::ClothingData& Configuration::configClothingData(Clothing clothes)
```

```cpp
const{
    return clothingData.at(clothes);
}

const Config::GoldModifiers& Configuration::configGoldModifiers() const{
    return goldModifiers;
}

const Config::XPModifiers& Configuration::configXPModifiers() const{
    return xpModifiers;
}

const Config::PotionData& Configuration::configPotionData(Potion potion) const{
    return potionData.at(potion);
}

float Configuration::configCriticalAttackChance() const{
    return criticalAttackChance;
}

float Configuration::configDodgeChance() const{
    return dodgeChance;
}

unsigned int Configuration::configNewbieLevel() const{
    return newbieLevel;
}

unsigned int Configuration::configMaxLevelDif() const{
    return maxLevelDif;
}

unsigned int Configuration::configTimeBetweenMonsterSpawns() const {
    return timeBetweenMonsterSpawns;
}

unsigned int Configuration::configMaxMonsterAmount() const {
    return maxMonsterAmount;
}

unsigned int Configuration::configMonsterSpawnAmount() const {
    return monsterSpawnAmount;
}

unsigned int Configuration::configInitialMerchantGold() const {
    return initialMerchantGold;
}

const std::string& Configuration::configGetGoldName() const {
    return goldName;
}

const std::unordered_map<GameType::Weapon, Config::WeaponData> &
                                Configuration::configAllWeaponsData() {
    return weaponData;
}

const std::unordered_map<GameType::Clothing, Config::ClothingData> &
                                Configuration::configAllClothingData() {
    return clothingData;
}

const std::unordered_map<GameType::Potion, Config::PotionData> &
                                Configuration::configAllPotionsData() {
    return potionData;
}
```

```cpp

unsigned int Configuration::configPlayerSpeed() const {
    return playerSpeed;
}

double Configuration::configPlayerRecoveryTime() const {
    return timeForPlayerRecovery;
}

const std::string &Configuration::configPort() const {
    return port;
}

const std::string &Configuration::configMapPath() const {
    return mapPath;
}

const std::string &Configuration::configSavePath() const {
    return savePath;
}

const std::string &Configuration::configIndexPath() const {
    return indexPath;
}
```

```cpp
//
// Created by ivan on 8/6/20.
//

#ifndef ARGENTUM_CONFIGFILEREADER_H
#define ARGENTUM_CONFIGFILEREADER_H

#include <fstream>
#include <unordered_map>
#include "../../libs/GameEnums.h"
#include "json.hpp"

namespace Config {

    struct Modifiers {
        unsigned int lifeMultiplier;
        unsigned int manaMultiplier;
        unsigned int constitution;
        unsigned int intelligence;
        unsigned int agility;
        unsigned int strength;
        unsigned int meditationRate;
        unsigned int recoveryRate;
    };

    struct WeaponData {
        std::string name;
        int minDmg;
        int maxDmg;
        unsigned int manaConsumption;
        unsigned int range;
        unsigned int price;
    };

    struct ClothingData {
        std::string name;
        unsigned int minDefense;
        unsigned int maxDefense;
        unsigned int price;
    };

    struct MonsterStats {
        int life;
        unsigned int damage;
        unsigned int constitution;
        unsigned int agility;
        unsigned int strength;
        unsigned int rangeOfVision;
        unsigned int minLevel;
        unsigned int maxLevel;
        unsigned int reactionSpeed;
        unsigned int speed;
    };

    struct PotionData {
        std::string name;
        unsigned int recoveryValue;
        unsigned int price;
    };

    struct GoldModifiers {
        float goldDropFactorMin;
        float goldDropFactorMax;
        unsigned int safeGoldFactor;
        float safeGoldLevelModifier;
    };

    struct XPModifiers {
        unsigned int nextLevelFactor;
        float nextLevelModifier;
        unsigned int attackXPModifier;
        unsigned int killXPModifier;
        float killXPMinRange;
        float killXPMaxRange;
    };

    /*La clase ConfigFileReader es la encargada de parsear el archivo de configu
racion de json que contiene los datos
     * tuneables del juego*/

    class ConfigFileReader {
    private:
        nlohmann::json obj;
        std::unordered_map<std::string, GameType::Class> classes;
        std::unordered_map<std::string, GameType::Race> races;
        std::unordered_map<std::string, GameType::Entity> monsters;
        std::unordered_map<std::string, GameType::Weapon> weapons;
        std::unordered_map<std::string, GameType::Clothing> clothing;
        std::unordered_map<std::string, GameType::Potion> potions;

    public:
        explicit ConfigFileReader(const std::string &path);

        /*Carga los modificadores de las clases*/
        void loadClassModifiers(std::unordered_map<GameType::Class, Modifiers> &
mods);

        /*Carga los modificadores de las razas*/
        void loadRaceModifiers(std::unordered_map<GameType::Race, Modifiers> &mo
ds);

        /*Carga los stats de los monsters*/
        void loadMonsterStats(std::unordered_map<GameType::Entity, MonsterStats>
 &stats);

        /*Carga los datos de las armas*/
        void loadWeaponData(std::unordered_map<GameType::Weapon, WeaponData> &st
ats);

        /*Carga los datos de la ropa*/
        void loadClothingData(std::unordered_map<GameType::Clothing, ClothingDat
a> &stats);

        /*Carga los modificadores del oro*/
        void loadGoldModifiers(GoldModifiers &goldModifiers);

        /*Carga los modificadores de la XP*/
        void loadXPModifiers(XPModifiers &xpModifiers);

        /*Carga la probabilidad de ataque critico*/
        float loadCritAttackChance();

        /*Carga la probabilidad de esquivar un ataque*/
        float loadDodgeChance();

        /*Carga el nivel de newbie de player*/
        unsigned int loadNewbieLevel();

        /*Carga la maxima diferencia de nivel permitida para el ataque entre pla
yers*/
        unsigned int loadmaxLevelDif();

```

```
        /*Carga la velocidad de los players para caminar*/
        unsigned int loadPlayerSpeed();

        /*Carga la data de las pociones (vida/mana recuperado)*/
        void loadPotionData(std::unordered_map<GameType::Potion, PotionData>& st
ats);

        /*Carga la data de spawn de los monsters (cada cuanto respawnean, cantid
ad que respawnean, etc)*/
        void loadMonsterSpawnData(unsigned int &maxMonsterAmount,
                                  unsigned int &timeBetweenMonsterSpawns,
                                  unsigned int &monsterSpawnAmount);

        /*Carga el oro incial que tienen los mercaderes (para comprar items de l
os players)*/
        unsigned int loadInitialMerchantGold();

        /*Carga el tiempo (en segundos) que debe pasar para que el player recupe
re vida/mana*/
        double loadTimeForPlayerRecovery();

        std::string loadPort();
        std::string loadMapPath();
        std::string loadSavePath();
        std::string loadIndexPath();

    private:
        static void
        _getModifiers(Modifiers &modifier, nlohmann::json &currModifier);

        static void
        _getMonsterStats(MonsterStats &stats, nlohmann::json &currMonster);

        static void
        _getWeaponData(WeaponData &stats, nlohmann::json &currWeapon);

        static void
        _getClothingData(ClothingData &stats, nlohmann::json &currClothing);

        static void _getPotionData(PotionData &stats, nlohmann::json &currPotion
);
    };
}

#endif //ARGENTUM_CONFIGFILEREADER_H
```

```cpp
//
// Created by ivan on 8/6/20.
//

#include "ConfigFileReader.h"
#include "../../libs/TPException.h"
#include <memory>

using namespace GameType;
using json = nlohmann::json;

Config::ConfigFileReader::ConfigFileReader(const std::string& path) :
    classes{{"Warrior", WARRIOR}, {"Wizard", WIZARD}, {"Paladin", PALADIN},
           {"Cleric", CLERIC}},
    races{{"Human", HUMAN}, {"Elf", ELF},{"Dwarf", DWARF},
          {"Gnome", GNOME}},
    monsters{{"Skeleton", SKELETON}, {"Zombie", ZOMBIE},{"Spider", SPIDER},
             {"Goblin", GOBLIN}},
    weapons{{"Longsword", LONGSWORD},{"Axe", AXE}, {"Warhammer", WARHAMMER},
            {"AshRod", ASH_ROD}, {"ElvenFlute", ELVEN_FLUTE},
            {"LinkedStaff", LINKED_STAFF},
            {"SimpleBow", SIMPLE_BOW}, {"CompositeBow", COMPOSITE_BOW},
            {"GnarledStaff", GNARLED_STAFF},
            {"Fist", FIST}},
    clothing{{"CommonClothing", COMMON_CLOTHING},{"LeatherArmor", LEATHER_ARMOR},
             {"PlateArmor", PLATE_ARMOR},
             {"KingArmor", KING_ARMOR},{"BlueTunic", BLUE_TUNIC},
             {"Hood", HOOD}, {"IronHelmet", IRON_HELMET},
            {"TurtleShield", TURTLE_SHIELD}, {"IronShield", IRON_SHIELD},
            {"MagicHat", MAGIC_HAT},
            {"NoHelmet", NO_HELMET}, {"NoShield", NO_SHIELD}},
    potions {{"HealthPotion", HEALTH_POTION}, {"ManaPotion", MANA_POTION}} {

    std::ifstream file(path);
    if (¬file.is_open()) {
        throw TPException("Could not open Config File, check whether"
                          " it exists or not");
    }
    try {
        file >> obj;
    } catch (...) {
        throw TPException("Json Config File parsing failed!");
    }
}

void Config::ConfigFileReader::loadClassModifiers(std::unordered_map<Class, Modifiers>& mods) {
    json& classModifiers = obj["Class"];
    Modifiers currMods{};
    for (auto & classModifier : classModifiers) {
       _getModifiers(currMods, classModifier);
        currMods.meditationRate = classModifier["MeditationRate"].get<uint>();
        currMods.recoveryRate = 0;
        mods.emplace(classes.at(classModifier["Name"].get<std::string>()), currMods);
    }
}

void Config::ConfigFileReader::loadRaceModifiers(std::unordered_map<Race, Modifiers>& mods) {
    json& raceModifiers = obj["Race"];
    Modifiers currMods{};
    for (auto & raceModifier : raceModifiers) {
        _getModifiers(currMods, raceModifier);
        currMods.recoveryRate = raceModifier["RecoveryRate"].get<uint>();
        currMods.meditationRate = 0;
```

```cpp
        mods.emplace(races.at(raceModifier["Name"].get<std::string>()), currMods);
    }
}

void Config::ConfigFileReader::loadWeaponData(std::unordered_map<Weapon, WeaponData>& stats) {
    json& weaponsStats = obj["Weapon"];
    WeaponData currStats{};
    for (auto & weaponStat : weaponsStats) {
        _getWeaponData(currStats, weaponStat);
        stats.emplace(weapons.at(weaponStat["Name"].get<std::string>()), currStats);
    }
}

void Config::ConfigFileReader::loadClothingData(std::unordered_map<Clothing, ClothingData>& stats) {
    json& clothingsStats = obj["Clothing"];
    ClothingData currStats{};
    for (auto & clothingStat : clothingsStats) {
        _getClothingData(currStats, clothingStat);
        stats.emplace(clothing.at(clothingStat["Name"].get<std::string>()), currStats);
    }
}

void Config::ConfigFileReader::loadPotionData(std::unordered_map<Potion, PotionData>& stats) {
    json& potionData = obj["Potion"];
    PotionData currPotion{};
    for (auto & potion : potionData) {
        _getPotionData(currPotion, potion);
        stats.emplace(potions.at(potion["Name"].get<std::string>()), currPotion);
    }
}


void Config::ConfigFileReader::loadMonsterStats(std::unordered_map<GameType::Entity, MonsterStats>& stats) {
    json& monsterStats = obj["Monster"];
    MonsterStats currStats{};
    for (auto & monsterStat : monsterStats) {
        _getMonsterStats(currStats, monsterStat);
        stats.emplace(monsters.at(monsterStat["Name"].get<std::string>()), currStats);
    }
}

void Config::ConfigFileReader::loadGoldModifiers(GoldModifiers& goldModifiers) {
    json& modifiers = obj["GoldModifiers"];
    goldModifiers.safeGoldFactor = modifiers["MaxSafeGoldFactor"].get<uint>();
    goldModifiers.safeGoldLevelModifier = modifiers["MaxGoldLevelModifier"]
            .get<float>();
    goldModifiers.goldDropFactorMin = modifiers["MinRange"].get<float>();
    goldModifiers.goldDropFactorMax = modifiers["MaxRange"].get<float>();
}

void Config::ConfigFileReader::loadXPModifiers(XPModifiers& xpModifiers) {
    json& modifiers = obj["XPModifiers"];
    xpModifiers.attackXPModifier = modifiers["AttackXPModifier"].get<unsigned int>();
    xpModifiers.killXPMinRange = modifiers["MinKillXPModifier"].get<float>();
    xpModifiers.killXPMaxRange = modifiers["MaxKillXPModifier"].get<float>();
    xpModifiers.nextLevelModifier = modifiers["NextLevelModifier"].get<float>();
    xpModifiers.nextLevelFactor = modifiers["NextLevelFactor"].get<unsigned int>();
    xpModifiers.killXPModifier = modifiers["KillXPModifier"].get<unsigned int>();
}

float Config::ConfigFileReader::loadCritAttackChance() {
    return obj["CritAttackProb"].get<float>();
}

float Config::ConfigFileReader::loadDodgeChance() {
    return obj["DodgeCoeff"].get<float>();
}

unsigned int Config::ConfigFileReader::loadNewbieLevel() {
    return obj["NewbieLevel"].get<unsigned int>();
}

unsigned int Config::ConfigFileReader::loadmaxLevelDif() {
    return obj["MaxLevelDif"].get<unsigned int>();
}

void Config::ConfigFileReader::_getModifiers(Modifiers& modifier, json& currModi
fier){
    modifier.lifeMultiplier = currModifier["Life"].get<unsigned int>();
    modifier.manaMultiplier = currModifier["Mana"].get<unsigned int>();
    modifier.constitution = currModifier["Constitution"].get<unsigned int>();
    modifier.intelligence = currModifier["Intelligence"].get<unsigned int>();
    modifier.agility = currModifier["Agility"].get<unsigned int>();
    modifier.strength = currModifier["Strength"].get<unsigned int>();
}

void Config::ConfigFileReader::_getMonsterStats(MonsterStats& stats, json& currM
onster){
    stats.life = currMonster["Life"].get<unsigned int>();
    stats.damage = currMonster["Damage"].get<unsigned int>();
    stats.rangeOfVision = currMonster["VisionRange"].get<unsigned int>();
    stats.minLevel = currMonster["LevelMin"].get<unsigned int>();
    stats.maxLevel = currMonster["LevelMax"].get<unsigned int>();
    stats.constitution = currMonster["Constitution"].get<unsigned int>();
    stats.agility = currMonster["Agility"].get<unsigned int>();
    stats.strength = currMonster["Strength"].get<unsigned int>();
    stats.reactionSpeed = currMonster["ReactionSpeed"].get<unsigned int>();
    stats.speed = currMonster["Speed"].get<unsigned int>();
}

void Config::ConfigFileReader::_getWeaponData(WeaponData& stats, json& currWeapo
n){
    stats.name = currWeapon["Name"].get<std::string>();
    stats.maxDmg = currWeapon["MaxDmg"].get<int>();
    stats.minDmg = currWeapon["MinDmg"].get<int>();
    stats.manaConsumption = currWeapon["ManaConsumption"].get<unsigned int>();
    stats.range = currWeapon["Range"].get<unsigned int>();
    stats.price = currWeapon["Price"].get<unsigned int>();
}

void Config::ConfigFileReader::_getClothingData(ClothingData& stats, json&
                                        currClothing){
    stats.name = currClothing["Name"].get<std::string>();
    stats.maxDefense = currClothing["MaxDefense"].get<unsigned int>();
    stats.minDefense = currClothing["MinDefense"].get<unsigned int>();
    stats.price = currClothing["Price"].get<unsigned int>();
}

void Config::ConfigFileReader::_getPotionData(PotionData& stats, json&
                                        currPotion){
    stats.name = currPotion["Name"].get<std::string>();
    stats.recoveryValue = currPotion["RecoveryValue"].get<unsigned int>();
```

```cpp
    stats.price = currPotion["Price"].get<unsigned int>();
}

void Config::ConfigFileReader::loadMonsterSpawnData(unsigned int &maxMonsterAmou
nt,
                                                    unsigned int &timeBetweenMonsterS
pawns,
                                                    unsigned int &monsterSpawnAmount)
 {
    json& data = obj["MonsterSpawnData"];
    maxMonsterAmount = data["MaxAmount"].get<unsigned int>();
    timeBetweenMonsterSpawns = data["TimeBetweenSpawns"].get<unsigned int>();
    monsterSpawnAmount = data["SpawnAmount"].get<unsigned int>();
}

unsigned int Config::ConfigFileReader::loadInitialMerchantGold() {
    return obj["InitialMerchantGold"].get<unsigned int>();
}

unsigned int Config::ConfigFileReader::loadPlayerSpeed() {
    return obj["PlayerSpeed"].get<unsigned int>();
}

double Config::ConfigFileReader::loadTimeForPlayerRecovery() {
    return obj["TimeForPlayerRecoveryInSeconds"].get<unsigned int>();
}

std::string Config::ConfigFileReader::loadPort() {
    return obj["Port"].get<std::string>();
}

std::string Config::ConfigFileReader::loadMapPath() {
    return obj["MapPath"].get<std::string>();
}

std::string Config::ConfigFileReader::loadSavePath() {
    return obj["SavePath"].get<std::string>();
}

std::string Config::ConfigFileReader::loadIndexPath() {
    return obj["IndexPath"].get<std::string>();
}
```

```
//
// Created by ivan on 10/6/20.
//

#ifndef ARGENTUM_CALCULATOR_H
#define ARGENTUM_CALCULATOR_H


class Calculator {
public:

    /*Calcula la cantidad maxima de vida del player*/
    static int calculateMaxLife(unsigned int constitution, unsigned int classLifeMultiplier,
                                unsigned int raceLifeMultiplier, unsigned int level);

    /*Calcula la cantidad maxima de mana del player*/
    static unsigned int calculateMaxMana(unsigned int intelligence, unsigned int classManaMultiplier,
                                         unsigned int raceManaMultiplier, unsigned int level);

    /*Calcula el drop de oro dropeado por un monstruo al matarlo*/
    static unsigned int calculateGoldDrop(unsigned int maxLife);

    /*Calcula cantidad de oro en mano segura*/
    static unsigned int calculateMaxSafeGold(unsigned int level);

    /*Calcula el XP necesario para subir de nivel*/
    static unsigned int calculateNextLevelXP(unsigned int level);

    /*Calcula el XP ganado en el ataque*/
    static unsigned int calculateAttackXP(int dmg, unsigned int
                                                          myLevel, unsigned int otherLevel);

    /*Calcula la cantidad de XP ganada por el player al matar a otro NPC (player o monster*/
    static unsigned int calculateKillXP(unsigned int myLevel, unsigned int otherLevel,
                                        unsigned int othermaxLife);

    /*Calcula el danio inicial del player, este danio podra ser esquivado o disminuido en base a la defensa del rival*/
    static int calculateDamage(unsigned int strength, int weaponDamage);

    /*Retorna si el npc o player pudo o no esquivar el ataque*/
    static bool canDodge(unsigned int agility);

    /*Retorna un int random entre minRage y maxRange*/
    static int getRandomInt(int minRange, int maxRange);

    /*Retorna true si el ataque fue critical, flase en caso contrario*/
    static bool isCritical();

    /*Retorna la cantidad de vida recupera en base al recovery rate del player y el tiempo pasado*/
    static int lifeRecovered(unsigned int recoveryRate, double timeElpased);

    /*Retorna la cantidad de mana recuperada sin estar en estado de meditacion en base al tiempo pasado*/
    static unsigned int manaRecoveredNoMeditation(unsigned int recoveryRate, double timeElpased);

    /*Retorna la cantidad de mana recuperado cuando se esta en estado de meditacion en base al tiempo pasado*/
```

```
    static unsigned int manaRecoveredWithMeditation(unsigned int meditationRate,
                                                    unsigned int intelligence, double timeElpased);

private:
    static float _getRandomFloat(float minRange, float maxRange);
};


#endif //ARGENTUM_CALCULATOR_H
```

```cpp
//
// Created by ivan on 10/6/20.
//

#include "Calculator.h"
#include "Configuration.h"
#include <random>

using namespace Config;

bool Calculator::isCritical() {
    Configuration& config = Configuration::getInstance();
    std::random_device seed;
    std::default_random_engine generator(seed());
    std::bernoulli_distribution dist(config.configCriticalAttackChance());
    return dist(generator);
}

int Calculator::calculateMaxLife(unsigned int constitution, unsigned int classLifeMultiplier,
                                unsigned int raceLifeMultiplier, unsigned int level) {
    return static_cast<int>(constitution * classLifeMultiplier * raceLifeMultiplier * level);
}

unsigned int Calculator::calculateMaxMana(unsigned int intelligence, unsigned int classManaMultiplier,
                                            unsigned int raceManaMultiplier, unsigned int level) {
    return intelligence * classManaMultiplier * raceManaMultiplier * level;
}

unsigned int Calculator::calculateGoldDrop(unsigned int maxLife) {

    float minRange = Configuration::getInstance().configGoldModifiers()
            .goldDropFactorMin;
    float maxRange = Configuration::getInstance().configGoldModifiers()
            .goldDropFactorMax;
    float randNum = _getRandomFloat(minRange, maxRange);

    return static_cast<unsigned int>(randNum * static_cast<float>(maxLife));
}

unsigned int Calculator::calculateMaxSafeGold(unsigned int level) {
    unsigned int multiplier = Configuration::getInstance()
            .configGoldModifiers().safeGoldFactor;
    float exponent = Configuration::getInstance()
            .configGoldModifiers().safeGoldLevelModifier;
    return (multiplier * static_cast<unsigned int>(pow(level, exponent)));
}

unsigned int Calculator::calculateNextLevelXP(unsigned int level) {
    unsigned int multiplier = Configuration::getInstance().configXPModifiers
            ().nextLevelFactor;
    float exponent = Configuration::getInstance().configXPModifiers
            ().nextLevelModifier;

    return (multiplier * static_cast<unsigned int>(pow(level, exponent)));
}

unsigned int Calculator::calculateAttackXP(int dmg, unsigned int
                                    myLevel, unsigned int otherLevel) {
    if (dmg < 0) {
        return 0;
    }
```

```cpp
    unsigned int modifier = Configuration::getInstance().configXPModifiers()
            .attackXPModifier;

    int multiplier = static_cast<int>(otherLevel − myLevel + modifier);
    return (dmg * std::max(multiplier, 0));
}

unsigned int Calculator::calculateKillXP(unsigned int myLevel,
                                         unsigned int otherLevel,
                                         unsigned int othermaxLife) {
    unsigned int modifier = Configuration::getInstance().configXPModifiers()
            .killXPModifier;
    int multiplier = static_cast<int>((otherLevel − myLevel + modifier));
    float minRange = Configuration::getInstance().configXPModifiers()
            .killXPMinRange;
    float maxRange = Configuration::getInstance().configXPModifiers()
            .killXPMaxRange;
    float random = _getRandomFloat(minRange, maxRange);

    return static_cast<unsigned int>((random * static_cast<float>(othermaxLife) * std::max(multiplier, 0)));
}


int Calculator::calculateDamage(unsigned int strength, int weaponDamage) {
    return static_cast<int>(strength) * weaponDamage;
}

bool Calculator::canDodge(unsigned int agility) {
    float random = _getRandomFloat(0, 1);
    Configuration& config = Configuration::getInstance();
    return (pow(random, agility) < config.configDodgeChance());
}

float Calculator::_getRandomFloat(float minRange, float maxRange) {
    std::random_device seed;
    std::default_random_engine generator(seed());
    std::uniform_real_distribution<float> dist(minRange, maxRange);
    return dist(generator);
}

int Calculator::getRandomInt(int minRange, int maxRange) {
    std::random_device seed;
    std::default_random_engine generator(seed());
    std::uniform_int_distribution<int> dist(minRange, maxRange);
    return dist(generator);
}

int Calculator::lifeRecovered(unsigned int recoveryRate, double timeElpased) {
    return static_cast<int>(static_cast<double>(recoveryRate) * timeElpased) * 10;
}

unsigned int Calculator::manaRecoveredNoMeditation(unsigned int recoveryRate, double timeElpased) {
    return static_cast<int>(static_cast<double>(recoveryRate) * timeElpased);
}

unsigned int Calculator::manaRecoveredWithMeditation(unsigned int meditationRate,
                                        unsigned int intelligence, double timeElpased) {

    return static_cast<int>(static_cast<double>(meditationRate * intelligence) * timeElpased);
}
```

```cpp
//
// Created by marcos on 6/6/20.
//

#ifndef ARGENTUM_TPEXCEPTION_H
#define ARGENTUM_TPEXCEPTION_H

#include <exception>
#define BUF_LEN 256

class TPException : public std::exception {
private:
    char errorMessage[BUF_LEN]{};
public:
    explicit TPException(const char *fmt, ...);
    const char * what() const noexcept override;
};


#endif //ARGENTUM_TPEXCEPTION_H
```

```cpp
//
// Created by marcos on 6/6/20.
//

#include <cstdarg>
#include <cstdio>
#include "TPException.h"

TPException::TPException(const char *fmt, ...) {
    va_list args;
    va_start(args, fmt);
    vsnprintf(errorMessage, BUF_LEN, fmt, args);
    va_end(args);
    errorMessage[BUF_LEN - 1] = 0;
}

const char *TPException::what() const noexcept {
    return errorMessage;
}

```

```cpp
//
// Created by marcos on 13/7/20.
//

#ifndef ARGENTUM_TIMER_H
#define ARGENTUM_TIMER_H

#include <chrono>

class Timer {
private:
    std::chrono::high_resolution_clock::time_point time1{};
    std::chrono::high_resolution_clock::time_point time2{};

public:
    //Starts the timer
    void start();

    //Returns time passed since start in milliseconds
    double getTime();
};


#endif //ARGENTUM_TIMER_H
```

```cpp
//
// Created by marcos on 13/7/20.
//

#include "Timer.h"
using namespace std::chrono;

void Timer::start() {
    time1 = high_resolution_clock::now();
}

double Timer::getTime() {
    time2 = high_resolution_clock::now();
    duration<double, std::milli> timeStep = time2 - time1;
    return timeStep.count();
}
```

```
#ifndef TP3TALLER_THREAD_H
#define TP3TALLER_THREAD_H

/*Esta clase es abstracta, las clases que heredan de ella pueden correr el
 * metodo run en un thread nuevo*/

#include <thread>

class Thread {
private:
    std::thread thread;

public:
    /*Ejecuta en un nuevo thread al metodo run*/
    void operator()();

    /*Joinea el thread*/
    virtual void join();

    /*Libera el thread violentamente*/
    void detach();

    virtual ~Thread() = default;

protected:
    /*Metodo abstracto, las clases hijas deben implementarlo*/
    virtual void run() = 0;
};


#endif //TP3TALLER_THREAD_H
```

```
#include "Thread.h"

void Thread::operator()() {
    thread = std::thread(&Thread::run, this);
}

void Thread::join() {
    thread.join();
}

void Thread::detach() {
    thread.detach();
}
```

```cpp
#ifndef TP3TALLER_SOCKET_H
#define TP3TALLER_SOCKET_H

#include <cstdio>
#include <string>
#include "TPException.h"

class Socket {
private:
    int fd; /*File Descriptor*/

public:
    Socket();
    Socket(const Socket&) = delete; /*Borro los constructores por copia*/
    Socket operator=(const Socket&) = delete;
    Socket& operator=(Socket&&) noexcept;
    Socket(Socket&& srcSocket) noexcept;

    /*Conecta el cliente al servidor*/
    void connect(std::string& host, std::string& port);

    /*Acepta una nueva conexion, retornando el socket generado*/
    Socket accept() const;

    /*Bindea a un socket*/
    void bind(const std::string& port);

    /*Setea la cantidad maxima de clientes que se tendran en espera*/
    void maxListen(int max) const;

    /*Envia el mensaje*/
    void send(const char* message, size_t length) const;

    /*Recibe el mensaje*/
    void receive(char* message, size_t length) const;

    /*Cierra el socket*/
    void close();

    ~Socket();

private:
    explicit Socket(int fd) : fd(fd) {}
    static struct addrinfo* _getAddresses(std::string* host, const std::string*
port);
};


#endif //TP3TALLER_SOCKET_H
```

```cpp
#include "Socket.h"
#include <netdb.h>
#include <unistd.h>
#include "TPException.h"
#include <cstring>

#define CONNECT_ERROR_MSG "Could not run. "
#define BIND_ERROR_MSG "Could not bind. "
#define ACCEPT_ERROR_MSG "Error in accept: "
#define SEND_ERROR_MSG "Error in send: "
#define RECV_ERROR_MSG "Error in recv: "
#define GETADDRINFO_ERROR_MSG "Error in getaddrinfo: %s"

struct addrinfo* Socket::_getAddresses(std::string* host, const std::string* por
t) {
    struct addrinfo hints{}, *result;
    int s; /*Para verificar errores*/
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    if (host) {
        hints.ai_flags = 0; /*cliente*/
        s = getaddrinfo(host->c_str(), port->c_str(), &hints, &result);
    } else {
        hints.ai_flags = AI_PASSIVE; /*server*/
        s = getaddrinfo(nullptr, port->c_str(), &hints, &result);
    }
    if (s != 0) throw TPException(GETADDRINFO_ERROR_MSG, gai_strerror(s));
    return result;
}

void Socket::connect(std::string& host, std::string& port) {
    struct addrinfo* addresses = _getAddresses(&host, &port);
    struct addrinfo* rp;
    for (rp = addresses; rp != nullptr; rp = rp->ai_next) {
        fd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
        if (fd  == -1)
            continue;

        if (::connect(fd , rp->ai_addr, rp->ai_addrlen) != -1)
            break; /*Logre conectarme*/

        ::close(fd);
    }
    freeaddrinfo(addresses);
    if (rp == nullptr) {
        throw TPException(CONNECT_ERROR_MSG);
    }
}

void Socket::bind(const std::string& port) {
    struct addrinfo* addresses = _getAddresses(nullptr, &port);
    struct addrinfo* rp;
    int val = 1;
    for (rp = addresses; rp != nullptr; rp = rp->ai_next) {
        fd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
        if (fd == -1)
            continue;

        setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &val, sizeof(val));

        if (::bind(fd, rp->ai_addr, rp->ai_addrlen) == 0)
            break; /*Logre bindear*/

        ::close(fd);
    }
    freeaddrinfo(addresses);
    if (rp ≡ nullptr) {
        throw TPException(BIND_ERROR_MSG);
    }
}

Socket Socket::accept() const {
    int peerFd = ::accept(fd, nullptr, nullptr);
    if (peerFd ≡ −1) {
        throw TPException(ACCEPT_ERROR_MSG);
    }
    return Socket(peerFd);
}

void Socket::send(const char* message, size_t length) const {
    size_t bytesSent = 0;
    int s = 0;

    while (bytesSent < length) {
        s = ::send(fd, message + bytesSent, length − bytesSent, MSG_NOSIGNAL);
        if (s < 1) throw TPException(SEND_ERROR_MSG);
        bytesSent += s;
    }
}

void Socket::receive(char* message, size_t length) const {
    size_t bytesReceived = 0;
    int s = 0;

    while (bytesReceived < length) {
        s = recv(fd, message + bytesReceived, length − bytesReceived, 0);
        if (s < 1) throw TPException(RECV_ERROR_MSG);
        bytesReceived += s;
    }
}

void Socket::maxListen(int max) const {
    listen(fd, max);
}

Socket::~Socket() {
    close();
}

Socket::Socket(Socket∧ srcSocket) noexcept {
    fd = srcSocket.fd;
    srcSocket.fd = −1;
}

Socket& Socket::operator=(Socket∧ srcSocket) noexcept {
    fd = srcSocket.fd;
    srcSocket.fd = −1;
    return *this;
}

void Socket::close() {
    if (fd ≠ −1) {
        shutdown(fd, SHUT_RDWR);
        ::close(fd);
        fd = −1;
    }
}

Socket::Socket() {
    fd = −1;
}
```

```
//
// Created by marcos on 17/7/20.
//

#ifndef ARGENTUM_SHAREDCONSTANTS_H
#define ARGENTUM_SHAREDCONSTANTS_H

const unsigned int TILE_DISTANCE_IN_METERS = 2000;

#endif //ARGENTUM_SHAREDCONSTANTS_H
```

```
//
// Created by marcos on 20/6/20.
//

#ifndef ARGENTUM_GAMEENUMS_H
#define ARGENTUM_GAMEENUMS_H

#include <cinttypes>

namespace GameType {

    enum ConnectionResponse : int32_t {
        ACCEPTED, INEXISTENT_PLAYER, UNAVAILABLE_PLAYER, UNKOWN_SERVER_ERROR
    };

    enum PlayerEvent : int32_t {
        PLAYER_START_MOVING, PLAYER_STOP_MOVING, CREATE_PLAYER, LOAD_PLAYER, PLAYER_ATTACK, PLAYER_USE_ITEM, PLAYER_UNEQUIP,
        PLAYER_PICK_UP, PLAYER_DROP, PLAYER_LIST, PLAYER_BUY, PLAYER_SELL, PLAYER_WITHDRAW,
        PLAYER_DEPOSIT, PLAYER_MEDITATE, PLAYER_RESURRECT, PLAYER_HEAL, PLAYER_SEND_MSG,
        PLAYER_REQUEST_INVENTORY_NAMES
    };

    enum EventID: int32_t {
        MOVED, ATTACK, UNEQUIP, EQUIPPED, CREATE_ENTITY, CREATE_ITEM, REMOVE_ENTITY,
        DESTROY_ITEM, PLAYER_DEATH, RESURRECTED, TELEPORTED, PLAYER_LEVEL_UP
    };

    enum Race : int32_t {
        HUMAN, ELF, DWARF, GNOME
    };

    enum Class : int32_t {
        WIZARD, CLERIC, PALADIN, WARRIOR
    };

    enum Entity: int32_t {
        SKELETON, ZOMBIE, SPIDER, GOBLIN, BANKER, GUARD, TRADER, PRIEST, PLAYER, NOTHING
    };

    enum ItemType: int32_t {
        ITEM_TYPE_GOLD, ITEM_TYPE_WEAPON, ITEM_TYPE_CLOTHING, ITEM_TYPE_POTION, ITEM_TYPE_NONE
    };

    enum Weapon: int32_t {
        LONGSWORD, AXE, WARHAMMER, ASH_ROD, ELVEN_FLUTE, LINKED_STAFF,
        SIMPLE_BOW, COMPOSITE_BOW, GNARLED_STAFF, FIST, ZOMBIE_ATTACK, SPIDER_ATTACK,
        GOBLIN_ATTACK, SKELETON_ATTACK
    };

    enum Clothing: int32_t {
        COMMON_CLOTHING, LEATHER_ARMOR, PLATE_ARMOR, KING_ARMOR, BLUE_TUNIC, HOOD,
        IRON_HELMET, TURTLE_SHIELD, IRON_SHIELD, MAGIC_HAT, NO_HELMET,
        NO_SHIELD
    };

    enum Potion: int32_t {
        HEALTH_POTION, MANA_POTION
    };
```

```

    enum FloorType: int32_t {
        GRASS0, GRASS1, GRASS2, GRASS3, SAND, WATER0, WATER1, WATER2, WATER3,
        PRETTY_ROAD0, PRETTY_ROAD1, PRETTY_ROAD2, PRETTY_ROAD3, PRETTY_GRASS0,
        PRETTY_GRASS1, PRETTY_GRASS2,PRETTY_GRASS3, DEAD_GRASS0, DEAD_GRASS1,
        DEAD_GRASS2, DEAD_GRASS3, DARK_WATER0, DARK_WATER1, DARK_WATER2, DARK_WATER3,
    };

    enum Structure: int32_t {
        BONE_GUY, BROKEN_RIP_STONE, BUSH, DEAD_BUSH, DEAD_GUY, DEAD_TREE,
        FAT_TREE, HANGED_GUY, HOUSE1, HOUSE2, HOUSE3, LONG_TREE, PALM_TREE,
        RIP_STONE, TREE, VERY_DEAD_GUY, SUNKEN_COLUMN, SUNKEN_SHIP, NO_STRUCTURE
    };

    enum Direction : int32_t {
        DIRECTION_UP, DIRECTION_DOWN, DIRECTION_LEFT, DIRECTION_RIGHT, DIRECTION_STILL
    };

    enum EquipmentPlace: int32_t {
        EQUIPMENT_PLACE_NONE, EQUIPMENT_PLACE_HEAD, EQUIPMENT_PLACE_CHEST, EQUIPMENT_PLACE_WEAPON,
        EQUIPMENT_PLACE_SHIELD
    };
}

#endif //ARGENTUM_GAMEENUMS_H
```

```
//
// Created by marcos on 7/9/20.
//

#ifndef ARGENTUM_UPDATETELEPORTENTITY_H
#define ARGENTUM_UPDATETELEPORTENTITY_H

#include "UpdateEvent.h"
#include <string>
#include "../Map/Coordinate.h"

class UpdateTeleportEntity : public UpdateEvent {
private:
    std::string nickname;
    Coordinate newPosition;

public:
    UpdateTeleportEntity(std::string∧ _nickname, Coordinate _position) :
                        nickname(_nickname), newPosition(_position) {}

    void operator()(GameGUI& game) override;
};


#endif //ARGENTUM_UPDATETELEPORTENTITY_H
```

```
//
// Created by marcos on 7/9/20.
//

#include "UpdateTeleportEntity.h"
#include "../Client/GameGUI.h"

void UpdateTeleportEntity::operator()(GameGUI &game) {
    bool isMyPlayer = (game.getPlayerInfo().getNickname() ≡ nickname);
    game.getMap().teleportEntity(nickname, newPosition, isMyPlayer);
}
```

```
//
// Created by marcos on 7/3/20.
//

#ifndef ARGENTUM_UPDATEREMOVEENTITY_H
#define ARGENTUM_UPDATEREMOVEENTITY_H

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

class UpdateRemoveEntity : public UpdateEvent {
private:
    std::string nickname;

public:
    explicit UpdateRemoveEntity(std::string∧ _nickname) : nickname(_nickname) {
}
    void operator()(GameGUI& game) override;
};


#endif //ARGENTUM_UPDATEREMOVEENTITY_H
```

```
//
// Created by marcos on 7/3/20.
//

#include "UpdateRemoveEntity.h"
#include "../Client/GameGUI.h"

void UpdateRemoveEntity::operator()(GameGUI &game) {
    game.getMap().removeEntity(nickname);
}
```

```
//
// Created by ivan on 9/7/20.
//

#ifndef ARGENTUM_UPDATEPLAYERRESURRECT_H
#define ARGENTUM_UPDATEPLAYERRESURRECT_H

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

class UpdatePlayerResurrect : public UpdateEvent {
private:
    std::string nickname;

public:
    explicit UpdatePlayerResurrect(std::string^ _nickname) : nickname(std::move
(_nickname)) {}
    void operator()(GameGUI& game);
};


#endif //ARGENTUM_UPDATEPLAYERRESURRECT_H
```

```
//
// Created by ivan on 9/7/20.
//

#include "UpdatePlayerResurrect.h"
#include "../Client/GameGUI.h"

void UpdatePlayerResurrect::operator()(GameGUI &game) {
    game.getMap().revivePlayer(nickname);
}
```

```cpp
//
// Created by marcos on 7/5/20.
//

#ifndef ARGENTUM_UPDATEPLAYERDEATH_H
#define ARGENTUM_UPDATEPLAYERDEATH_H

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

class UpdatePlayerDeath : public UpdateEvent {
private:
    std::string nickname;

public:
    explicit UpdatePlayerDeath(std::string^ _nickname) : nickname(std::move(_nickname)) {}
    void operator()(GameGUI& game);
};


#endif //ARGENTUM_UPDATEPLAYERDEATH_H
```

```cpp
//
// Created by marcos on 7/5/20.
//

#include "UpdatePlayerDeath.h"
#include "../Client/GameGUI.h"

void UpdatePlayerDeath::operator()(GameGUI &game) {
    game.getMap().killPlayer(nickname);
}
```

```cpp
//
// Created by marcos on 6/29/20.
//

#ifndef ARGENTUM_UPDATEMOVE_H
#define ARGENTUM_UPDATEMOVE_H

#include "UpdateEvent.h"
#include <string>
#include "../../libs/GameEnums.h"

class UpdateMove : public UpdateEvent {
private:
    std::string nickname;
    GameType::Direction direction;
    unsigned int distanceTravelled;
    bool reachedDestination;

public:
    UpdateMove(std::string∧ _nickname, GameType::Direction _direction,
            unsigned int _distanceTravelled, bool _reachedDestination) :
            nickname(std::move(_nickname)), direction(_direction), distanceTrave
lled(_distanceTravelled),
            reachedDestination(_reachedDestination){}

    void operator()(GameGUI& game) override;
};


#endif //ARGENTUM_UPDATEMOVE_H
```

```cpp
//
// Created by marcos on 6/29/20.
//

#include "UpdateMove.h"
#include "../Map/Map.h"

#include "../Client/GameGUI.h"

void UpdateMove::operator()(GameGUI& game) {
    game.getMap().moveEntity(nickname, direction, distanceTravelled,
                                                            reachedDestination);
}
```

jul 21, 20 15:47 **UpdateLevelUp.h** Page 1/1

```
//
// Created by marcos on 20/7/20.
//

#ifndef ARGENTUM_UPDATELEVELUP_H
#define ARGENTUM_UPDATELEVELUP_H

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

class UpdateLevelUp : public UpdateEvent {
private:
    std::string playerNickname;
    int level;

public:
    explicit UpdateLevelUp(std::string^ _playerNickname, int _level) :
            playerNickname(std::move(_playerNickname)), level(_level) {}
    void operator()(GameGUI& game) override;
};


#endif //ARGENTUM_UPDATELEVELUP_H
```

jul 21, 20 15:47 **UpdateLevelUp.cpp** Page 1/1

```
//
// Created by marcos on 20/7/20.
//

#include "UpdateLevelUp.h"
#include "../Client/GameGUI.h"

void UpdateLevelUp::operator()(GameGUI &game) {
    game.getMap().updatePlayerLevel(playerNickname, level);
}
```

```
//
// Created by marcos on 7/2/20.
//

#ifndef ARGENTUM_UPDATEGUI_H
#define ARGENTUM_UPDATEGUI_H

#include "UpdateEvent.h"
#include "../Client/EntityData.h"

class UpdateGUI : public UpdateEvent {
private:
    PlayerData data;

public:
    explicit UpdateGUI(PlayerData^ _data) : data(std::move(_data)) {}
    void operator()(GameGUI& game) override;
};


#endif //ARGENTUM_UPDATEGUI_H
```

```
//
// Created by marcos on 7/2/20.
//

#include "UpdateGUI.h"
#include "../Client/GameGUI.h"

void UpdateGUI::operator()(GameGUI &game) {
    for (const auto & item : data.equippedItems) {
        game.getPlayerInventory().addEquipableItem(std::get<0>(item),
                                                   std::get<1>(item));
    }
    for (const auto & item : data.inventoryItems) {
        game.getPlayerInventory().addInventoryItem(std::get<0>(item),
                                                   std::get<1>(item));
    }
    game.getPlayerInfo().update(data.generalInfo);
    game.getMinichat().receiveText(data.minichatText);
}
```

```
//
// Created by marcos on 6/29/20.
//

#ifndef ARGENTUM_UPDATEEVENT_H
#define ARGENTUM_UPDATEEVENT_H

/*Interfaz, los eventos que updatean el juego deben implmenetar cada caso*/

class GameGUI;

class UpdateEvent {
public:
    virtual void operator()(GameGUI& game) = 0;
    virtual ~UpdateEvent() = default;
};


#endif //ARGENTUM_UPDATEEVENT_H
```

```
//
// Created by marcos on 7/5/20.
//

#ifndef ARGENTUM_UPDATEEQUIP_H
#define ARGENTUM_UPDATEEQUIP_H

#include "UpdateEvent.h"
#include <string>
#include "../../libs/GameEnums.h"
#include "../Texture/TextureID.h"

const int UNEQUIP = -1; /*Uso esta constante para decirle que es un unequip*/

class UpdateEquip : public UpdateEvent {
private:
    std::string nickname;
    GameType::EquipmentPlace place;
    TextureID equipment;

public:
    UpdateEquip(std::string^ _nickname, GameType::EquipmentPlace _place,
                int32_t _equipment);

    void operator()(GameGUI& game) override;
};


#endif //ARGENTUM_UPDATEEQUIP_H
```

```
//
// Created by marcos on 7/5/20.
//

#include "UpdateEquip.h"
#include "../Client/ProtocolEnumTranslator.h"
#include "../Client/GameGUI.h"

UpdateEquip::UpdateEquip(std::string ∧_nickname,
                         GameType::EquipmentPlace _place, int32_t _equipment) {

    ProtocolEnumTranslator translator;
    nickname = std::move(_nickname);
    place = _place;
    if (_equipment ≡ UNEQUIP) {
        equipment = Nothing;
    } else {
        switch (place) {
            case GameType::EQUIPMENT_PLACE_WEAPON:
                equipment = translator.getWeaponTexture(static_cast<GameType::We
apon>(_equipment));
                break;

            default:
                equipment = translator.getClothingTexture(static_cast<GameType::
Clothing>(_equipment));
        }
    }
}

void UpdateEquip::operator()(GameGUI &game) {
    game.getMap().equipOnPlayer(nickname, place, equipment);
}
```

```
//
// Created by marcos on 9/7/20.
//

#ifndef ARGENTUM_UPDATEDESTROYITEM_H
#define ARGENTUM_UPDATEDESTROYITEM_H

#include "UpdateEvent.h"
#include "../Map/Coordinate.h"

class UpdateDestroyItem : public UpdateEvent {
private:
    Coordinate position;

public:
    explicit UpdateDestroyItem(Coordinate _position) : position(_position) {}
    void operator()(GameGUI& game) override;
};


#endif //ARGENTUM_UPDATEDESTROYITEM_H
```

```cpp
//
// Created by marcos on 9/7/20.
//

#include "UpdateDestroyItem.h"
#include "../Client/GameGUI.h"

void UpdateDestroyItem::operator()(GameGUI &game) {
    game.getMap().destroyItem(position);
}
```

```cpp
//
// Created by ivan on 1/7/20.
//

#ifndef ARGENTUM_UPDATECREATEPLAYER_H
#define ARGENTUM_UPDATECREATEPLAYER_H

#include "UpdateEvent.h"
#include <string>
#include "../../libs/GameEnums.h"
#include "../Client/ClientProtocol.h"

class UpdateCreatePlayer : public UpdateEvent{
private:
    MapPlayerData data;
public:
    explicit UpdateCreatePlayer(MapPlayerData& _data) : data(_data) {}
    void operator()(GameGUI& game) override;
};


#endif //ARGENTUM_UPDATECREATEPLAYER_H
```

```cpp
//
// Created by ivan on 1/7/20.
//

#include "UpdateCreatePlayer.h"
#include "../Client/GameGUI.h"

void UpdateCreatePlayer::operator()(GameGUI& game) {
    game.addPlayer(data);
}
```

```cpp
//
// Created by marcos on 7/2/20.
//

#ifndef ARGENTUM_UPDATECREATENPC_H
#define ARGENTUM_UPDATECREATENPC_H

#include "UpdateEvent.h"
#include "../Client/ClientProtocol.h"

class UpdateCreateNPC : public UpdateEvent {
private:
    EntityData data;
public:
    explicit UpdateCreateNPC(EntityData& _data) : data(_data) {}
    void operator()(GameGUI& game) override;
};

#endif //ARGENTUM_UPDATECREATENPC_H
```

```
//
// Created by marcos on 7/2/20.
//

#include "UpdateCreateNPC.h"
#include "../Client/GameGUI.h"

void UpdateCreateNPC::operator()(GameGUI& game) {
    game.addNPC(data);
}
```

```
//
// Created by marcos on 7/5/20.
//

#ifndef ARGENTUM_UPDATECREATEITEM_H
#define ARGENTUM_UPDATECREATEITEM_H

#include "UpdateEvent.h"
#include "../../libs/GameEnums.h"
#include "../Map/Coordinate.h"
#include "../Texture/TextureID.h"

class UpdateCreateItem : public UpdateEvent {
private:
    TextureID item;
    Coordinate position{};

public:
    UpdateCreateItem(GameType::ItemType _type, int32_t _item, Coordinate _positi
on);
    void operator()(GameGUI& game) override;
};


#endif //ARGENTUM_UPDATECREATEITEM_H
```

```cpp
//
// Created by marcos on 7/5/20.
//

#include "UpdateCreateItem.h"
#include "../Client/GameGUI.h"
#include "../Client/ProtocolEnumTranslator.h"

UpdateCreateItem::UpdateCreateItem(GameType::ItemType _type, int32_t _item,
                                    Coordinate _position) {
    ProtocolEnumTranslator translator;
    position = _position;
    try {
        switch (_type) {
            case GameType::ITEM_TYPE_GOLD:
                item = Gold;
                break;
            case GameType::ITEM_TYPE_WEAPON:
                item = translator.getWeaponDropTexture(static_cast<GameType::Weapon>(_item));
                break;
            case GameType::ITEM_TYPE_CLOTHING:
                item = translator.getClothingDropTexture(static_cast<GameType::Clothing>(_item));
                break;
            case GameType::ITEM_TYPE_POTION:
                item = translator.getPotionTexture(static_cast<GameType::Potion>(_item));
                break;
            case GameType::ITEM_TYPE_NONE:
                throw TPException("Tried to create a null item");
        }
    } catch (std::exception& e) {
        std::cerr << e.what() << "Tried to create an unknown item" << std::endl;
    }
}

void UpdateCreateItem::operator()(GameGUI &game) {
    game.getMap().createItem(position, item);
}
```

```cpp
//
// Created by marcos on 7/7/20.
//

#ifndef ARGENTUM_UPDATEATTACK_H
#define ARGENTUM_UPDATEATTACK_H

#include "UpdateEvent.h"
#include "../Map/Coordinate.h"
#include "../../libs/GameEnums.h"
#include <string>

class UpdateAttack : public UpdateEvent {
private:
    Coordinate position{};
    GameType::Weapon weapon;
    GameType::Direction attackDir;
    std::string nickname;

public:
    UpdateAttack(std::string& _nickname, Coordinate _position, int32_t _weapon,
                                        GameType::Direction _attackDir);
    void operator()(GameGUI& game) override;

private:
    bool _shouldPlaySound(); //Para no spammear siempre sonidos de ataque de los monstruos
};


#endif //ARGENTUM_UPDATEATTACK_H
```

```cpp
//
// Created by marcos on 7/7/20.
//

#include "UpdateAttack.h"
#include "../Client/GameGUI.h"
#include <random>

#define ATTACK_HEARING_DISTANCE 6

const float MONSTER_SOUND_PROBABILITY = 0.15;

void UpdateAttack::operator()(GameGUI &game) {
    switch (weapon) {
        case GameType::GNARLED_STAFF:
            game.getMap().addSpell(position, MagicMissile);
            game.getMap().verifyQueueSound(position, Explotion3Sound, ATTACK_HEARING_DISTANCE);
            break;
        case GameType::ELVEN_FLUTE:
            game.getMap().addSpell(position, Heal);
            game.getMap().verifyQueueSound(position, HealingSound, ATTACK_HEARING_DISTANCE);
            break;
        case GameType::LINKED_STAFF:
            game.getMap().addSpell(position, Explosion);
            game.getMap().verifyQueueSound(position, Explotion1Sound, ATTACK_HEARING_DISTANCE);
            break;
        case GameType::ASH_ROD:
            game.getMap().addSpell(position, MagicArrow);
            game.getMap().verifyQueueSound(position, Explotion2Sound, ATTACK_HEARING_DISTANCE);
            break;
        case GameType::LONGSWORD:
            game.getMap().verifyQueueSound(position, SwordAttackSound, ATTACK_HEARING_DISTANCE);
            break;
        case GameType::AXE:
            game.getMap().verifyQueueSound(position, Attack2Sound, ATTACK_HEARING_DISTANCE);
            break;
        case GameType::WARHAMMER:
            game.getMap().verifyQueueSound(position, HeavyAttackSound, ATTACK_HEARING_DISTANCE);
            break;
        case GameType::COMPOSITE_BOW:
            game.getMap().addArrow(nickname, position, CompositeArrow);
            game.getMap().verifyQueueSound(position, ArrowSound, ATTACK_HEARING_DISTANCE);
            break;
        case GameType::SIMPLE_BOW:
            game.getMap().addArrow(nickname, position, SimpleArrow);
            game.getMap().verifyQueueSound(position, ArrowSound, ATTACK_HEARING_DISTANCE);
            break;
        case GameType::ZOMBIE_ATTACK:
            if (_shouldPlaySound())
                game.getMap().verifyQueueSound(position, ZombieSound, ATTACK_HEARING_DISTANCE);
            break;
        case GameType::SPIDER_ATTACK:
            if (_shouldPlaySound())
                game.getMap().verifyQueueSound(position, SpiderSound, ATTACK_HEARING_DISTANCE);
            break;
```

```cpp
        case GameType::SKELETON_ATTACK:
            if (_shouldPlaySound())
                game.getMap().verifyQueueSound(position, SkeletonSound, ATTACK_HEARING_DISTANCE);
            break;
        case GameType::GOBLIN_ATTACK:
            if (_shouldPlaySound())
                game.getMap().verifyQueueSound(position, GoblinSound, ATTACK_HEARING_DISTANCE);
            break;
        default:
            break;
    }
    game.getMap().changeEntityLookDirection(nickname, attackDir);
}

UpdateAttack::UpdateAttack(std::string& _nickname, Coordinate _position,
                        int32_t _weapon, GameType::Direction _attackDir) {
    nickname = std::move(_nickname);
    position = _position;
    weapon = static_cast<GameType::Weapon>(_weapon);
    attackDir = _attackDir;
}

bool UpdateAttack::_shouldPlaySound() {
    std::random_device seed;
    std::default_random_engine generator(seed());
    std::bernoulli_distribution dist(MONSTER_SOUND_PROBABILITY);
    return dist(generator);
}
```

```
//
// Created by marcos on 9/6/20.
//

#ifndef ARGENTUM_TEXTUREREPOSITORY_H
#define ARGENTUM_TEXTUREREPOSITORY_H

#include <unordered_map>
#include "Texture.h"
#include "TextureID.h"

class TextureRepository {
private:
    std::unordered_map<TextureID, Texture> textures;
    SDL_Renderer& renderer;

public:
    explicit TextureRepository(SDL_Renderer& renderer);
    Texture& getTexture(TextureID texture);

    /*Devuelve el mismo renderer que la window. No es ideal pero fue la mejor so
lucion
     * para poder crear el texto del nivel/nombre de las entidades ya que necesi
tamos
     * el renderer para text y sino tendriamos que pedirlo de screen, estando es
te
     * mucho mas arriba en jerarquia y no es posible sin cambiar todo el modelo*
/
    SDL_Renderer& getRenderer() const;

private:
    void _loadClothing();
    void _loadHeads();
    void _loadWeapons();
    void _loadTiles();
    void _loadStructures();
    void _loadNPCS();
    void _loadDrops();
    void _loadMiscellaneous();
    void _loadUI();

    void _setImage(TextureID TextureID, std::string∧ image,
                    int width, int height, int xOffset = 0, int yOffset = 0, int
 scale = 1
                            , ColorKey_t key = {0, 0, 0});
    void _setSpellImage(TextureID TextureID, std::string∧ spellImage,
                            int width, int height, int xOffset = 0, int yOffset
= 0);
    void _setNPCImage(TextureID TextureID, std::string∧ npcImage, int width, in
t height
                                                , int xOffset = 0, int yOffset =
 0);
    void _setBodyImage(TextureID texture, std::string∧ bodyImage);
    void _setShieldImage(TextureID TextureID, std::string∧ shieldImage);
    void _setWeaponImage(TextureID TextureID, std::string∧ weaponImage);
    void _setTileImage(TextureID TextureID, std::string∧ tileImage, bool indivi
dualTile);
    void _setHeadImage(TextureID TextureID, std::string∧ headImage);
    void _setHelmetImage(TextureID TextureID, std::string∧ helmetImage,
                         int xOffset = 0, int yOffset = 0);

    static void _addBodySprites(Texture& texture, int y, bool lateralSide);
    static void _addWeaponSprites(Texture& texture, int y, bool lateralSide);
    static void _addShieldSprites(Texture& texture, int y, bool lateralSide);
    static void _addNPCSprites(Texture& texture, int y, bool lateralSide, int wi
dth, int height);
```

```
    static void _addTileSprites(Texture& texture, int y, bool individualTile);
    static void _addSprites(Texture& texture, int width, int height);
    static void _addSpellSprites(Texture& texture, int y, int width, int height)
;
};


#endif //ARGENTUM_TEXTUREREPOSITORY_H
```

```
//
// Created by marcos on 9/6/20.
//

#include "TextureRepository.h"
#include "../Client/GameConstants.h"

#define PLAYER_GHOST_PATH "/var/Argentum/Assets/Images/Miscellaneous/PlayerGhost.png"
#define BLUE_TUNIC_PATH "/var/Argentum/Assets/Images/Clothing/BlueTunic.png"
#define BLUE_TUNIC_DROP_PATH "/var/Argentum/Assets/Images/Clothing/BlueTunicDrop.png"
#define COMMON_CLOTHING_PATH "/var/Argentum/Assets/Images/Clothing/CommonClothing.png"
#define COMMON_CLOTHING_DROP_PATH "/var/Argentum/Assets/Images/Clothing/CommonClothingDrop.png"
#define HOOD_PATH "/var/Argentum/Assets/Images/Clothing/Hood.png"
#define HOOD_DROP_PATH "/var/Argentum/Assets/Images/Clothing/HoodDrop.png"
#define IRON_HELMET_PATH "/var/Argentum/Assets/Images/Clothing/IronHelmet.png"
#define IRON_HELMET_DROP_PATH "/var/Argentum/Assets/Images/Clothing/IronHelmetDrop.png"
#define IRON_SHIELD_PATH "/var/Argentum/Assets/Images/Clothing/IronShield.png"
#define IRON_SHIELD_DROP_PATH "/var/Argentum/Assets/Images/Clothing/IronShieldDrop.png"
#define KING_ARMOR_PATH "/var/Argentum/Assets/Images/Clothing/KingArmor.png"
#define KING_ARMOR_DROP_PATH "/var/Argentum/Assets/Images/Clothing/KingArmorDrop.png"
#define LEATHER_ARMOR_PATH "/var/Argentum/Assets/Images/Clothing/LeatherArmor.png"
#define LEATHER_ARMOR_DROP_PATH "/var/Argentum/Assets/Images/Clothing/LeatherArmorDrop.png"
#define MAGIC_HAT_PATH "/var/Argentum/Assets/Images/Clothing/MagicHat.png"
#define MAGIC_HAT_DROP_PATH "/var/Argentum/Assets/Images/Clothing/MagicHatDrop.png"
#define PLATE_ARMOR_PATH "/var/Argentum/Assets/Images/Clothing/PlateArmor.png"
#define PLATE_ARMOR_DROP_PATH "/var/Argentum/Assets/Images/Clothing/PlateArmorDrop.png"
#define TURTLE_SHIELD_PATH "/var/Argentum/Assets/Images/Clothing/TurtleShield.png"
#define TURTLE_SHIELD_DROP_PATH "/var/Argentum/Assets/Images/Clothing/TurtleShieldDrop.png"
#define DWARF_HEAD_PATH "/var/Argentum/Assets/Images/Heads/DwarfHead.png"
#define ELF_HEAD_PATH "/var/Argentum/Assets/Images/Heads/ElfHead.png"
#define GNOME_HEAD_PATH "/var/Argentum/Assets/Images/Heads/GnomeHead.png"
#define HUMAN_HEAD_PATH "/var/Argentum/Assets/Images/Heads/HumanHead.png"
#define ASH_ROD_PATH "/var/Argentum/Assets/Images/Items/AshRod.png"
#define ASH_ROD_DROP_PATH "/var/Argentum/Assets/Images/Items/AshRodDrop.png"
#define AXE_PATH "/var/Argentum/Assets/Images/Items/Axe.png"
#define AXE_DROP_PATH "/var/Argentum/Assets/Images/Items/AxeDrop.png"
#define COMPOSITE_BOW_PATH "/var/Argentum/Assets/Images/Items/CompositeBow.png"
#define COMPOSITE_BOW_DROP_PATH "/var/Argentum/Assets/Images/Items/CompositeBowDrop.png"
#define ELVEN_FLUTE_DROP_PATH "/var/Argentum/Assets/Images/Items/ElvenFluteDrop.png"
#define LINKED_STAFF_PATH "/var/Argentum/Assets/Images/Items/LinkedStaff.png"
#define LINKED_STAFF_DROP_PATH "/var/Argentum/Assets/Images/Items/LinkedStaffDrop.png"
#define GNARLED_STAFF_PATH "/var/Argentum/Assets/Images/Items/GnarledStaff.png"
#define GNARLED_STAFF_DROP_PATH "/var/Argentum/Assets/Images/Items/GnarledStaffDrop.png"
#define LONG_SWORD_PATH "/var/Argentum/Assets/Images/Items/LongSword.png"
#define LONG_SWORD_DROP_PATH "/var/Argentum/Assets/Images/Items/LongSwordDrop.png"
#define SIMPLE_BOW_PATH "/var/Argentum/Assets/Images/Items/SimpleBow.png"
#define SIMPLE_BOW_DROP_PATH "/var/Argentum/Assets/Images/Items/SimpleBowDrop.png"
#define WAR_HAMMER_PATH "/var/Argentum/Assets/Images/Items/WarHammer.png"
#define WAR_HAMMER_DROP_PATH "/var/Argentum/Assets/Images/Items/WarHammerDrop.png"
#define HEALTH_POTION_PATH "/var/Argentum/Assets/Images/Items/HealthPotion.png"
#define MANA_POTION_PATH "/var/Argentum/Assets/Images/Items/ManaPotion.png"
#define GRASS_PATH "/var/Argentum/Assets/Images/Map/Grass.png"
#define PRETTY_GRASS_PATH "/var/Argentum/Assets/Images/Map/PrettyGrass.png"
#define PRETTY_ROAD_PATH "/var/Argentum/Assets/Images/Map/PrettyRoad.png"
#define DEAD_GRASS_PATH "/var/Argentum/Assets/Images/Map/DeadGrass.png"
#define SAND_PATH "/var/Argentum/Assets/Images/Map/Sand.png"
#define WATER_PATH "/var/Argentum/Assets/Images/Map/Water.png"
#define DARK_WATER_PATH "/var/Argentum/Assets/Images/Map/DarkWater.png"
#define SKELETON_PATH "/var/Argentum/Assets/Images/Monsters/Skeleton.png"
#define GOBLIN_PATH "/var/Argentum/Assets/Images/Monsters/Goblin.png"
#define ZOMBIE_PATH "/var/Argentum/Assets/Images/Monsters/Zombie.png"
#define SPIDER_PATH "/var/Argentum/Assets/Images/Monsters/Spider.png"
#define PRIEST_PATH "/var/Argentum/Assets/Images/Citizens/Priest.png"
#define TRADER_PATH "/var/Argentum/Assets/Images/Citizens/Trader.png"
#define BANKER_PATH "/var/Argentum/Assets/Images/Citizens/Banker.png"
```

```
#define GUARD_PATH "/var/Argentum/Assets/Images/Citizens/Guard.png"
#define TREE_PATH "/var/Argentum/Assets/Images/Map/Tree.png"
#define LONG_TREE_PATH "/var/Argentum/Assets/Images/Map/LongTree.png"
#define FAT_TREE_PATH "/var/Argentum/Assets/Images/Map/FatTree.png"
#define PALM_TREE_PATH "/var/Argentum/Assets/Images/Map/PalmTree.png"
#define DEAD_TREE_PATH "/var/Argentum/Assets/Images/Map/DeadTree.png"
#define BUSH_PATH "/var/Argentum/Assets/Images/Map/Bush.png"
#define DEAD_BUSH_PATH "/var/Argentum/Assets/Images/Map/DeadBush.png"
#define HOUSE1_PATH "/var/Argentum/Assets/Images/Map/House1.png"
#define HOUSE2_PATH "/var/Argentum/Assets/Images/Map/House2.png"
#define HOUSE3_PATH "/var/Argentum/Assets/Images/Map/House3.png"
#define SUNKEN_COLUMN_PATH "/var/Argentum/Assets/Images/Map/SunkenColumn.png"
#define SUNKEN_SHIP_PATH "/var/Argentum/Assets/Images/Map/SunkenShip.png"
#define BONE_GUY_PATH "/var/Argentum/Assets/Images/Map/BoneGuy.png"
#define BROKEN_RIP_STONE_PATH "/var/Argentum/Assets/Images/Map/BrokenRipStone.png"
#define DEAD_GUY_PATH "/var/Argentum/Assets/Images/Map/DeadGuy.png"
#define VERY_DEAD_GUY_PATH "/var/Argentum/Assets/Images/Map/VeryDeadGuy.png"
#define HANGED_GUY_PATH "/var/Argentum/Assets/Images/Map/HangedGuy.png"
#define RIP_STONE_PATH "/var/Argentum/Assets/Images/Map/RipStone.png"
#define EXPLOSION_PATH "/var/Argentum/Assets/Images/Spells/Explosion.png"
#define MAGIC_ARROW_PATH "/var/Argentum/Assets/Images/Spells/MagicArrow.png"
#define MAGIC_MISSILE_PATH "/var/Argentum/Assets/Images/Spells/MagicMissile.png"
#define HEAL_PATH "/var/Argentum/Assets/Images/Spells/Heal.png"
#define GOLD_PATH "/var/Argentum/Assets/Images/Miscellaneous/Gold.png"
#define SIMPLE_ARROW_PATH "/var/Argentum/Assets/Images/Miscellaneous/SimpleArrow.png"
#define COMPOSITE_ARROW_PATH "/var/Argentum/Assets/Images/Miscellaneous/CompositeArrow.png"
#define BACKGROUND_PATH "/var/Argentum/Assets/Images/UI/Background.png"
#define MAIN_MENU_PATH "/var/Argentum/Assets/Images/UI/MainMenuBackground.png"

TextureRepository::TextureRepository(SDL_Renderer& renderer) : renderer(renderer) {
    _loadClothing();
    _loadHeads();
    _loadWeapons();
    _loadTiles();
    _loadStructures();
    _loadNPCS();
    _loadDrops();
    _loadMiscellaneous();
    _loadUI();
}

void TextureRepository::_loadUI() {
    _setImage(Background, BACKGROUND_PATH, 1495, 937, 0
            , 0, 1, {-1, -1, -1});
    _setImage(MainMenu, MAIN_MENU_PATH, 1499, 937, 0,
            0, 1, {-1, -1, -1});
}

void TextureRepository::_loadMiscellaneous() {
    _setSpellImage(Explosion, EXPLOSION_PATH, 256, 256, -10, -10);
    _setSpellImage(MagicArrow, MAGIC_ARROW_PATH, 96, 100, 20, 15);
    _setSpellImage(MagicMissile, MAGIC_MISSILE_PATH, 128, 128, 8, 5);
    _setSpellImage(Heal, HEAL_PATH, 76, 76, 25, 20);
    _setImage(SimpleArrow, SIMPLE_ARROW_PATH, 32, 32, 45, 45, 1);
    _setImage(CompositeArrow, COMPOSITE_ARROW_PATH, 32, 32, 45, 45, 1);
}

void TextureRepository::_loadDrops() {
    _setImage(BlueTunicDrop, BLUE_TUNIC_DROP_PATH, 32, 32, 30, 30, 2);
    _setImage(LongSwordDrop, LONG_SWORD_DROP_PATH, 32, 32, 33, 30, 2);
    _setImage(LinkedStaffDrop, LINKED_STAFF_DROP_PATH, 32, 32, 30, 30, 2);
    _setImage(GnarledStaffDrop, GNARLED_STAFF_DROP_PATH, 32, 32, 35, 30, 2);
    _setImage(MagicHatDrop, MAGIC_HAT_DROP_PATH, 32, 32, 50, 45);
    _setImage(HealthPotion, HEALTH_POTION_PATH, 32, 32, 50, 45);
    _setImage(ManaPotion, MANA_POTION_PATH, 32, 32, 50, 45);
```

```cpp
    _setImage(CommonClothingDrop, COMMON_CLOTHING_DROP_PATH, 32, 32, 35, 30, 2);
    _setImage(KingArmorDrop, KING_ARMOR_DROP_PATH, 32, 32, 35, 30, 2);
    _setImage(LeatherArmorDrop, LEATHER_ARMOR_DROP_PATH, 32, 32, 35, 30, 2);
    _setImage(PlateArmorDrop, PLATE_ARMOR_DROP_PATH, 16, 32, 48, 35, 2);
    _setImage(HoodDrop, HOOD_DROP_PATH, 32, 32, 50, 45);
    _setImage(IronHelmetDrop, IRON_HELMET_DROP_PATH, 32, 32, 50, 45);
    _setImage(IronShieldDrop, IRON_SHIELD_DROP_PATH, 32, 32, 35, 30, 2);
    _setImage(TurtleShieldDrop, TURTLE_SHIELD_DROP_PATH, 32, 32, 50, 45);
    _setImage(AshRodDrop, ASH_ROD_DROP_PATH, 32, 32, 35, 30, 2);
    _setImage(AxeDrop, AXE_DROP_PATH, 32, 32, 32, 30, 2);
    _setImage(CompositeBowDrop, COMPOSITE_BOW_DROP_PATH, 32, 32, 32, 30, 2);
    _setImage(ElvenFluteDrop, ELVEN_FLUTE_DROP_PATH, 32, 32, 32, 30, 2);
    _setImage(SimpleBowDrop, SIMPLE_BOW_DROP_PATH, 32, 32, 32, 30, 2);
    _setImage(WarHammerDrop, WAR_HAMMER_DROP_PATH, 32, 32, 32, 28, 2);
    _setImage(Gold, GOLD_PATH, 32, 32, 45, 50, 1);
}

void TextureRepository::_loadClothing() {
    _setBodyImage(BlueTunic, BLUE_TUNIC_PATH);
    _setBodyImage(CommonClothing, COMMON_CLOTHING_PATH);
    _setShieldImage(IronShield, IRON_SHIELD_PATH);
    _setBodyImage(LeatherArmor, LEATHER_ARMOR_PATH);
    _setBodyImage(PlateArmor, PLATE_ARMOR_PATH);
    _setBodyImage(KingArmor, KING_ARMOR_PATH);
    _setShieldImage(TurtleShield, TURTLE_SHIELD_PATH);
    _setHelmetImage(Hood, HOOD_PATH);
    _setHelmetImage(IronHelmet, IRON_HELMET_PATH);
    _setHelmetImage(MagicHat, MAGIC_HAT_PATH, −1, −24);
}

void TextureRepository::_loadHeads() {
    _setHeadImage(DwarfHead, DWARF_HEAD_PATH);
    _setHeadImage(ElfHead, ELF_HEAD_PATH);
    _setHeadImage(GnomeHead, GNOME_HEAD_PATH);
    _setHeadImage(HumanHead, HUMAN_HEAD_PATH);
}

void TextureRepository::_loadWeapons() {
    _setWeaponImage(AshRod, ASH_ROD_PATH);
    _setWeaponImage(Axe, AXE_PATH);
    _setWeaponImage(CompositeBow, COMPOSITE_BOW_PATH);
    _setWeaponImage(LinkedStaff, LINKED_STAFF_PATH);
    _setWeaponImage(GnarledStaff, GNARLED_STAFF_PATH);
    _setWeaponImage(LongSword, LONG_SWORD_PATH);
    _setWeaponImage(SimpleBow, SIMPLE_BOW_PATH);
    _setWeaponImage(WarHammer, WAR_HAMMER_PATH);
}

void TextureRepository::_loadTiles() {
    _setTileImage(Grass, GRASS_PATH, false);
    _setTileImage(PrettyGrass, PRETTY_GRASS_PATH, false);
    _setTileImage(PrettyRoad, PRETTY_ROAD_PATH, false);
    _setTileImage(DeadGrass, DEAD_GRASS_PATH, false);
    _setTileImage(Water, WATER_PATH, false);
    _setTileImage(DarkWater, DARK_WATER_PATH, false);
    _setTileImage(Sand, SAND_PATH, true);
}

void TextureRepository::_loadStructures() {
    _setImage(Tree, TREE_PATH, 256, 256, −60, −180);
    _setImage(LongTree, LONG_TREE_PATH, 256, 256, −60, −180);
    _setImage(FatTree, FAT_TREE_PATH, 256, 256, −60, −180);
    _setImage(PalmTree, PALM_TREE_PATH, 256, 256, −60, −180);
    _setImage(DeadTree, DEAD_TREE_PATH, 256, 256, −60, −160);
    _setImage(Bush, BUSH_PATH, 75, 65, 35, 35);
    _setImage(BoneGuy, BONE_GUY_PATH, 75, 65, 30, 40);
```

```cpp
    _setImage(BrokenRipStone, BROKEN_RIP_STONE_PATH, 75, 65, 30, 20);
    _setImage(DeadGuy, DEAD_GUY_PATH, 75, 65, 30, −60, 2);
    _setImage(VeryDeadGuy, VERY_DEAD_GUY_PATH, 75, 65, 0, 10, 2);
    _setImage(HangedGuy, HANGED_GUY_PATH, 75, 65, 5, −60, 2);
    _setImage(RipStone, RIP_STONE_PATH, 75, 65, 30, 40);
    _setImage(DeadBush, DEAD_BUSH_PATH, 75, 65, 30, 40);
    _setImage(House1, HOUSE1_PATH, 196, 200, 40, −150);
    _setImage(House2, HOUSE2_PATH, 181, 213, 40, −150);
    _setImage(House3, HOUSE3_PATH, 200, 239, 30, −165);
    _setImage(SunkenShip, SUNKEN_SHIP_PATH, 256, 256, −120, −10, 2);
    _setImage(SunkenColumn, SUNKEN_COLUMN_PATH, 256, 256, 5, −185);
}

void TextureRepository::_loadNPCS() {
    _setNPCImage(Skeleton, SKELETON_PATH, 25, 52, 35, 30);
    _setNPCImage(Goblin, GOBLIN_PATH, 24, 31, 38, 48);
    _setNPCImage(Zombie, ZOMBIE_PATH, 25, 45, 35, 30);
    _setNPCImage(Spider, SPIDER_PATH, 34, 34, 30, 48);
    _setNPCImage(Priest, PRIEST_PATH, 25, 45, 37, 33);
    _setNPCImage(Trader, TRADER_PATH, 24, 48,37, 33);
    _setNPCImage(Banker, BANKER_PATH, 25, 45, 37, 33);
    _setNPCImage(Guard, GUARD_PATH, 28, 52, 37, 33);
    _setNPCImage(PlayerGhost, PLAYER_GHOST_PATH, 47, 71, 20, −10); /*tiene el mi
smo formato*/
}

void TextureRepository::_setImage(TextureID TextureID, std::string^ image,
                    int width, int height, int xOffset, int yOffset, int scale,
ColorKey_t key) {
    try {
        textures.emplace(TextureID, renderer);
        Texture& texture = textures.at(TextureID);
        texture.loadFromFile(image, key, xOffset, yOffset, scale);
        _addSprites(texture, width, height);
    } catch (TPException& e) {
        throw TPException("Failed to load %s sprite sheet texture!\n", image.c_str());
    }
}

void TextureRepository::_setSpellImage(TextureID TextureID, std::string^ spellI
mage,
                                                int width, int height, int xOffset, i
nt yOffset) {
    try {
        ColorKey_t key = {0, 0, 0};
        textures.emplace(TextureID, renderer);
        Texture& texture = textures.at(TextureID);
        texture.loadFromFile(spellImage, key, xOffset, yOffset);
        _addSpellSprites(texture, 0, width, height);
        _addSpellSprites(texture, height, width, height);
        _addSpellSprites(texture, 2*height, width, height);
        _addSpellSprites(texture, 3*height, width, height);
    } catch (TPException& e) {
        throw TPException("Failed to load %s sprite sheet texture!\n", spellImage.c_str());
    }
}

void TextureRepository::_setTileImage(TextureID TextureID, std::string^ tileIma
ge, bool individualTile) {
    try {
        textures.emplace(TextureID, renderer);
        Texture& texture = textures.at(TextureID);
        texture.loadFromFile(tileImage);
        _addTileSprites(texture, 0, individualTile);
    } catch (TPException& e) {
        throw TPException("Failed to load %s sprite sheet texture!\n", tileImage.c_str());
    }
}

void TextureRepository::_setNPCImage(TextureID TextureID, std::string∧ npcImage
, int width, int height
                                        , int xOffset, int yOffset) {
    try {
        ColorKey_t key = {0, 0, 0};
        textures.emplace(TextureID, renderer);
        Texture& texture = textures.at(TextureID);
        texture.loadFromFile(npcImage, key, xOffset, yOffset);
        /*Front*/
        _addNPCSprites(texture, 0, false, width, height);
        /*Back*/
        _addNPCSprites(texture, height, false, width, height);
        /*Left*/
        _addNPCSprites(texture, 2*height, true, width, height);
        /*Rigth*/
        _addNPCSprites(texture, 3*height, true, width, height);
    } catch (TPException& e) {
        throw TPException("Failed to load %s sprite sheet texture!\n", npcImage.c_str());
    }
}

void TextureRepository::_addNPCSprites(Texture& texture, int y, bool lateralSide
, int width, int height) {
    for (int i = 0; i < 5; ++i) {
        texture.addSprite(width*i, y, width, height);
    }
    if (lateralSide) texture.addSprite(4*width, y, width, height);
    else texture.addSprite(5*width, y, width, height);
}

void TextureRepository::_setBodyImage(TextureID TextureID, std::string∧ bodyIma
ge) {
    try {
        ColorKey_t key = {0, 0, 0};
        textures.emplace(TextureID, renderer);
        Texture& texture = textures.at(TextureID);
        texture.loadFromFile(bodyImage, key);
        /*Front*/
        _addBodySprites(texture, 0, false);
        /*Back*/
        _addBodySprites(texture, 45, false);
        /*Left*/
        _addBodySprites(texture, 90, true);
        /*Rigth*/
        _addBodySprites(texture, 135, true);
    } catch (TPException& e) {
        throw TPException("Failed to load %s sprite sheet texture!\n", bodyImage.c_str());
    }
}

void TextureRepository::_setWeaponImage(TextureID TextureID, std::string∧ weapo
nImage) {
    try {
        ColorKey_t key = {0, 0, 0};
        textures.emplace(TextureID, renderer);
        Texture& texture = textures.at(TextureID);
        texture.loadFromFile(weaponImage, key);
        /*Front*/
        _addWeaponSprites(texture, 0, false);
        /*Back*/
        _addWeaponSprites(texture, 45, false);
        /*Left*/
        _addWeaponSprites(texture, 90, true);
```

```cpp
        /*Rigth*/
        _addWeaponSprites(texture, 135, true);
    } catch (TPException& e) {
        throw TPException("Failed to load %s sprite sheet texture!\n", weaponImage.c_str());
    }
}

void TextureRepository::_addWeaponSprites(Texture& texture, int y, bool lateralS
ide) {
    texture.addSprite(0, y, 24, 45);
    texture.addSprite(25, y, 25, 45);
    texture.addSprite(51, y − 1, 23, 45);
    texture.addSprite(76, y − 1, 23, 45);
    texture.addSprite(101, y − 1, 24, 45);
    if (lateralSide) texture.addSprite(101, y, 24, 45);
    else texture.addSprite(126, y, 25, 45);
}

void TextureRepository::_addBodySprites(Texture& texture, int y, bool lateralSid
e) {
    texture.addSprite(0, y, 24, 45); /*hasta 24 porque sino en la plate armor ha
y un poco de la otra imagen*/
    texture.addSprite(25, y, 25, 45);
    texture.addSprite(51, y, 24, 45); /*pongo 51 porque sino se veia un poco del
 pie de otro en algunas ropas*/
    texture.addSprite(75, y, 25, 45);
    texture.addSprite(100, y, 25, 45);
    if (lateralSide) texture.addSprite(100, y, 25, 45);
    else texture.addSprite(125, y, 25, 45);
}

void TextureRepository::_setHeadImage(TextureID TextureID, std::string∧ headIma
ge) {
    try {
        ColorKey_t key = {0, 0, 0};
        textures.emplace(TextureID, renderer);
        Texture& texture = textures.at(TextureID);
        texture.loadFromFile(headImage, key);
        texture.addSprite(0, 0, 17, 15);
        texture.addSprite(17, 0, 17, 15);
        texture.addSprite(34, 0, 17, 15);
        texture.addSprite(51, 0, 17, 15);
    } catch (TPException& e) {
        throw TPException("Failed to load %s sprite sheet texture!\n", headImage.c_str());
    }
}

void TextureRepository::_setHelmetImage(TextureID TextureID, std::string∧ helme
tImage,
                                                int xOffset, int yOffset) {
    try {
        ColorKey_t key = {0, 0, 0};
        textures.emplace(TextureID, renderer);
        Texture& texture = textures.at(TextureID);
        texture.loadFromFile(helmetImage, key, xOffset, yOffset);
        texture.addSprite(0, 0, 17, 17);
        texture.addSprite(17, 0, 17, 17);
        texture.addSprite(34, 0, 17, 17);
        texture.addSprite(51, 0, 17, 17);
    } catch (TPException& e) {
        throw TPException("Failed to load %s sprite sheet texture!\n", helmetImage.c_str());
    }
}

void TextureRepository::_setShieldImage(TextureID TextureID, std::string∧ shiel
dImage) {
    try {
        ColorKey_t key = {0, 0, 0};
        textures.emplace(TextureID, renderer);
        Texture& texture = textures.at(TextureID);
        texture.loadFromFile(shieldImage, key);
        /*Front*/
        _addShieldSprites(texture, 0, false);
        /*Back*/
        _addShieldSprites(texture, 45, false);
        /*Left*/
        _addShieldSprites(texture, 90, true);
        /*Rigth*/
        _addShieldSprites(texture, 135, true);
    } catch (TPException& e) {
        throw TPException("Failed to load %s sprite sheet texture!\n", shieldImage.c_str());
    }
}

void TextureRepository::_addShieldSprites(Texture& texture, int y, bool lateralSide) {
    texture.addSprite(0, y, 25, 35);
    texture.addSprite(26, y, 25, 35);
    texture.addSprite(51, y, 24, 35);
    texture.addSprite(76, y, 25, 35);
    texture.addSprite(101, y, 24, 35);
    if (lateralSide) texture.addSprite(101, y, 24, 35);
    else texture.addSprite(126, y, 25, 35);
}

void TextureRepository::_addTileSprites(Texture& texture, int y, bool individualTile) {
    texture.addSprite(0, 0, TILE_WIDTH, TILE_HEIGHT);
    if (¬individualTile) {
        texture.addSprite(TILE_WIDTH, 0, TILE_WIDTH, TILE_HEIGHT);
        texture.addSprite(2*TILE_WIDTH, 0, TILE_WIDTH, TILE_HEIGHT);
        texture.addSprite(3*TILE_WIDTH, 0, TILE_WIDTH, TILE_HEIGHT);
    }
}

void TextureRepository::_addSprites(Texture& texture, int width, int height) {
    texture.addSprite(0, 0, width, height);
}

void TextureRepository::_addSpellSprites(Texture& texture, int y, int width, int height) {
    for (int i = 0; i < 6; ++i) {
        texture.addSprite(width*i, y, width, height);
    }
}

Texture& TextureRepository::getTexture(TextureID texture) {
    return textures.at(texture);
}

SDL_Renderer &TextureRepository::getRenderer() const {
    return renderer;
}
```

```cpp
//
// Created by marcos on 7/5/20.
//

#ifndef ARGENTUM_TEXTUREID_H
#define ARGENTUM_TEXTUREID_H

enum TextureID {
    Nothing, /*Auxiliar, lo uso para el equipo del Player*/
    PlayerGhost,
    BlueTunic,
    BlueTunicDrop,
    CommonClothing,
    CommonClothingDrop,
    Hood,
    HoodDrop,
    IronHelmet,
    IronHelmetDrop,
    IronShield,
    IronShieldDrop,
    KingArmor,
    KingArmorDrop,
    LeatherArmor,
    LeatherArmorDrop,
    MagicHat,
    MagicHatDrop,
    PlateArmor,
    PlateArmorDrop,
    TurtleShield,
    TurtleShieldDrop,
    DwarfHead,
    ElfHead,
    GnomeHead,
    HumanHead,
    AshRod,
    AshRodDrop,
    Axe,
    AxeDrop,
    CompositeBow,
    CompositeBowDrop,
    SimpleArrow,
    CompositeArrow,
    ElvenFluteDrop,
    LinkedStaff,
    LinkedStaffDrop,
    GnarledStaff,
    GnarledStaffDrop,
    LongSword,
    LongSwordDrop,
    SimpleBow,
    SimpleBowDrop,
    WarHammer,
    WarHammerDrop,
    HealthPotion,
    ManaPotion,
    Grass,
    PrettyGrass,
    DeadGrass,
    PrettyRoad,
    Sand,
    Water,
    DarkWater,
    Skeleton,
    Goblin,
    Zombie,
    Spider,
    Priest,
    Trader,
    Banker,
    Guard,
    Tree,
    LongTree,
    FatTree,
    PalmTree,
    DeadTree,
    Bush,
    DeadBush,
    House1,
    House2,
    House3,
    SunkenColumn,
    SunkenShip,
    BoneGuy,
    BrokenRipStone,
    DeadGuy,
    VeryDeadGuy,
    HangedGuy,
    RipStone,
    Explosion,
    MagicArrow,
    MagicMissile,
    Heal,
    Gold,
    Background,
    MainMenu
};

#endif //ARGENTUM_TEXTUREID_H
```

```
//
// Created by marcos on 6/6/20.
//

#ifndef ARGENTUM_TEXTURE_H
#define ARGENTUM_TEXTURE_H

/*Esta clase representa la textura cargada. La textura puede contener mas de
 * una imagen, permitiendo renderizar solo lo que el programador elija*/

#include <SDL.h>
#include <SDL_image.h>
#include "../../libs/TPException.h"
#include "../Graphics/Text/Font.h"
#include <string>
#include <vector>


struct ColorKey_t {
    int red;
    int green;
    int blue;
};

struct SpriteDimensions_t {
    int width;
    int height;
};

class Texture {
private:
    SDL_Renderer& renderer;
    SDL_Texture* mTexture;
    int mWidth;
    int mHeight;
    int xOffset;
    int yOffset;
    int defaultScale;
    std::vector<SDL_Rect> gSpriteClips; /*Sprites de la textura*/

public:
    Texture(SDL_Renderer& renderer);
    ~Texture();
    Texture(const Texture&) = delete;
    Texture& operator=(const Texture&) = delete;
    Texture(Texture^ other) noexcept;

    /*Carga la imagen de path, ignorando el color recibido en key. Opcionalmente
     * se le puede setear un offset de renderizacion y una escala distinta a la
     * imagen*/
    void loadFromFile(const std::string& path, ColorKey_t key = {−1, −1, −1},
                                int xOff = 0, int yOff = 0, int scale = 1);

    /*Especifica una dimension (un clip) que representa un sprite de la textura*
/
    void addSprite(int x, int y, int width, int height);

    /*Hago sobrecarga para poder pasar por parametro default a la escala de la t
extura
     * cuando la cree*/
    void render(int x, int y, int spritePosition = 0, double angle = 0);

    /*Renderiza el sprite de la textura en la posicion, angulo y escala indicado
s*/
    void render(int x, int y, int spritePosition, double angle, int scale);

    /*Retorna las dimensiones del sprite de la textura*/
    SpriteDimensions_t getSpriteDimensions(int spritePosition = 0);

    /*Crea una textura en base al texto recibido*/
    void loadFromRenderedText(const std::string& text, SDL_Color textColor, TTF_
Font* font);


private:
    //Deallocates texture
    void _free();
};


#endif //ARGENTUM_TEXTURE_H
```

```
//
// Created by marcos on 6/6/20.
//

#include "Texture.h"

Texture::Texture(SDL_Renderer& renderer) : renderer(renderer) {
    mTexture = nullptr;
    mWidth = 0;
    mHeight = 0;
    xOffset = 0;
    yOffset = 0;
    defaultScale = 1;
}

Texture::~Texture() {
    _free();
}

void Texture::loadFromFile(const std::string& path, ColorKey_t key, int xOff, in
t yOff,
                                                                  int scale) {
    //Libero la textura anterior
    _free();

    //cargo la imagen de path
    SDL_Surface* loadedSurface = IMG_Load(path.c_str());
    if (loadedSurface ≡ nullptr) {
        throw TPException("Unable to load image %s! SDL_image Error: %s\n",
                          path.c_str(), IMG_GetError() );
    } else {
        if (key.red > −1 ∧ key.green > −1 ∧ key.blue > −1) {
            SDL_SetColorKey(loadedSurface, SDL_TRUE,
                            SDL_MapRGB(loadedSurface→format, key.red, key.green
, key.blue));
            /*Con esto aclaras que pixel hacer transparente*/
        }

        //Crea la textura
        mTexture = SDL_CreateTextureFromSurface(&renderer, loadedSurface);
        if (mTexture ≡ nullptr) {
            //Si falla libero la superficie
            SDL_FreeSurface(loadedSurface);
            throw TPException("Unable to create texture from %s! "
                              "Graphics Error: %s\n", path.c_str(), SDL_GetError());
        } else {
            mWidth = loadedSurface→w;
            mHeight = loadedSurface→h;
        }

        //Libero la superficie
        SDL_FreeSurface(loadedSurface);
    }

    xOffset = xOff;
    yOffset = yOff;
    defaultScale = scale;
}

void Texture::_free() {
    if (mTexture ≠ nullptr) {
        SDL_DestroyTexture(mTexture);
        mTexture = nullptr;
        mWidth = 0;
        mHeight = 0;
    }
```

```cpp
}

void Texture::render(int x, int y, int spritePosition, double angle, int scale)
{
    SDL_Rect renderQuad = {x + xOffset, y + yOffset, mWidth, mHeight};
    SDL_Rect& clip = gSpriteClips.at(spritePosition);

    //Setea las dimensiones del rectangulo a renderizar
    renderQuad.w = clip.w*scale;
    renderQuad.h = clip.h*scale;

    //Renderiza
    SDL_RenderCopyEx(&renderer, mTexture, &clip, &renderQuad, angle,
            nullptr, SDL_FLIP_NONE);
}

void Texture::addSprite(int x, int y, int width, int height) {
    gSpriteClips.push_back({x, y, width, height});
}

Texture::Texture(Texture∧ other) noexcept : renderer(other.renderer) {
    mWidth = other.mWidth;
    mHeight = other.mHeight;
    other.mWidth = 0;
    other.mHeight = 0;
    xOffset = other.xOffset;
    yOffset = other.yOffset;
    other.xOffset = 0;
    other.yOffset = 0;
    defaultScale = other.defaultScale;
    other.defaultScale = 1;
    mTexture = other.mTexture;
    other.mTexture = nullptr;
    gSpriteClips = std::move(other.gSpriteClips);
}

SpriteDimensions_t Texture::getSpriteDimensions(int spritePosition) {
    SDL_Rect& spriteDimensions = gSpriteClips.at(spritePosition);
    SpriteDimensions_t dimensions = {spriteDimensions.w, spriteDimensions.h};
    return dimensions;
}

void Texture::loadFromRenderedText(const std::string& text, SDL_Color
                                                textColor, TTF_Font* font) {
    //Libero la textura anterior
    _free();

    //Creo una superficie con el texto
    SDL_Surface* textSurface = TTF_RenderText_Solid(font, text.c_str(), textColo
r);
    if(textSurface ≡ nullptr) {
        throw TPException("Unable to _render text surface! SDL_ttf Error:"
                          " %s\n", TTF_GetError());
    } else {
        //Crea la textura
        mTexture = SDL_CreateTextureFromSurface(&renderer, textSurface);

        if(mTexture ≡ nullptr) {
            //Si falla libera la superficie
            SDL_FreeSurface(textSurface);
            throw TPException("Unable to create texture from rendered text! "
                              "Graphics Error: %s\n", SDL_GetError());
        } else {
            mWidth = textSurface→w;
            mHeight = textSurface→h;
            gSpriteClips.assign(1, {0, 0, mWidth, mHeight});
```

```cpp
        }

        //Libero al superficie
        SDL_FreeSurface(textSurface);
    }
}

void Texture::render(int x, int y, int spritePosition, double angle) {
    render(x, y, spritePosition, angle, defaultScale);
}
```

```
//
// Created by marcos on 6/6/20.
//

#ifndef ARGENTUM_PLAYERTEXTURE_H
#define ARGENTUM_PLAYERTEXTURE_H


#include "EntityTexture.h"
#include "../../libs/TPException.h"
#include "TextureRepository.h"
#include "PlayerEquipment.h"
#include "../../libs/GameEnums.h"

/*Representa la textura de un player*/

class PlayerTexture : public EntityTexture {
private:
    TextureRepository& textureRepo;
    Texture* helmet;
    Texture* head;
    Texture* body;
    Texture* shield;
    Texture* weapon;
    Font textFont;
    Text nickname, level;
    int textNicknameOffset{0}, textLevelOffset{0};

public:
    PlayerTexture(TextureRepository& repo, PlayerEquipment equipment, const std:
:string& _level,
                    const std::string& _nickname = "");

    void renderFront(int x, int y, int frame) override;
    void renderBack(int x, int y, int frame) override;
    void renderRight(int x, int y, int frame) override;
    void renderLeft(int x, int y, int frame) override;

    void setLevel(const std::string &_level);

    /*Cambia una textura del player*/
    void equip(GameType::EquipmentPlace place, TextureID equipment);
};


#endif //ARGENTUM_PLAYERTEXTURE_H
```

```
//
// Created by marcos on 6/6/20.
//

#include "PlayerTexture.h"
#include "../Client/GameConstants.h"

PlayerTexture::PlayerTexture(TextureRepository& repo, PlayerEquipment equipment,
                            const std::string& _level, const std::string& _nickn
ame) :textureRepo(repo),
                            textFont("/var/Argentum/Assets/Fonts/Raleway−Medium.ttf", 20),
                            nickname(textFont, repo.getRenderer(), _nickname),
                            level(textFont, repo.getRenderer(), "(" + _level + "
)") {

    textLevelOffset = level.getTextTextureWidth();
    textNicknameOffset = (nickname.getTextTextureWidth() + textLevelOffset)/2;
    if (equipment.helmet ≠ Nothing) helmet = &textureRepo.getTexture(equipment.h
elmet);
    else helmet = nullptr;
    head = &textureRepo.getTexture(equipment.head);
    body = &textureRepo.getTexture(equipment.body);
    if (equipment.shield ≠ Nothing) shield = &textureRepo.getTexture(equipment.s
hield);
    else shield = nullptr;
    if (equipment.weapon ≠ Nothing) weapon = &textureRepo.getTexture(equipment.w
eapon);
    else weapon = nullptr;
}

void PlayerTexture::renderFront(int x, int y, int frame) {
    if (frame < 0 ∨ frame > 5) throw TPException("I dont have that character frame!");
    EntityTexture::render(head, x + 45, y + 15, 0);
    EntityTexture::render(body, x + 37, y + 30, frame);
    EntityTexture::render(helmet, x + 45, y + 15, 0);
    EntityTexture::render(shield, x + 52, y + 30, frame);
    EntityTexture::render(weapon, x + 37, y + 15, frame);
    nickname.render(x + TILE_WIDTH/2 − textNicknameOffset, y + TILE_HEIGHT − 15)
;
    level.render(x + TILE_WIDTH/2 + textNicknameOffset − textLevelOffset, y + TI
LE_HEIGHT − 15);
}

void PlayerTexture::renderBack(int x, int y, int frame) {
    if (frame < 0 ∨ frame > 5) throw TPException("I dont have that character frame!");
    EntityTexture::render(head, x + 45, y + 15, 3);
    EntityTexture::render(weapon, x + 40, y + 20, frame + 6);
    EntityTexture::render(shield, x + 37, y + 15, frame + 6);
    EntityTexture::render(body, x + 37, y + 30, frame + 6);
    EntityTexture::render(helmet, x + 45, y + 11, 3);
    nickname.render(x + TILE_WIDTH/2 − textNicknameOffset, y + TILE_HEIGHT − 15)
;
    level.render(x + TILE_WIDTH/2 + textNicknameOffset − textLevelOffset, y + TI
LE_HEIGHT − 15);
}

void PlayerTexture::renderRight(int x, int y, int frame) {
    if (frame < 0 ∨ frame > 5) throw TPException("I dont have that character frame!");
    EntityTexture::render(head, x + 46, y + 15, 1);
    EntityTexture::render(shield, x + 40, y + 30, frame + 18);
    EntityTexture::render(body, x + 37, y + 30, frame + 18);
    EntityTexture::render(helmet, x + 45, y + 11, 1);
    EntityTexture::render(weapon, x + 37, y + 20, frame + 18);
    nickname.render(x + TILE_WIDTH/2 − textNicknameOffset, y + TILE_HEIGHT − 15)
;
    level.render(x + TILE_WIDTH/2 + textNicknameOffset − textLevelOffset, y + TI
LE_HEIGHT − 15);
}

void PlayerTexture::renderLeft(int x, int y, int frame) {
    if (frame < 0 ∨ frame > 5) throw TPException("I dont have that character frame!");
    EntityTexture::render(head, x + 43, y + 15, 2);
    EntityTexture::render(weapon, x + 33, y + 20, frame + 12);
    EntityTexture::render(body, x + 37, y + 30, frame + 12);
    EntityTexture::render(helmet, x + 43, y + 11, 2);
    EntityTexture::render(shield, x + 47, y + 30, frame + 12);
    nickname.render(x + TILE_WIDTH/2 − textNicknameOffset, y + TILE_HEIGHT − 15)
;
    level.render(x + TILE_WIDTH/2 + textNicknameOffset − textLevelOffset, y + TI
LE_HEIGHT − 15);
}

void PlayerTexture::setLevel(const std::string &_level) {
    *(level.updateText( "(" + _level + ")"));
    textLevelOffset = level.getTextTextureWidth();
    textNicknameOffset = (nickname.getTextTextureWidth() + textLevelOffset)/2;
}

void PlayerTexture::equip(GameType::EquipmentPlace place, TextureID equipment) {
    Texture* texture = nullptr;
    if (equipment ≠ Nothing) {
        texture = &textureRepo.getTexture(equipment);
    }
    switch (place) {
        case GameType::EQUIPMENT_PLACE_HEAD:
            helmet = texture;
            break;
        case GameType::EQUIPMENT_PLACE_CHEST:
            body = texture;
            break;
        case GameType::EQUIPMENT_PLACE_SHIELD:
            shield = texture;
            break;
        case GameType::EQUIPMENT_PLACE_WEAPON:
            weapon = texture;
            break;
        default:
            //do nothing
            break;
    }
}

```

```
//
// Created by marcos on 6/27/20.
//

#ifndef ARGENTUM_PLAYEREQUIPMENT_H
#define ARGENTUM_PLAYEREQUIPMENT_H

#include "TextureID.h"

/*Encapsula lo equipado de un player*/

struct PlayerEquipment {
    TextureID helmet;
    TextureID head;
    TextureID body;
    TextureID shield;
    TextureID weapon;
};

#endif //ARGENTUM_PLAYEREQUIPMENT_H
```

```
//
// Created by marcos on 6/8/20.
//

#ifndef ARGENTUM_NPCTEXTURE_H
#define ARGENTUM_NPCTEXTURE_H

#include "EntityTexture.h"
#include "../../libs/TPException.h"
#include "TextureRepository.h"
#include "../Graphics/Text/Text.h"

/*Representa la textura de un npc*/

class NPCTexture : public EntityTexture {
private:
    TextureRepository& textureRepo;
    Font textFont;
    Texture* body;
    Text nickname, level;
    int textNicknameOffset{0}, textLevelOffset{0};

public:
    explicit NPCTexture(TextureRepository& repo, TextureID texture, const std::string& _level = "",
                        const std::string& _nickname = "");
    void renderFront(int x, int y, int frame) override;
    void renderBack(int x, int y, int frame) override;
    void renderRight(int x, int y, int frame) override;
    void renderLeft(int x, int y, int frame) override;
    void setLevel(const std::string& _level);
};


#endif //ARGENTUM_NPCTEXTURE_H
```

```
//
// Created by marcos on 6/8/20.
//

#include "NPCTexture.h"
#include "../Client/GameConstants.h"

NPCTexture::NPCTexture(TextureRepository& repo, TextureID texture, const std::string& _level,
                        const std::string& _nickname) : textureRepo(repo),
                    textFont("/var/Argentum/Assets/Fonts/Raleway−Medium.ttf", 20),
                    nickname(textFont, repo.getRenderer(), _nickname),
                    level(textFont, repo.getRenderer(), _level) {

    textLevelOffset = level.getTextTextureWidth();
    textNicknameOffset = (nickname.getTextTextureWidth() + textLevelOffset)/2;
    if (textLevelOffset ≡ 0) {
        textLevelOffset = TILE_WIDTH/2 − 30;
    }
    body = &textureRepo.getTexture(texture);
}

void NPCTexture::renderFront(int x, int y, int frame) {
    if (frame < 0 ∨ frame > 5) throw TPException("I dont have that character frame!");
    EntityTexture::render(body, x + 4, y − 20, frame);
    nickname.render(x + TILE_WIDTH/2 − textNicknameOffset, y + TILE_HEIGHT − 15);
    level.render(x + TILE_WIDTH/2 + textNicknameOffset − textLevelOffset, y + TILE_HEIGHT − 15);
}

void NPCTexture::renderBack(int x, int y, int frame) {
    if (frame < 0 ∨ frame > 5) throw TPException("I dont have that character frame!");
    EntityTexture::render(body, x + 4, y − 20, frame + 6);
    nickname.render(x + TILE_WIDTH/2 − textNicknameOffset, y + TILE_HEIGHT − 15);
    level.render(x + TILE_WIDTH/2 + textNicknameOffset − textLevelOffset, y + TILE_HEIGHT − 15);
}

void NPCTexture::renderRight(int x, int y, int frame) {
    if (frame < 0 ∨ frame > 5) throw TPException("I dont have that character frame!");
    EntityTexture::render(body, x + 4, y − 20, frame + 18);
    nickname.render(x + TILE_WIDTH/2 − textNicknameOffset, y + TILE_HEIGHT − 15);
    level.render(x + TILE_WIDTH/2 + textNicknameOffset − textLevelOffset, y + TILE_HEIGHT − 15);
}

void NPCTexture::renderLeft(int x, int y, int frame) {
    if (frame < 0 ∨ frame > 5) throw TPException("I dont have that character frame!");
    EntityTexture::render(body, x + 4, y − 20, frame + 12);
    nickname.render(x + TILE_WIDTH/2 − textNicknameOffset, y + TILE_HEIGHT − 15);
    level.render(x + TILE_WIDTH/2 + textNicknameOffset − textLevelOffset, y + TILE_HEIGHT − 15);
}

void NPCTexture::setLevel(const std::string &_level) {
    *(level.updateText(_level));
    textLevelOffset = level.getTextTextureWidth();
    textNicknameOffset = (nickname.getTextTextureWidth() + textLevelOffset)/2;
}

```

```
//
// Created by marcos on 6/9/20.
//

#ifndef ARGENTUM_ENTITYTEXTURE_H
#define ARGENTUM_ENTITYTEXTURE_H

#include "Texture.h"
#include "../Graphics/Text/Text.h"

/*Clase Abstracta, los hijos deben implementar la renderizacion*/

class EntityTexture {
public:
    virtual void renderFront(int x, int y, int frame) = 0;
    virtual void renderBack(int x, int y, int frame) = 0;
    virtual void renderRight(int x, int y, int frame) = 0;
    virtual void renderLeft(int x, int y, int frame) = 0;
    static void render(Texture* texture, int x, int y, int spritePosition);
    virtual ~EntityTexture() = default;
};


#endif //ARGENTUM_ENTITYTEXTURE_H
```

```
//
// Created by marcos on 6/9/20.
//

#include "EntityTexture.h"

const int SCALE = 2; /*Factor de escala de la imagen*/

void EntityTexture::render(Texture* texture, int x, int y, int spritePosition) {
    if (texture ≠ nullptr) texture→render(x, y, spritePosition, 0, SCALE);
}
```

```cpp
#ifndef ARGENTUM_SOUNDREPOSITORY_H
#define ARGENTUM_SOUNDREPOSITORY_H


#include <iostream>
#include <unordered_map>
#include <queue>
#include "../../libs/TPException.h"
#include "Sound.h"

enum SoundID {SwordAttackSound, ArrowSound, Explotion1Sound, Explotion2Sound, Ex
plotion3Sound,
        Death1Sound, Death2Sound, Attack1Sound, Attack2Sound, HeavyAttackSound,
StepSound, HealingSound,
        ZombieSound, SpiderSound, SkeletonSound, GoblinSound, LevelUpSound};

class SoundRepository {
private:
    std::unordered_map<SoundID, Sound> sounds;
    Mix_Music* music{};
public:
    SoundRepository();

    /* Me devuelve la musica */
    Mix_Music* getMusic();

    /*Me devuelve el sonido */
    Mix_Chunk* getSound(SoundID id);

    ~SoundRepository();

private:
    void _loadSounds();
    void _loadMusic();
};


#endif //ARGENTUM_SOUNDREPOSITORY_H
```

```cpp
#include "SoundRepository.h"

#define QUEUE_SIZE 3

#define SWORD_ATTACK_PATH "/var/Argentum/Assets/Sounds/swordAttack.wav"
#define HEAVY_ATTACK_PATH "/var/Argentum/Assets/Sounds/HeavyAttack.wav"
#define ATTACK_1_PATH "/var/Argentum/Assets/Sounds/genericAttack1.wav"
#define ATTACK_2_PATH "/var/Argentum/Assets/Sounds/genericAttack2.wav"
#define ARROW_PATH "/var/Argentum/Assets/Sounds/arrow.wav"
#define DEATH_1_PATH "/var/Argentum/Assets/Sounds/Death.wav"
#define DEATH_2_PATH "/var/Argentum/Assets/Sounds/YodaDeath.wav"
#define STEP_PATH "/var/Argentum/Assets/Sounds/Step.wav"
#define EXPLOTION_1_PATH "/var/Argentum/Assets/Sounds/Explotion1.wav"
#define EXPLOTION_2_PATH "/var/Argentum/Assets/Sounds/Explotion2.wav"
#define EXPLOTION_3_PATH "/var/Argentum/Assets/Sounds/Explotion3.wav"
#define HEALING_PATH "/var/Argentum/Assets/Sounds/heal.wav"
#define ZOMBIE_PATH "/var/Argentum/Assets/Sounds/Zombie.wav"
#define SPIDER_PATH "/var/Argentum/Assets/Sounds/Spider.wav"
#define SKELETON_PATH "/var/Argentum/Assets/Sounds/Skeleton.wav"
#define GOBLIN_PATH "/var/Argentum/Assets/Sounds/Goblin.wav"
#define LEVEL_UP_SOUND "/var/Argentum/Assets/Sounds/LevelUp.wav"

#define MUSIC_PATH "/var/Argentum/Assets/Sounds/argentumOnlineOST.mp3"

SoundRepository::SoundRepository() {
    _loadSounds();
    _loadMusic();
}

void SoundRepository::_loadSounds() {
    try {
        sounds.emplace(SwordAttackSound, SWORD_ATTACK_PATH);
        sounds.emplace(HeavyAttackSound, HEAVY_ATTACK_PATH);
        sounds.emplace(Attack1Sound, ATTACK_1_PATH);
        sounds.emplace(Attack2Sound, ATTACK_2_PATH);
        sounds.emplace(ArrowSound, ARROW_PATH);
        sounds.emplace(Death1Sound, DEATH_1_PATH);
        sounds.emplace(Death2Sound, DEATH_2_PATH);
        sounds.emplace(StepSound, STEP_PATH);
        sounds.emplace(Explotion1Sound, EXPLOTION_1_PATH);
        sounds.emplace(Explotion2Sound, EXPLOTION_2_PATH);
        sounds.emplace(Explotion3Sound, EXPLOTION_3_PATH);
        sounds.emplace(HealingSound, HEALING_PATH);
        sounds.emplace(ZombieSound, ZOMBIE_PATH);
        sounds.emplace(SpiderSound, SPIDER_PATH);
        sounds.emplace(SkeletonSound, SKELETON_PATH);
        sounds.emplace(GoblinSound, GOBLIN_PATH);
        sounds.emplace(LevelUpSound, LEVEL_UP_SOUND);

    } catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
    Mix_Volume(-1, 25);
}

void SoundRepository::_loadMusic(){
    music = Mix_LoadMUS(MUSIC_PATH);
    Mix_VolumeMusic(20);
    if(music ≡ nullptr ) {
        throw TPException("Failed to load beat music! SDL_mixer Error: "
                          "%s\n", Mix_GetError());
    }
}

SoundRepository::~SoundRepository() {
    //Cierra el mixer
    Mix_FreeMusic(music);
}

Mix_Music* SoundRepository::getMusic() {
    return music;
}

Mix_Chunk* SoundRepository::getSound(SoundID id) {
    return sounds.at(id).getSound();
}
```

```cpp
//
// Created by ivan on 22/6/20.
//

#ifndef ARGENTUM_SOUNDPLAYER_H
#define ARGENTUM_SOUNDPLAYER_H

#include "SoundRepository.h"
#include <mutex>
#include "../../libs/Timer.h"

class SoundPlayer {
private:
    std::queue<SoundID> soundQueue;
    SoundRepository repo;
    std::mutex m;
    Timer timer;
    bool blocked{false}; /*Para regular que no le puedan meter muchos sonidos en
 poco tiempo*/

public:
    SoundPlayer();

    /* Encola un sonido */
    void queueSound(SoundID id);

    /* Reproduce los sonidos que estan encolados */
    void playSounds();

    /* Reproduce la musica */
    void playMusic();

    /* Pausa la musica */
    void pauseMusic();

    /* Devuelve true si la musica se esta reproduciendo */
    static bool isMusicPlaying();

private:
    static SoundID _getRandomDeathSound();
};


#endif //ARGENTUM_SOUNDPLAYER_H
```

```
//
// Created by ivan on 22/6/20.
//

#include "SoundPlayer.h"

#define QUEUE_SIZE 10

const int TIME_BETWEEN_SOUND_UPDATES = 50;

SoundPlayer::SoundPlayer() {
    std::srand(std::clock());
    timer.start();
}

void SoundPlayer::playMusic() {
    std::lock_guard<std::mutex> l(m);
    if( Mix_PlayingMusic() ≡ 0 ) {//Empieza musica si no habia
        Mix_PlayMusic(repo.getMusic(), −1);
    } else if (Mix_PausedMusic() ≡ 1) {//Resume musica si estaba en pausa
        Mix_ResumeMusic();
    }
}

void SoundPlayer::pauseMusic() {
    std::lock_guard<std::mutex> l(m);
    Mix_PauseMusic();
}

bool SoundPlayer::isMusicPlaying() {
    return ¬Mix_PausedMusic();
}

SoundID SoundPlayer::_getRandomDeathSound() {
    int rand = std::rand() % 2;
    switch (rand) {
        case 0:
            return Death1Sound;
        case 1:
            return Death2Sound;
        default:
            return Death1Sound;
    }
}

void SoundPlayer::queueSound(SoundID id) {
    std::lock_guard<std::mutex> l(m);
    if (timer.getTime() > TIME_BETWEEN_SOUND_UPDATES) {
        blocked = false;
    }
    if (¬blocked) {
        if (id ≡ Death1Sound) id = _getRandomDeathSound();
        if (soundQueue.size() < QUEUE_SIZE)
            soundQueue.push(id);
    }
}

void SoundPlayer::playSounds() {
    std::lock_guard<std::mutex> l(m);
    Mix_Chunk* soundToPlay;
    long unsigned int queueSize = soundQueue.size();
    for (long unsigned int i = 0; i < queueSize; i++){
        soundToPlay = repo.getSound(soundQueue.front());
        Mix_PlayChannel(−1, soundToPlay, 0);
        soundQueue.pop();
    }
```

```
    if (¬blocked) {
        blocked = true;
        timer.start();
    }
}
```

```
//
// Created by ivan on 10/6/20.
//

#ifndef ARGENTUM_SOUND_H
#define ARGENTUM_SOUND_H

#include <SDL_mixer.h>
#include <string>

#include "../../libs/TPException.h"

class Sound {
private:
    Mix_Chunk* sound;
public:
    explicit Sound(const std::string& path);
    Sound(const Sound&) = delete;
    Sound& operator=(const Sound&) = delete;

    /* Devuelve el sonido */
    Mix_Chunk* getSound();


    ~Sound();
};


#endif //ARGENTUM_SOUND_H
```

```
//
// Created by ivan on 10/6/20.
//

#include "Sound.h"

Sound::Sound(const std::string& path) {
    sound = Mix_LoadWAV(path.c_str());
    if(sound ≡ nullptr) {
        throw TPException("Failed to load sound effect! SDL_mixer "
                          "Error: %s\n", Mix_GetError() );
    }
}

Mix_Chunk *Sound::getSound() {
    return sound;
}

Sound::~Sound() {
    Mix_FreeChunk(sound);
}
```

```
//
// Created by marcos on 11/6/20.
//

#ifndef ARGENTUM_WINDOW_H
#define ARGENTUM_WINDOW_H

#include <SDL.h>
#include <unordered_map>

enum Viewports {
    ScreenViewport,
    MapViewport,
    InventoryViewport,
    MinichatViewport,
    PlayerInfoViewport
};

/*Esta clase maneja la instancia de la ventana del juego, tiene el ownership
 * del renderizador y se encarga de mostrar los cambios renderizados y
 * setear los viewports*/

class Window {
private:
    SDL_Window* mWindow;
    SDL_Renderer* renderer;

    std::unordered_map<Viewports, SDL_Rect> viewports;
    int mWidth;
    int mHeight;

    bool mFullScreen;
    bool mMinimized;

public:
    Window();

    /* Maneja los eventos de la ventana. Por ejemplo resize o minimizar */
    bool handleEvent(SDL_Event& e);

    /* Limpia la ventana para poder renderizar */
    void clear();

    /* Muestra lo que haya renderizado */
    void show();

    /* Setea un viewport */
    void setViewport(Viewports viewport);

    SDL_Renderer& getRenderer();
    int getWidth() const;
    int getHeight() const;

    ~Window();

private:
    void _createViewports();
    void _createWindow();
    void _createRenderer();
    void _handleResizeEvent(SDL_Event& e, bool& handled);
};

#endif //ARGENTUM_WINDOW_H
```

```
//
// Created by marcos on 11/6/20.
//

#include "Window.h"
#include "../Client/GameConstants.h"
#include "../../libs/TPException.h"
#include <SDL.h>

Window::Window() {
    mWindow = nullptr;
    renderer = nullptr;
    mFullScreen = false;
    mMinimized = false;
    mWidth = 0;
    mHeight = 0;
    _createWindow();
    _createRenderer();
    _createViewports();
}

void Window::_createViewports(){
    viewports.emplace(ScreenViewport, SDL_Rect{0,0,DEFAULT_SCREEN_WIDTH,
                                            DEFAULT_SCREEN_HEIGHT});
    viewports.emplace(MapViewport, SDL_Rect{20,236,DEFAULT_MAP_WIDTH,
                                             DEFAULT_MAP_HEIGHT});

    viewports.emplace(InventoryViewport, SDL_Rect{20 + DEFAULT_MAP_WIDTH,0,
                                                  DEFAULT_INVENTORY_WIDTH,
                                                  DEFAULT_INVENTORY_HEIGHT});

    viewports.emplace(MinichatViewport, SDL_Rect{15 ,15,
                                                 DEFAULT_MINICHAT_WIDTH,
                                            DEFAULT_MINICHAT_HEIGHT});

    viewports.emplace(PlayerInfoViewport, SDL_Rect{20, DEFAULT_MINICHAT_HEIGHT
    + DEFAULT_MAP_HEIGHT + 30, DEFAULT_PLAYER_INFO_WIDTH,
    DEFAULT_PLAYER_INFO_HEIGHT});
}

void Window::_createWindow() {
    mWindow = SDL_CreateWindow( "Argentum Online", SDL_WINDOWPOS_UNDEFINED,
            SDL_WINDOWPOS_UNDEFINED, DEFAULT_SCREEN_WIDTH, DEFAULT_SCREEN_HEIGHT
,
            SDL_WINDOW_SHOWN | SDL_WINDOW_RESIZABLE);
    if (mWindow ≠ nullptr) {
        mWidth = DEFAULT_SCREEN_WIDTH;
        mHeight = DEFAULT_SCREEN_HEIGHT;
    } else {
        throw TPException("Window could not be created! Graphics Error: %s\n",
                            SDL_GetError());
    }
}

void Window::_createRenderer() {
    renderer = SDL_CreateRenderer(mWindow, −1, SDL_RENDERER_ACCELERATED
                                        | SDL_RENDERER_PRESENTVSYNC);
    if (renderer ≡ nullptr) throw TPException("Renderer could not be created! "
                                                 "Graphics Error: %s\n", SDL_GetError()
);
    SDL_SetRenderDrawColor(renderer, 0xFF, 0xFF, 0xFF, 0xFF);
}

bool Window::handleEvent(SDL_Event& e) {
    bool handled = false;
    if (e.type ≡ SDL_WINDOWEVENT) {
        switch (e.window.event) {
            case SDL_WINDOWEVENT_SIZE_CHANGED:
                mWidth = e.window.data1;
                mHeight = e.window.data2;
                show();
                break;
            case SDL_WINDOWEVENT_EXPOSED:
                SDL_RenderPresent(renderer);
                break;
            case SDL_WINDOWEVENT_FOCUS_GAINED:
                mMinimized = false;
                break;
            case SDL_WINDOWEVENT_MAXIMIZED:
                mMinimized = false;
                break;
            case SDL_WINDOWEVENT_RESTORED:
                mMinimized = false;
                break;
        }
        handled = true;
    }
    _handleResizeEvent(e, handled);
    return handled;
}

void Window::_handleResizeEvent(SDL_Event& e, bool& handled) {
    if (e.type ≡ SDL_KEYDOWN ∧ e.key.keysym.sym ≡ SDLK_F1) {
        handled = true;
        mWidth = DEFAULT_SCREEN_WIDTH;
        mHeight = DEFAULT_SCREEN_HEIGHT;
        if (mFullScreen) {
            SDL_SetWindowFullscreen(mWindow, SDL_FALSE);
            SDL_SetWindowSize(mWindow, mWidth, mHeight);
            mFullScreen = false;
        } else {
            SDL_SetWindowFullscreen(mWindow, SDL_TRUE);
            mWidth = 1600;
            mHeight = 1024;
            SDL_SetWindowSize(mWindow, mWidth, mHeight);
            mFullScreen = true;
            mMinimized = false;
        }
    } else if (e.type ≡ SDL_KEYDOWN ∧ e.key.keysym.sym ≡ SDLK_F2) {
        handled = true;
        SDL_SetWindowFullscreen(mWindow, SDL_FALSE);
        mFullScreen = false;
        mWidth = DEFAULT_SCREEN_WIDTH;
        mHeight = DEFAULT_SCREEN_HEIGHT;
        SDL_RestoreWindow(mWindow);
        SDL_SetWindowSize(mWindow, mWidth, mHeight);
    }
}

SDL_Renderer& Window::getRenderer() {
    return *renderer;
}

Window::~Window() {
    if (renderer ≠ nullptr) SDL_DestroyRenderer(renderer);
    if (mWindow ≠ nullptr) SDL_DestroyWindow(mWindow);
}

void Window::clear() {
    SDL_SetRenderDrawColor(renderer, 0xFF, 0xFF, 0xFF, 0xFF);
    SDL_RenderClear(renderer);
}
```

```

void Window::show() {
    if (¬mMinimized) {
        float x_scale = (float)mWidth/(float)DEFAULT_SCREEN_WIDTH;
        float y_scale = (float)mHeight/(float)DEFAULT_SCREEN_HEIGHT;
        SDL_RenderSetScale(renderer, x_scale, y_scale);
        SDL_RenderPresent(renderer);
    }
}

void Window::setViewport(Viewports viewport){
    SDL_RenderSetViewport(renderer, &viewports.at(viewport));
}

int Window::getWidth() const {
    return mWidth;
}

int Window::getHeight() const {
    return mHeight;
}
```

```
//
// Created by ivan on 23/6/20.
//

#ifndef ARGENTUM_MAINMENU_H
#define ARGENTUM_MAINMENU_H

#include "../Graphics/Text/Text.h"
#include "Window.h"
#include "../../libs/GameEnums.h"

class Socket;
class GameInitializer;

struct GameStartInfo {
    GameType::Class myClass;
    GameType::Race myRace;
};

class MainMenu {
private:
    Window& window;
    Font mainMenuFont;
    Text text;//El texto general. "Exit" "Connect" etc
    Text hostInputText;
    Text portInputText;
    Text nicknameInputText;
    Text errorText;
    Text strength;
    Text constitution;
    Text intelligence;
    Text agility;
    Texture& mainMenuBackground;
    GameStartInfo info{};
    bool hostInput, portInput, nicknameInput;

public:
    MainMenu(Texture& texture, Window& window);

    /* Menu Principal antes de iniciar el juego */
    void menuScreen(bool& quit, GameInitializer& initializer, Socket& socket);


    ~MainMenu();

private:
    void _playerSelectionScreen(bool &quit, bool& createPlayer, bool& loadPlayer
);
    void _connectScreen(bool &quit, bool& goBack, Socket& socket);
    void _playerCreationScreen(bool &quit, bool& goBack);
    void _playerLoadScreen(bool &quit, bool& goBack);

    void _attemptToConnect(Socket &socket, bool &finished);
    void _connectLoadedPlayer(GameInitializer &initializer, Socket &socket, bool
 &success);
    void _connectCreatedPlayer(GameInitializer& initializer, Socket& socket, boo
l& success);

    void _renderPlayerSelectionScreen();
    void _renderConnectScreen();
    void _renderCreatePlayerScreen();
    void _renderLoadPlayerScreen();

    void _renderRace();
    void _renderClass();

```

```
    void _updateWarriorSkills();
    void _updatePaladinSkills();
    void _updateClericSkills();
    void _updateWizardSkills();
    void _updateGnomeSkills();
    void _updateDwarfSkills();
    void _updateHumanSkills();
    void _updateElfSkills();

    void _handleTextInput(SDL_Event &e);
    void _handleBackspace();
    void _verifyClassSelection(int x, int y);
    void _verifyRaceSelection(int x, int y);
    static bool _isInsideRect(int x, int y, SDL_Rect rect);
};


#endif //ARGENTUM_MAINMENU_H
```

```cpp
//
// Created by ivan on 23/6/20.
//

#include "MainMenu.h"
#include "../Client/GameConstants.h"
#include "../Client/GameInitializer.h"
#include "../../libs/Socket.h"
#include <netdb.h>

#define START_BUTTON {1375, 875, 100, 25}
#define CONNECT_BUTTON {1375, 875, 100, 25}
#define EXIT_BUTTON {50,875,90,25}
#define BACK_BUTTON {50,875,90,25}

#define INPUT_HOST_BOX {115,100,365,25}
#define INPUT_PORT_BOX {115,200,365,25}
#define INPUT_NICKNAME_BOX {165,100,365,25}

#define WARRIOR_BUTTON {150, 200, 100, 25}
#define WIZARD_BUTTON {300, 200, 100, 25}
#define CLERIC_BUTTON {450, 200, 100, 25}
#define PALADIN_BUTTON {600, 200, 100, 25}

#define HUMAN_BUTTON {150, 300, 100, 25}
#define ELF_BUTTON {300, 300, 100, 25}
#define DWARF_BUTTON {450, 300, 100, 25}
#define GNOME_BUTTON {600, 300, 100, 25}

#define LOAD_PLAYER_BUTTON {50,200,175,25}
#define CREATE_PLAYER_BUTTON {50,100,175,25}

#define MAX_TEXT_LEN 25
#define MAX_NICKNAME_LEN 13

#define MAIN_MENU_FONT_PATH "/var/Argentum/Assets/Fonts/medieval.ttf"


MainMenu::MainMenu(Texture& texture, Window& window) : window(window),
mainMenuFont(MAIN_MENU_FONT_PATH, 25),
text(mainMenuFont, window.getRenderer()),
hostInputText(mainMenuFont, window.getRenderer()),
portInputText(mainMenuFont, window.getRenderer()) ,
nicknameInputText(mainMenuFont, window.getRenderer()),
errorText(mainMenuFont, window.getRenderer()),
strength(mainMenuFont, window.getRenderer()),
constitution(mainMenuFont, window.getRenderer()),
intelligence(mainMenuFont, window.getRenderer()),
agility(mainMenuFont, window.getRenderer()),
mainMenuBackground(texture) {

    hostInput = false;
    portInput = false;
    nicknameInput = false;
    SDL_StartTextInput();
    info = {GameType::WARRIOR, GameType::HUMAN};
}

void MainMenu::menuScreen(bool& quit, GameInitializer& initializer, Socket& sock
et) {
    bool createPlayer = false;
    bool loadPlayer = false;
    bool success = false;
    bool goBack = false;

    while (¬success ∧ ¬quit) {
```

```cpp
        _playerSelectionScreen(quit, createPlayer, loadPlayer);//Veo si quiere h
acer load o create
        if (createPlayer) {

            do {
                goBack = false;
                _playerCreationScreen(quit, goBack);
                if (goBack) break;
                _connectScreen(quit, goBack, socket);
                if (¬quit ∧ ¬goBack)
                    _connectCreatedPlayer(initializer, socket, success);
            } while (goBack);

        } else if (loadPlayer) {

            do {
                goBack = false;
                _playerLoadScreen(quit, goBack);
                if (goBack) break;
                _connectScreen(quit, goBack, socket);
                if (¬quit ∧ ¬goBack)//xq puedo hacer quit en el run
                    _connectLoadedPlayer(initializer, socket, success);
            } while (goBack);

        }

        //Chequeo goBack porque si no me trate de conectar no tengo que cerrar e
l socket
        if (¬success ∧ ¬goBack) {
            socket.close();
        }
    }
}

/* Intenta conectarse al host/port que ingresa el usuario */
void MainMenu::_connectScreen(bool& quit, bool& goBack, Socket& socket) {
    errorText.updateText("");
    bool finished = quit;
    SDL_Event e;
    while (¬finished) {
        while (SDL_PollEvent(&e) ≠ 0){

            if (e.type ≡ SDL_QUIT){
                quit = true;
                finished = true;
            }

            window.handleEvent(e);
            if (e.type ≡ SDL_MOUSEBUTTONDOWN){
                int x = 0, y = 0;
                SDL_GetMouseState( &x, &y );
                x = (float)x * ((float)DEFAULT_SCREEN_WIDTH / (float)window.getW
idth());
                y = (float)y * ((float)DEFAULT_SCREEN_HEIGHT / (float)window.get
Height());
                if (_isInsideRect(x, y, INPUT_HOST_BOX)) {
                    hostInput = true;
                    portInput = false;
                } else if (_isInsideRect(x, y, INPUT_PORT_BOX)) {
                    hostInput = false;
                    portInput = true;
                } else if (_isInsideRect(x,y,CONNECT_BUTTON)) {
                    _attemptToConnect(socket, finished);
                } else if (_isInsideRect(x,y,BACK_BUTTON)) {
                    goBack = true;
                    finished = true;
                }
            } else if (e.type ≡ SDL_TEXTINPUT){
                _handleTextInput(e);
            } else if (e.type ≡ SDL_KEYDOWN) {
                if (e.key.keysym.sym ≡ SDLK_BACKSPACE) {
                    _handleBackspace();
                }
            }
        }

        _renderConnectScreen();
    }
    hostInput = false;
    portInput = false;
    errorText.updateText("");
}

/* Chequea si el usuario quiere cargar un jugador o crear uno nuevo */
void MainMenu::_playerSelectionScreen(bool& quit, bool& createPlayer, bool& load
Player) {
    bool finished = quit;
    SDL_Event e;
    while (¬finished) {
        while (SDL_PollEvent(&e) ≠ 0) {

            if (e.type ≡ SDL_QUIT){
                quit = true;
                finished = true;
            }
            window.handleEvent(e);
            if (e.type ≡ SDL_MOUSEBUTTONDOWN) {
                int x = 0, y = 0;
                SDL_GetMouseState(&x, &y);
                x = (float) x * ((float) DEFAULT_SCREEN_WIDTH / (float) window.g
etWidth());
                y = (float) y * ((float) DEFAULT_SCREEN_HEIGHT / (float) window.
getHeight());
                if (_isInsideRect(x, y, EXIT_BUTTON)) {
                    quit = true;
                    finished = true;
                } else if (_isInsideRect(x, y, CREATE_PLAYER_BUTTON)) {
                    createPlayer = true;
                    loadPlayer = false;
                    finished = true;
                } else if (_isInsideRect(x, y, LOAD_PLAYER_BUTTON)) {
                    loadPlayer = true;
                    createPlayer = false;
                    finished = true;
                }
            }
        }

        _renderPlayerSelectionScreen();
    }
}

/* Permite al usuario elegir el nickname del player que quiere cargar */
void MainMenu::_playerLoadScreen(bool &quit, bool& goBack) {
    errorText.updateText("");
    bool finished = quit;
    SDL_Event e;
    while (¬finished){
        while (SDL_PollEvent(&e) ≠ 0){
            if (e.type ≡ SDL_QUIT){
                quit = true;
                finished = true;
```

```
            }
            //Por si hago resize
            window.handleEvent(e);
            if (e.type ≡ SDL_MOUSEBUTTONDOWN){
                int x = 0, y = 0;
                SDL_GetMouseState( &x, &y );
                x = (float)x * ((float)DEFAULT_SCREEN_WIDTH/(float)window.getWid
th());
                y = (float)y * ((float)DEFAULT_SCREEN_HEIGHT/(float)window.getHe
ight());
                if (_isInsideRect(x, y, INPUT_NICKNAME_BOX)){
                    nicknameInput = true;
                } else if (_isInsideRect(x,y,BACK_BUTTON)) {
                    goBack = true;
                    finished = true;
                } else if (_isInsideRect(x,y,START_BUTTON)) {
                    if (¬nicknameInputText.getText().empty()) {
                        finished = true;
                    } else {
                        errorText.updateText("Nickname is empty");
                    }
                }
            } else if (e.type ≡ SDL_TEXTINPUT){
                _handleTextInput(e);
            } else if (e.type ≡ SDL_KEYDOWN) {
                if (e.key.keysym.sym ≡ SDLK_BACKSPACE) {
                    _handleBackspace();
                }
            }
        }
        _renderLoadPlayerScreen();
    }
    nicknameInput = false;
}

/* Permite al usuario elegir los datos del jugador que quiere crear */
void MainMenu::_playerCreationScreen(bool &quit, bool &goBack) {
    errorText.updateText("");
    bool finished = quit;
    SDL_Event e;
    while (¬finished){
        while (SDL_PollEvent(&e) ≠ 0){
            if (e.type ≡ SDL_QUIT){
                quit = true;
                finished = true;
            }
            //Por si hago resize
            window.handleEvent(e);
            if (e.type ≡ SDL_MOUSEBUTTONDOWN){
                int x = 0, y = 0;
                SDL_GetMouseState( &x, &y );
                x = (float)x * ((float)DEFAULT_SCREEN_WIDTH/(float)window.getWid
th());
                y = (float)y * ((float)DEFAULT_SCREEN_HEIGHT/(float)window.getHe
ight());
                if (_isInsideRect(x, y, INPUT_NICKNAME_BOX)){
                    nicknameInput = true;
                } else if (_isInsideRect(x,y,BACK_BUTTON)) {
                    goBack = true;
                    finished = true;
                } else if (_isInsideRect(x,y,START_BUTTON)) {
                    if (¬nicknameInputText.getText().empty()) {
                        finished = true;
                    } else {
                        errorText.updateText("Nickname is empty");
                    }
```

```cpp
                } else {
                    _verifyClassSelection(x, y);
                    _verifyRaceSelection(x, y);
                }
            } else if (e.type ≡ SDL_TEXTINPUT){
                _handleTextInput(e);
            } else if (e.type ≡ SDL_KEYDOWN) {
                if (e.key.keysym.sym ≡ SDLK_BACKSPACE) {
                    _handleBackspace();
                }
            }
        }
        _renderCreatePlayerScreen();
    }
    nicknameInput = false;
}

/* Verifica si se hizo click en alguna clase */
void MainMenu::_verifyClassSelection(int x, int y){
    if (_isInsideRect(x,y,WARRIOR_BUTTON)) info.myClass = GameType::WARRIOR;
    else if (_isInsideRect(x,y,WIZARD_BUTTON)) info.myClass = GameType::WIZARD;
    else if (_isInsideRect(x,y,CLERIC_BUTTON)) info.myClass = GameType::CLERIC;
    else if (_isInsideRect(x,y,PALADIN_BUTTON)) info.myClass = GameType::PALADIN
;
}

/* Verifica si se hizo click en alguna raza */
void MainMenu::_verifyRaceSelection(int x, int y) {
    if (_isInsideRect(x,y,HUMAN_BUTTON)) info.myRace = GameType::HUMAN;
    else if (_isInsideRect(x,y,ELF_BUTTON)) info.myRace = GameType::ELF;
    else if (_isInsideRect(x,y,DWARF_BUTTON)) info.myRace = GameType::DWARF;
    else if (_isInsideRect(x,y,GNOME_BUTTON)) info.myRace = GameType::GNOME;
}

/* Intenta conectarse al servidor con el player que se quiere crear */
void MainMenu::_connectCreatedPlayer(GameInitializer& initializer, Socket& socke
t, bool& success) {
    if (nicknameInputText.getText().find(' ') ≠ std::string::npos) {
        errorText.updateText("Nickname cannot contain spaces");
        return;
    }
    if (¬nicknameInputText.getText().empty()) {
        initializer.createPlayer(nicknameInputText.getText(), info.myRace,
                                 info.myClass);
        GameType::ConnectionResponse response{};
        socket.receive(reinterpret_cast<char*>(&response), sizeof(response));
        response = static_cast<GameType::ConnectionResponse>(ntohl(response));
        switch (response) {
            case GameType::ACCEPTED:
                success = true;
                break;
            case GameType::UNAVAILABLE_PLAYER:
                errorText.updateText("Nickname \"" + nicknameInputText.getText() +
                                                          "\" is already in use")
;
                break;
            case GameType::UNKOWN_SERVER_ERROR:
                errorText.updateText("Unknown Server Error");
                break;
            default:
                errorText.updateText("Unknown Error");
                break;
        }
    }
}

```

```cpp
/* Intenta conectarse al servidor con el player que se quiere cargar */
void MainMenu::_connectLoadedPlayer(GameInitializer& initializer, Socket& socket
, bool& success) {
    if (¬nicknameInputText.getText().empty()) {
        initializer.loadPlayer(nicknameInputText.getText());
        GameType::ConnectionResponse response{};
        socket.receive(reinterpret_cast<char*>(&response), sizeof(int32_t));
        response = static_cast<GameType::ConnectionResponse>(ntohl(response));
        switch (response) {
            case GameType::ACCEPTED:
                success = true;
                break;
            case GameType::INEXISTENT_PLAYER:
                errorText.updateText("Player \"" + nicknameInputText.getText()
                                                    + "\" does not exist")
;
                break;
            case GameType::UNAVAILABLE_PLAYER:
                errorText.updateText("Player \"" + nicknameInputText.getText() +
                                     "\" is already logged in");
                break;
            case GameType::UNKOWN_SERVER_ERROR:
                errorText.updateText("Unknown Server Error");
                break;
            default:
                errorText.updateText("Unknown Error");
                break;
        }
    }
}

/* Intenta establecer una conexion con el servidor */
void MainMenu::_attemptToConnect(Socket& socket, bool& finished) {
    try {
        socket.connect(hostInputText.getText(), portInputText.getText());
        finished = true;
    } catch (std::exception& e) {
        errorText.updateText("Could not connect");
    }
}

/* Chequea en donde se quiere hacer el input y lo procesa */
void MainMenu::_handleTextInput(SDL_Event& e) {
    std::string newInput = e.text.text;
    if (hostInput) {
        if (hostInputText.getTextLength() < MAX_TEXT_LEN)
            hostInputText += std::move(newInput);
    } else if (portInput) {
        if (portInputText.getTextLength() < MAX_TEXT_LEN)
            portInputText += std::move(newInput);
    } else if (nicknameInput) {
        if (nicknameInputText.getTextLength() < MAX_NICKNAME_LEN)
            nicknameInputText += std::move(newInput);
    }
}

/* Borra una letra del texto donde se hizo click */
void MainMenu::_handleBackspace() {
    if (hostInput) {
        --hostInputText;
    } else if (portInput) {
        --portInputText;
    } else if (nicknameInput) {
        --nicknameInputText;
    }
}
```

```

/* Chequea si se hizo click dentro de un rectangulo */
bool MainMenu::_isInsideRect(int x, int y, SDL_Rect rect){
    return ((x > rect.x) ∧ (x < rect.x + rect.w) ∧ (y > rect.y) ∧ (y <
    rect.y + rect.h));
}

/* Renderiza la pantalla de conexion */
void MainMenu::_renderConnectScreen(){
    window.clear();
    window.setViewport(ScreenViewport);
    mainMenuBackground.render(0,0);

    /* Outline de la text box para el input de host y port */
    SDL_Rect outlineRect = INPUT_HOST_BOX;
    SDL_SetRenderDrawColor(&window.getRenderer(), 0x00, 0x00,
                           0x00, 0xFF);
    SDL_RenderDrawRect( &window.getRenderer(), &outlineRect );
    outlineRect = INPUT_PORT_BOX;
    SDL_SetRenderDrawColor(&window.getRenderer(),  0x00, 0x00,
                           0x00, 0xFF);
    SDL_RenderDrawRect( &window.getRenderer(), &outlineRect );
    text.updateText("Host: ").operator*({0, 0, 0});
    text.render(50, 100);
    text.updateText("Port: ").operator*({0, 0, 0});
    text.render(50, 200);
    *(text.updateText("Connect"));
    text.render(1375, 875);
    *(text.updateText("Back"));
    text.render(50, 875);
    hostInputText.operator*({0, 0, 0}).render(115, 100);
    portInputText.operator*({0, 0, 0}).render(115, 200);
    (*errorText).render(650, 875);
    window.show();
}

void MainMenu::_renderPlayerSelectionScreen() {
    window.clear();
    window.setViewport(ScreenViewport);
    mainMenuBackground.render(0,0);
    text.updateText("Create Player").operator*({0, 0, 0});
    text.render(50, 100);
    text.updateText("Load Player").operator*({0, 0, 0});;
    text.render(50, 200);
    *(text.updateText("Exit"));
    text.render(50, 875);
    (*errorText).render(650, 875);
    window.show();
}

void MainMenu::_renderLoadPlayerScreen() {
    window.clear();
    window.setViewport(ScreenViewport);
    mainMenuBackground.render(0,0);
    /* Outline de la text box para el input de nickname */
    SDL_Rect outlineRect = INPUT_NICKNAME_BOX;
    SDL_SetRenderDrawColor(&window.getRenderer(), 0x00, 0x00,
                           0x00, 0xFF);
    SDL_RenderDrawRect( &window.getRenderer(), &outlineRect );
    text.updateText("Nickname: ").operator*({0, 0, 0});
    text.render(50, 100);
    *(text.updateText("Start"));
    text.render(1375, 875);
    *(text.updateText("Back"));
    text.render(50, 875);
    nicknameInputText.operator*({0, 0, 0}).render(165, 100);
```

```
    (*errorText).render(650, 875);
    window.show();
}

void MainMenu::_renderCreatePlayerScreen() {
    window.clear();
    window.setViewport(ScreenViewport);
    mainMenuBackground.render(0,0);
    /* Outline de la text box para el input de nickname */
    SDL_Rect outlineRect = INPUT_NICKNAME_BOX;
    SDL_SetRenderDrawColor(&window.getRenderer(),  0x00, 0x00,
                           0x00, 0xFF);
    SDL_RenderDrawRect( &window.getRenderer(), &outlineRect );
    text.updateText("Nickname: ").operator*({0, 0, 0});
    text.render(50, 100);
    (strength.updateText("Strength")).operator*({0x00,0x00,0x00});
    (constitution.updateText("Constitution")).operator*({0x00,0x00,0x00});
    (intelligence.updateText("Intelligence")).operator*({0x00,0x00,0x00});
    (agility.updateText("Agility")).operator*({0x00,0x00,0x00});
    _renderClass();
    _renderRace();
    strength.render(125, 400);
    constitution.render(125, 500);
    intelligence.render(375, 400);
    agility.render(375, 500);
    nicknameInputText.operator*({0, 0, 0}).render(165, 100);
    (*errorText).render(650, 875);
    *(text.updateText("Start"));
    text.render(1375, 875);
    *(text.updateText("Back"));
    text.render(50, 875);
    window.show();
}

void MainMenu::_renderClass() {
    text.updateText("Class: ").operator*({0x00,0x00,0x00});
    text.render(50, 200);
    text.updateText("Warrior").operator*({0x00,0x00,0x00});
    text.render(150, 200);
    text.updateText("Wizard").operator*({0x00,0x00,0x00});
    text.render(300, 200);
    text.updateText("Cleric").operator*({0x00,0x00,0x00});
    text.render(450, 200);
    text.updateText("Paladin").operator*({0x00,0x00,0x00});
    text.render(600, 200);
    /* Outline de la clase que tengo seleccionada */
    SDL_Rect outlineRect;
    switch (info.myClass) {
        case GameType::WARRIOR:
            outlineRect = WARRIOR_BUTTON;
            _updateWarriorSkills();
            break;
        case GameType::WIZARD:
            outlineRect = WIZARD_BUTTON;
            _updateWizardSkills();
            break;
        case GameType::CLERIC:
            outlineRect = CLERIC_BUTTON;
            _updateClericSkills();
            break;
        case GameType::PALADIN:
            outlineRect = PALADIN_BUTTON;
            _updatePaladinSkills();
            break;
    }
    SDL_SetRenderDrawColor(&window.getRenderer(), 0x3f, 0x2a,
                              0x14, 0xFF);
    SDL_RenderDrawRect( &window.getRenderer(), &outlineRect );
}

void MainMenu::_renderRace() {
    text.updateText("Race: ").operator*({0x00,0x00,0x00});
    text.render(50, 300);
    text.updateText("Human").operator*({0x00,0x00,0x00});
    text.render(150, 300);
    text.updateText("Elf").operator*({0x00,0x00,0x00});
    text.render(300, 300);
    text.updateText("Dwarf").operator*({0x00,0x00,0x00});
    text.render(450, 300);
    text.updateText("Gnome").operator*({0x00,0x00,0x00});
    text.render(600, 300);
    /* Outline de la raza que tengo seleccionada */
    SDL_Rect outlineRect;
    switch (info.myRace) {
        case GameType::HUMAN:
            outlineRect = HUMAN_BUTTON;
            _updateHumanSkills();
            break;
        case GameType::ELF:
            outlineRect = ELF_BUTTON;
            _updateElfSkills();
            break;
        case GameType::DWARF:
            outlineRect = DWARF_BUTTON;
            _updateDwarfSkills();
            break;
        case GameType::GNOME:
            outlineRect = GNOME_BUTTON;
            _updateGnomeSkills();
            break;
    }
    SDL_SetRenderDrawColor(&window.getRenderer(), 0x3f, 0x2a,
                              0x14, 0xFF);
    SDL_RenderDrawRect( &window.getRenderer(), &outlineRect );
}

void MainMenu::_updateWarriorSkills() {
    (strength += "++++").operator*({0, 0, 0});
    (constitution += "+++").operator*({0, 0, 0});
}

void MainMenu::_updateWizardSkills() {
    (intelligence += "+++++").operator*({0, 0, 0});
}

void MainMenu::_updateClericSkills() {
    (strength += "++").operator*({0, 0, 0});
    (constitution += "++").operator*({0, 0, 0});
    (intelligence += "+++").operator*({0, 0, 0});
}

void MainMenu::_updatePaladinSkills() {
    (strength += "+++").operator*({0, 0, 0});
    (constitution += "+++").operator*({0, 0, 0});
    (intelligence += "+").operator*({0, 0, 0});
}

void MainMenu::_updateHumanSkills() {
    (strength += "++").operator*({0, 0, 0});
    (constitution += "++").operator*({0, 0, 0});
    (intelligence += "++").operator*({0, 0, 0});
    (agility += "++").operator*({0, 0, 0});
```

```
}

void MainMenu::_updateElfSkills() {
    (intelligence += "++++").operator*({0, 0, 0});
    (agility += "+++").operator*({0, 0, 0});
}

void MainMenu::_updateDwarfSkills() {
    (strength += "++++").operator*({0, 0, 0});
    (constitution += "+++").operator*({0, 0, 0});
}

void MainMenu::_updateGnomeSkills() {
    (strength += "++").operator*({0, 0, 0});
    (constitution += "+++").operator*({0, 0, 0});
    (intelligence += "++").operator*({0, 0, 0});
}

MainMenu::~MainMenu(){
    SDL_StopTextInput();
}
```

```
//
// Created by marcos on 6/13/20.
//

#ifndef ARGENTUM_SPELL_H
#define ARGENTUM_SPELL_H

#include "../Texture/Texture.h"

class Spell {
private:
    Texture& sTexture;
    SDL_Rect& camera;
    float timePassed;
    int currentFrame;
    float xPosition, width;
    float yPosition, height;
    bool finished{false};

public:
    Spell(Texture& texture, SDL_Rect& camera, float x, float y);

    /*Renderiza el hechizo (si fuera visible)*/
    void render();

    /*Actualiza el frame de la animacion del hechizo*/
    void updateFrame(double timeStep);

    /*Retorna true si el hechizo completo su animacion asi lo borro*/
    bool finishedAnimation() const;

    /*Cambia la posicion, lo utilizo para que vaya siguiendo al entity*/
    void setPosition(float x, float y);
};


#endif //ARGENTUM_SPELL_H
```

```
//
// Created by marcos on 6/13/20.
//

#include "Spell.h"
#include "../Client/GameConstants.h"
#include "../Miscellaneous/CameraCollisionVerifier.h"

const float ANIMATION_TIME = 20000.f;
const int SPELL_SPEED = 30;

Spell::Spell(Texture& texture, SDL_Rect &camera, float x, float y) :
                                sTexture(texture), camera(camera) {
    currentFrame = 0;
    timePassed = 0;
    xPosition = x;
    yPosition = y;
    width = (float)TILE_WIDTH/2;
    height = (float)TILE_HEIGHT/2 + 15;
}

void Spell::updateFrame(double timeStep) {
    //Calculo time step
    float offset = SPELL_SPEED*timeStep;
    if ( (timePassed + offset) ≥ ANIMATION_TIME) {
        timePassed = ANIMATION_TIME;
    } else {
        timePassed += offset;
    }
    if (timePassed ≥ ANIMATION_TIME) {
        currentFrame = 0;
        timePassed = 0;
        finished = true;
    } else {
        for (int i = 0; i < 24; ++i) { /*6 es la cantidad de frames distintos de
l spell*/
            if (timePassed < ((float)ANIMATION_TIME/24 * (float)(i+1))) {
                currentFrame = i;
                break;
            }
        }
    }
}

void Spell::render() {
    if (CameraCollisionVerifier::isInsideCamera(camera, {(int)xPosition,
                                                          (int)yPosition, (int)wi
dth, (int)height})) {
        sTexture.render((int)(xPosition) − camera.x,
                                (int)(yPosition) − camera.y, currentFrame);
    };
}

bool Spell::finishedAnimation() const {
    return finished;
}

void Spell::setPosition(float x, float y) {
    xPosition = x;
    yPosition = y;
}
```

```cpp
//
// Created by marcos on 19/7/20.
//

#ifndef ARGENTUM_CAMERACOLLISIONVERIFIER_H
#define ARGENTUM_CAMERACOLLISIONVERIFIER_H

#include <SDL_rect.h>

class CameraCollisionVerifier {
public:
    static bool isInsideCamera(SDL_Rect a, SDL_Rect b, int adjustment);
    static bool isInsideCamera(SDL_Rect a, SDL_Rect b);
};


#endif //ARGENTUM_CAMERACOLLISIONVERIFIER_H
```

```cpp
//
// Created by marcos on 19/7/20.
//

#include "CameraCollisionVerifier.h"

/*Verifica si el se encuentra adentro de la camara (para renderizarlo solo
 * si hace falta). Adjustment es para las estructuras ya que ocupan mas de un
 * tile y las dimensiones son ligeramentes distintas*/

/*Este se usa para las estructuras*/
bool CameraCollisionVerifier::isInsideCamera(SDL_Rect a, SDL_Rect b, int adjustm
ent) {
    int leftA, leftB;
    int rightA, rightB;
    int topA, topB;
    int bottomA, bottomB;
    //Calculo los lados de A
    leftA = a.x;
    rightA = a.x + a.w;
    topA = a.y;
    bottomA = a.y + a.h;

    //Calculo los lados de B
    leftB = b.x;
    rightB = b.x + b.w + adjustment;
    topB = b.y  + adjustment − b.h;
    bottomB = b.y + adjustment; /*Porque centro las estructuras en el medio del
tile*/

    //Si alguno de los lados de A esta fuera de B
    if(bottomA ≤ topB) return false;
    if(topA ≥ bottomB) return false;
    if(rightA ≤ leftB) return false;
    if(leftA ≥ rightB) return false;

    //Si ningun lado de A esta fuera de B
    return true;
}

/*Este se usa para las que no son estructuras*/
bool CameraCollisionVerifier::isInsideCamera(SDL_Rect a, SDL_Rect b) {
    int leftA, leftB;
    int rightA, rightB;
    int topA, topB;
    int bottomA, bottomB;
    //Calculo los lados de A
    leftA = a.x;
    rightA = a.x + a.w;
    topA = a.y;
    bottomA = a.y + a.h;

    //Calculo los lados de A
    leftB = b.x;
    rightB = b.x + b.w;
    topB = b.y;
    bottomB = b.y + b.h;

    if(bottomA ≤ topB) return false;
    if(topA ≥ bottomB) return false;
    if(rightA ≤ leftB) return false;
    if(leftA ≥ rightB) return false;

    return true;
}
```

```
//
// Created by marcos on 14/7/20.
//

#ifndef ARGENTUM_ARROW_H
#define ARGENTUM_ARROW_H

#include "../Texture/Texture.h"

/*Esta clase encapsula el comportamiento de una flecha*/

class Arrow {
private:
    Texture& sTexture;
    SDL_Rect& camera;
    float angle, distanceToTravel;
    float currDistance;
    float xPosition, width;
    float yPosition, height;
    bool finished{false};

public:
    Arrow(Texture& texture, SDL_Rect& camera, float x, float y,
                                          float xTarget, float yTarget);

    /*Renderiza la flecha (si fuera visible)*/
    void render();

    /*Actualiza la posicion de la flecha*/
    void updatePosition(double timeStep);

    /*Devuelve true si la flecha alcanzo su objetivo, asi puedo borrarla*/
    bool reachedTarget() const;

private:
    void _calculateTrajectory(float xTarget, float yTarget);
};


#endif //ARGENTUM_ARROW_H
```

```
//
// Created by marcos on 14/7/20.
//

#include "Arrow.h"
#include "../Client/GameConstants.h"
#include "../Miscellaneous/CameraCollisionVerifier.h"

const int ARROW_SPEED = 2;

Arrow::Arrow(Texture& texture, SDL_Rect &camera, float xPos, float yPos,
                float xTarget, float yTarget) :
        sTexture(texture), camera(camera) {
    currDistance = 0;
    xPosition = xPos;
    yPosition = yPos;
    _calculateTrajectory(xTarget, yTarget);
    width = (float)TILE_WIDTH/2;
    height = (float)TILE_HEIGHT/2 + 15;
}

void Arrow::_calculateTrajectory(float xTarget, float yTarget) {
    float relativeXTarget = xTarget − xPosition; /*Lo llevo relativo al origen q
ue es la posicion de mi flecha*/
    float relativeYTarget = yPosition − yTarget;
    angle = atan2(relativeYTarget, relativeXTarget) * 180 / M_PI; /*Calculo el a
ngulo de la recta*/
    distanceToTravel = sqrt(pow(relativeXTarget, 2.0) + pow(relativeYTarget, 2.0
));
}

void Arrow::render() {
    if (CameraCollisionVerifier::isInsideCamera(camera, {(int)xPosition, (int)yP
osition,
                                 (int)width, (int)height})) {
        sTexture.render((int)(xPosition) − camera.x,
                        (int)(yPosition) − camera.y, 0, 40 − angle);
    };
}

void Arrow::updatePosition(double timeStep) {
    if (¬finished) {
        float moved = ARROW_SPEED * timeStep;
        xPosition += moved * cos(angle * M_PI / 180);
        yPosition −= moved * sin(angle * M_PI / 180);
        currDistance += moved;
        if (currDistance ≥ distanceToTravel) {
            finished = true;
        }
    }
}

bool Arrow::reachedTarget() const {
    return finished;
}
```

```cpp
//
// Created by marcos on 6/7/20.
//

#ifndef ARGENTUM_TILE_H
#define ARGENTUM_TILE_H

#include "../Texture/TextureRepository.h"
#include "ItemDrop.h"
#include "Structure.h"
#include "../Character/Entity.h"
#include "Coordinate.h"
#include <memory>
#include "../../libs/GameEnums.h"
#include <list>
#include "../Miscellaneous/Spell.h"

class Tile {
private:
    SDL_Rect box{};
    Texture* tileTexture{nullptr};
    ItemDrop item;
    Structure structure;
    std::weak_ptr<Entity> entity;
    std::weak_ptr<Spell> spell;
    int type; //La mayoria de las texturas de los tiles son varias en una
              //con esto puedo especificar cual quiero en particular

public:
    explicit Tile(Coordinate position);

    /*Carga la data inicial del tile, esto es, el tipo de piso y si guarda una estructura*/
    void loadData(Texture& _tileTexture, Texture* sTexture = nullptr, int tileType = 0);

    /*Crea un item en el tile*/
    void createItem(Texture& _itemTexture);

    /*Renderiza el piso del tile*/
    void renderGround(SDL_Rect& camera);

    /*Renderiza la estructura (si tuviera)*/
    void renderStructure(SDL_Rect& camera);

    /*Renderiza la entity (si tuviera)*/
    void renderEntity();

    /*Agrega un entity al tile*/
    void addEntity(std::shared_ptr<Entity>& _entity);

    /*Elimina el entity del tile*/
    void removeEntity();

    /*Agrega un spell al tile*/
    void addSpell(std::shared_ptr<Spell>& newSpell, SDL_Rect& camera);

    /*Elimina el spell del tile*/
    void destroyItem();

    /*Pasa el spell de entity (si tuviera) al tile. Esto es por si el entity
     * muere, para no perder la animacion*/
    void retrieveEntitySpell();
};


```

```cpp
#endif //ARGENTUM_TILE_H
```

```cpp
//
// Created by marcos on 6/7/20.
//

#include "Tile.h"
#include "../Client/GameConstants.h"

Tile::Tile(Coordinate position) : item(position), structure(position) {
    box = {position.j*TILE_WIDTH, position.i*TILE_HEIGHT, TILE_WIDTH, TILE_HEIGH
T};
    type = 0;
}

void Tile::renderGround(SDL_Rect& camera) {
    tileTexture->render(box.x - camera.x, box.y - camera.y, type);
    item.render(camera);
}

void Tile::renderStructure(SDL_Rect &camera) {
    structure.render(camera);
}

void Tile::loadData(Texture& _tileTexture, Texture* sTexture, int tileType) {
    type = tileType;
    tileTexture = &_tileTexture;
    if (sTexture) {
        structure.setTexture(*sTexture);
    }
}

void Tile::createItem(Texture& _itemTexture) {
    item.setItem(&_itemTexture);
}

void Tile::addEntity(std::shared_ptr<Entity>& _entity) {
    entity = _entity;
}

void Tile::renderEntity() {
    auto _entity = entity.lock();
    if (_entity) {
        _entity->render();
    }
    auto _spell = spell.lock();
    if (_spell) {
        _spell->render();
    }
}

void Tile::removeEntity() {
    entity.reset();
}

void Tile::addSpell(std::shared_ptr<Spell>& newSpell, SDL_Rect& camera) {
    auto _entity = entity.lock();
    if (_entity) {
        _entity->addSpell(newSpell);
    } else {
        spell = newSpell;
    }
}

void Tile::destroyItem() {
    item.setItem(nullptr);
}

```

```cpp
void Tile::retrieveEntitySpell() {
    auto _entity = entity.lock();
    if (_entity) {
        spell = _entity->getSpell();
    }
}
```

```
//
// Created by marcos on 10/6/20.
//

#ifndef ARGENTUM_STRUCTURE_H
#define ARGENTUM_STRUCTURE_H

#include "../Texture/Texture.h"
#include "Coordinate.h"

/*Esta clase representa una estructura en el juego, esto puede variar
 * desde casas hasta arboles/arbustos o cadaveres*/

class Structure {
private:
    SDL_Rect box{};
    Texture* sTexture;

public:
    explicit Structure(Coordinate position, Texture* sTexture = nullptr);

    /*Setea la textura a renderizar de la estructura*/
    void setTexture(Texture& texture);

    /*Renderiza la estructura si es visible en la camara del player*/
    void render(SDL_Rect& camera);
};


#endif //ARGENTUM_STRUCTURE_H
```

```
//
// Created by marcos on 10/6/20.
//

#include "Structure.h"
#include "../Client/GameConstants.h"
#include "../Miscellaneous/CameraCollisionVerifier.h"

Structure::Structure(Coordinate position, Texture* sTexture) : sTexture(sTexture
) {
    if (sTexture ≠ nullptr) {
        SpriteDimensions_t dimensions = sTexture→getSpriteDimensions();
        box = {position.j*TILE_WIDTH, position.i*TILE_HEIGHT, dimensions.width,
dimensions.height};
    } else {
        box = {position.j*TILE_WIDTH, position.i*TILE_HEIGHT, 0, 0};
    }
}

void Structure::render(SDL_Rect& camera) {
    //Si se ve el tile en la pantalla
    if (sTexture ≠ nullptr ∧ CameraCollisionVerifier::isInsideCamera(camera, box
, TILE_HEIGHT/2)) {
        sTexture→render(box.x − camera.x, box.y − camera.y);
    }
}

void Structure::setTexture(Texture& texture) {
    sTexture = &texture;
    SpriteDimensions_t dimensions = sTexture→getSpriteDimensions();
    box.w = dimensions.width;
    box.h = dimensions.height;
}
```

```
//
// Created by marcos on 6/7/20.
//

#ifndef ARGENTUM_MAP_H
#define ARGENTUM_MAP_H

#include "Tile.h"
#include <vector>
#include "../Texture/TextureRepository.h"
#include "Structure.h"
#include "../Client/ProtocolEnumTranslator.h"
#include "../Texture/PlayerEquipment.h"
#include <list>
#include "../Client/EntityData.h"
#include "../Miscellaneous/Spell.h"
#include "../Client/CitizenData.h"
#include "../Sound/SoundPlayer.h"
#include "../Miscellaneous/Arrow.h"

/*Esta clase representa al mapa del juego y tiene el ownership de las entidades,
 * flechas y hechizos en juego*/

class Map {
private:
    TextureRepository& textureRepo;
    SoundPlayer& soundPlayer;
    std::vector<Tile> tiles;
    SDL_Rect& camera;
    std::unordered_map<std::string, std::pair<std::shared_ptr<Entity>, Coordinat
e>> entities;
    std::list<std::tuple<std::shared_ptr<Entity>*, Coordinate, std::string>> ent
itiesToUpdateTilePosition;
    /*Esto es para no pisar entities entre si cuando terminan de moverse*/

    std::list<std::shared_ptr<Spell>> spells;
    std::list<std::unique_ptr<Arrow>> arrows;
    std::string playerNickname;

public:
    Map(TextureRepository& repo, SDL_Rect& camera, SoundPlayer& soundPlayer);

    /*Setea el tamanio del vector unidimensional tiles*/
    void setSize(int rows, int columns);

    /*Carga la data inicial del tile position, es decir, su tipo de piso,
     * estructura (si tuviera) y citizen (si tuviera). Es lo primero que recibo
     * cuando me conecto al server, para cada tile*/
    void loadTileData(Coordinate position, FloorTypeTexture floor, TextureID str
ucture,
                      CitizenData& citizen);

    /*Crea el item con la textura pedida en el tile de coordenada position*/
    void createItem(Coordinate position, TextureID itemTexture);

    /*Crea un npc en el mapa en base a la data provista (textura, posicion, etc)
*/
    void addNPC(EntityData& data);

    /*Crea un player en el mapa en base a la data provista (textura, posicion, e
tc)*/
    void addPlayer(MapPlayerData& playerData);

    /*Desplaza a la entity en la direccion recibida la distanceTravelled indicad
a,
     * , si termino de moverse (reachedDestination = true) le resetea la animaci
```

```
on.
     * Si es el entity estaba quieto entonces lo encola a la
     * lista de entitesToUpdateTilePosition para actualizarle su posicion*/
    void moveEntity(std::string& nickname, GameType::Direction direction,
            unsigned int distanceTravelled, bool reachedDestination);

    /*Centra la camara en la entidad cuyo nombre matchee con nickname
     * (se utiliza para el propio player)*/
    void setCameraOn(std::string& nickname);

    /*Borra del mapa a la entity*/
    void removeEntity(std::string& nickname);

    /*Equipa en el EquipmentPlace indicado el equipment recibido, se usa para
     * cambiar lo que tienen equipados los players)*/
    void equipOnPlayer(std::string& nickname, GameType::EquipmentPlace place,
                       TextureID equipment);

    /*Setea la texture del player en el fantasma*/
    void killPlayer(std::string& nickname);

    /*Setea la texture del player en la que tiene cuando esta vivo, es decir,
     * si tiene la del fantasma se la quita*/
    void revivePlayer(std::string &nickname);

    /*Agrego un spell en la posicion indicada. Si hubiera un player en dicha
     * posicion se lo agrega a el, sino lo agrega al tile*/
    void addSpell(Coordinate position, TextureID spellTexture);

    /*Agrega un arrow a la lista de arrows, seteando su posicion inicial en
     * la del arquero, dicha arrow se movera al target a medida que pase el tiem
po*/
    void addArrow(std::string& archerNickname, Coordinate target, TextureID arro
wTexture);

    /*Actualiza las animaciones del juego en base al timeStep y mueve a los enti
ties
     * a sus nuevos tiles correspondientes*/
    void update(double timeStep = 0);

    /*Elimina el item en la posicion indicada*/
    void destroyItem(Coordinate itemPosition);

    /*Teletransporta un entity de una posicion a otra. Se usa para cuando
     * en el juego un player revive en una ciudad*/
    void teleportEntity(const std::string& nickname, Coordinate newPosition,
                        bool isMyPlayer);

    /*Setea el nickname del player propio, util para poder referenciarlo y saber
     * que sonidos realizar (lo buscamos en el map y conseguimos su ubicacion
     * en el mapa)*/
    void setPlayerNickname(const std::string& nickname);

    /*Verifica si el sonido a realizar esta dentro del rango de sonido del playe
r
     * propio*/
    void verifyQueueSound(Coordinate tile, SoundID sound, int maxDistance);

    /*Cambia la direccion de renderizado del entity, se utiliza en el evento de
un ataque
     * para que mire a su target*/
    void changeEntityLookDirection(std::string& nickname, GameType::Direction di
rection);

    /*Actualiza el nivel que se muestra del player*/
    void updatePlayerLevel(const std::string& _playerNickname, int level);
    /*Renderiza el mapa, esto incluye entities, estructuras, spells, etc*/
    void render();

private:
    static Coordinate _calculateNewTile(Coordinate position, GameType::Direction
 direction);
    void _updateSpells(double timeStep);
    void _updateArrows(double timeStep);
    void _moveEntitiesToNewTile();
    void _addEntity(EntityData& data, std::shared_ptr<Entity>& entity);
    void _renderGround();
    void _renderStructures();
    void _renderNPCS();
    void _renderArrows();
};


#endif //ARGENTUM_MAP_H
```

```
//
// Created by marcos on 6/7/20.
//

#include "Map.h"
#include <fstream>
#include <algorithm>
#include "../Client/GameConstants.h"
#include "../Character/NPC.h"
#include "../Character/Player.h"

Map::Map(TextureRepository& repo, SDL_Rect& camera, SoundPlayer& soundPlayer) :
                    textureRepo(repo), soundPlayer(soundPlayer),camera(camera){
    this→camera = camera;
}

void Map::_renderGround() {
    for (int i = 0; i < (VISIBLE_VERTICAL_TILES + 1); ++i) {
        for (int j = 0; j < (VISIBLE_HORIZONTAL_TILES + 1); ++j) {
            float x = (float)camera.x + (float)j * TILE_WIDTH;
            float y = (float)camera.y + (float)i * TILE_HEIGHT;
            if (x ≥ LEVEL_WIDTH ∨ x < 0) continue;
            if (y ≥ LEVEL_HEIGHT ∨ y < 0) continue;
            int xTile = floor(x / TILE_WIDTH);
            int yTile = floor(y / TILE_HEIGHT);
            int tile = yTile*TOTAL_HORIZONTAL_TILES + xTile;
            tiles.at(tile).renderGround(camera);
        }
    }
}

void Map::_renderStructures() {
    for (int i = −1; i < (VISIBLE_VERTICAL_TILES + 3); ++i) {
        for (int j = −1; j < (VISIBLE_HORIZONTAL_TILES + 3); ++j) {
            float x = (float)camera.x + (float)j * TILE_WIDTH;
            float y = (float)camera.y + (float)i * TILE_HEIGHT;
            if (x ≥ LEVEL_WIDTH ∨ x < 0) continue;
            if (y ≥ LEVEL_HEIGHT ∨ y < 0) continue;
            int xTile = floor(x / TILE_WIDTH);
            int yTile = floor(y / TILE_HEIGHT);
            int tile = yTile*TOTAL_HORIZONTAL_TILES + xTile;
            tiles.at(tile).renderStructure(camera);
        }
    }
}

void Map::_renderNPCS() {
    for (int i = −2; i < (VISIBLE_VERTICAL_TILES + 2); ++i) {
        for (int j = −2; j < (VISIBLE_HORIZONTAL_TILES + 2); ++j) {
            float x = (float)camera.x + (float)j * TILE_WIDTH;
            float y = (float)camera.y + (float)i * TILE_HEIGHT;
            if (x ≥ LEVEL_WIDTH ∨ x < 0) continue;
            if (y ≥ LEVEL_HEIGHT ∨ y < 0) continue;
            int xTile = floor(x / TILE_WIDTH);
            int yTile = floor(y / TILE_HEIGHT);
            int tile = yTile*TOTAL_HORIZONTAL_TILES + xTile;
            tiles.at(tile).renderEntity();
        }
    }
}

void Map::setSize(int rows, int columns) {
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < columns; ++j) {
            tiles.emplace_back(Coordinate{i, j});
        }
    }
}

void Map::loadTileData(Coordinate position, FloorTypeTexture floor, TextureID structure,
                       CitizenData& citizen) {
    unsigned int tile = position.i*TOTAL_HORIZONTAL_TILES + position.j;
    if (structure ≠ Nothing) {
        tiles.at(tile).loadData(textureRepo.getTexture(floor.texture),
                &textureRepo.getTexture(structure), floor.index);
    } else {
        tiles.at(tile).loadData(textureRepo.getTexture(floor.texture),
                nullptr, floor.index);
    }
    if (citizen.texture ≠ Nothing) {
        EntityData data = {citizen.texture, std::move(citizen.nickname), position,
                           GameType::DIRECTION_STILL, 0};
        addNPC(data);
    }
}

void Map::_addEntity(EntityData& data, std::shared_ptr<Entity>& entity) {
    if (data.currentDir ≠ GameType::DIRECTION_STILL) {
        Coordinate destination = _calculateNewTile(data.pos,
                                                   data.currentDir);
        entity→move(data.currentDir, data.distanceMoved, false);
        entities.emplace(data.nickname, std::make_pair(std::move(entity),
                                                       data.pos));
        entitiesToUpdateTilePosition.emplace_back(&entities.at(data.nickname).first,
                                                  destination, data.nickname);
    } else {
        int tile = data.pos.i*TOTAL_HORIZONTAL_TILES + data.pos.j;
        tiles.at(tile).addEntity(entity);
        entities.emplace(data.nickname, std::make_pair(std::move(entity),
                                                       data.pos));
    }
}

void Map::addNPC(EntityData& data) {
    if (entities.count(data.nickname) ≡ 0) {
        std::string npcLevel;
        if (data.level > 0) {
            npcLevel = std::to_string(data.level);
        }
        std::shared_ptr<Entity> npc = std::make_shared<NPC>(textureRepo,
                camera, data.pos.j*TILE_WIDTH,data.pos.i*TILE_HEIGHT, data.texture,
                std::move(npcLevel));
        _addEntity(data, npc);
    }
}

void Map::addPlayer(MapPlayerData& playerData) {
    if (entities.count(playerData.entityData.nickname) ≡ 0) {
        std::shared_ptr<Entity> player = std::make_shared<Player>(textureRepo,camera,
                playerData.entityData.pos.j*TILE_WIDTH,playerData.entityData.pos.i*TILE_HEIGHT,
                playerData.equipment, playerData.isAlive, std::to_string(playerData.entityData.level),
                        playerData.entityData.nickname);
        _addEntity(playerData.entityData, player);
    }
}
```

```

void Map::createItem(Coordinate position, TextureID itemTexture) {
    unsigned int tile = position.i*TOTAL_HORIZONTAL_TILES + position.j;
    tiles.at(tile).createItem(textureRepo.getTexture(itemTexture));
}

void Map::moveEntity(std::string &nickname, GameType::Direction direction,
                     unsigned int distanceTravelled, bool reachedDestination) {

    if (entities.count(nickname) ≡ 1) { /*Si no lo mataron en el update*/
        Entity* entity = entities.at(nickname).first.get();
        GameType::Direction previousDirection = entity→move(direction,
                                                distanceTravelled, reachedDestination);
        if (previousDirection ≡ GameType::DIRECTION_STILL) { /*Se empezo a mover de tile*/
            Coordinate oldPosition = entities.at(nickname).second;
            verifyQueueSound(oldPosition, StepSound, 3);
            Coordinate newPosition = _calculateNewTile(oldPosition, direction);
            int tile = oldPosition.i*TOTAL_HORIZONTAL_TILES + oldPosition.j;
            tiles.at(tile).removeEntity();
            entitiesToUpdateTilePosition.emplace_back(&entities.at(nickname).first,
                                                    newPosition, std::move(nickname));
        }
    }
}

void Map::verifyQueueSound(Coordinate tile, SoundID sound, int maxDistance) {
    Coordinate playerPos = entities.at(playerNickname).second;
    int distance = std::abs(playerPos.j − tile.j) + std::abs(playerPos.i − tile.i);
    if (distance ≤ maxDistance) {
        soundPlayer.queueSound(sound);
    }
}

Coordinate Map::_calculateNewTile(Coordinate position, GameType::Direction direction) {
    switch (direction) {
        case GameType::DIRECTION_UP:
            position.i −= 1;
            break;
        case GameType::DIRECTION_DOWN:
            position.i += 1;
            break;
        case GameType::DIRECTION_LEFT:
            position.j −= 1;
            break;
        case GameType::DIRECTION_RIGHT:
            position.j += 1;
            break;
        case GameType::DIRECTION_STILL:
            //do nothing
            break;
    }
    return position;
}

void Map::_moveEntitiesToNewTile() {
    if (¬entitiesToUpdateTilePosition.empty()) {
        for (auto ∧ entity : entitiesToUpdateTilePosition) {
            if (entities.count(std::get<2>(entity)) ≡ 1) { //este chequeo es para el posible caso donde lo mueven al
                                                                    // nuevo tile y matan en el mismo update
```

```
                int tile = std::get<1>(entity).i * TOTAL_HORIZONTAL_TILES + std::get<1>(entity).j;
                entities.at(std::get<2>(entity)).second = std::get<1>(entity);
                tiles.at(tile).addEntity(*std::get<0>(entity));
            }
        }
        entitiesToUpdateTilePosition.clear();
    }
}

void Map::setCameraOn(std::string& nickname) {
    entities.at(nickname).first->activateCamera();
}

void Map::removeEntity(std::string &nickname) {
    Coordinate position = entities.at(nickname).second;
    int tile = position.i * TOTAL_HORIZONTAL_TILES + position.j;
    tiles.at(tile).retrieveEntitySpell();
    tiles.at(tile).removeEntity();
    entities.erase(nickname);
}

void Map::equipOnPlayer(std::string &nickname, GameType::EquipmentPlace place,
                        TextureID equipment) {
    Entity* entity = entities.at(nickname).first.get();
    auto player = dynamic_cast<Player*>(entity);
    if (player) {
        player->equip(place, equipment);
    }
}

void Map::killPlayer(std::string &nickname) {
    Entity* entity = entities.at(nickname).first.get();
    Coordinate position = entities.at(nickname).second;
    verifyQueueSound(position, Death1Sound, 10);
    auto player = dynamic_cast<Player*>(entity);
    if (player) {
        player->kill();
    }
}

void Map::revivePlayer(std::string &nickname) {
    Entity* entity = entities.at(nickname).first.get();
    auto player = dynamic_cast<Player*>(entity);
    if (player) {
        player->revive();
    }
}

void Map::addSpell(Coordinate position, TextureID spellTexture) {
    int tile = position.i*TOTAL_HORIZONTAL_TILES + position.j;
    std::shared_ptr<Spell> spell(new Spell(textureRepo.getTexture(spellTexture),
            camera,position.j*TILE_WIDTH, position.i*TILE_HEIGHT)); //No uso make shared ya que el spell
                                                                    //no lo borro del weak ptr al destruirse
                                                                    // y con make shared eso resultaria en conservar mas memoria
    tiles.at(tile).addSpell(spell, camera);
    spells.emplace_back(std::move(spell));
}

void Map::addArrow(std::string& archerNickname, Coordinate target, TextureID arrowTexture) {
    Coordinate archerPosition = entities.at(archerNickname).second;
    arrows.emplace_back(new Arrow(textureRepo.getTexture(arrowTexture), camera,
```

```
                archerPosition.j*TILE_WIDTH, archerPosition.i*TILE_HEIGHT,
                target.j*TILE_WIDTH, target.i*TILE_HEIGHT));
}

void Map::update(double timeStep) {
    _moveEntitiesToNewTile();
    _updateSpells(timeStep);
    _updateArrows(timeStep);
}

static bool shouldSpellBeRemoved(std::shared_ptr<Spell>& spell) {
    if (spell) {
        return spell->finishedAnimation();
    }
    return true;
}

static bool shouldArrowBeRemoved(std::unique_ptr<Arrow>& arrow) {
    if (arrow) {
        return arrow->reachedTarget();
    }
    return true;
}

void Map::_updateArrows(double timeStep) {
    if (!arrows.empty()) {
        for (auto & arrow : arrows) {
            if (arrow) {
                arrow->updatePosition(timeStep);
            }
        }
        arrows.erase(std::remove_if(arrows.begin(), arrows.end(),
                                    shouldArrowBeRemoved), arrows.end());
    }
}

void Map::_updateSpells(double timeStep) {
    if (!spells.empty()) {
        for (auto & spell : spells) {
            if (spell) {
                spell->updateFrame(timeStep);
            }
        }
        spells.erase(std::remove_if(spells.begin(), spells.end(),
                                    shouldSpellBeRemoved), spells.end());
    }
}

void Map::destroyItem(Coordinate itemPosition) {
    int tile = itemPosition.i*TOTAL_HORIZONTAL_TILES + itemPosition.j;
    tiles.at(tile).destroyItem();
}

void Map::teleportEntity(const std::string &nickname, Coordinate newPosition,
                         bool isMyPlayer) {
    Entity* entity = entities.at(nickname).first.get();
    Coordinate oldPosition = entities.at(nickname).second;
    int oldTile = oldPosition.i*TOTAL_HORIZONTAL_TILES + oldPosition.j;
    tiles.at(oldTile).removeEntity();
    int newTile = newPosition.i*TOTAL_HORIZONTAL_TILES + newPosition.j;
    entity->setPosition(newPosition.j*TILE_WIDTH, newPosition.i*TILE_HEIGHT);
    if (isMyPlayer) {
        entity->activateCamera();
    }
    tiles.at(newTile).addEntity(entities.at(nickname).first);
    entities.at(nickname).second = newPosition;
```

```cpp
}

void Map::setPlayerNickname(const std::string &nickname) {
    playerNickname = nickname;
}

void Map::changeEntityLookDirection(std::string& nickname, GameType::Direction direction) {
    Entity* entity = entities.at(nickname).first.get();
    entity->setLookDirection(direction);
}

void Map::render() {
    _renderGround();
    _renderNPCS();
    _renderArrows();
    _renderStructures();
}

void Map::_renderArrows() {
    for (auto & arrow : arrows) {
        arrow->render();
    }
}

void Map::updatePlayerLevel(const std::string& _playerNickname, int level) {
    Entity* player = entities.at(_playerNickname).first.get();
    player->updateLevel(level);
}
```

```cpp
//
// Created by marcos on 6/13/20.
//

#ifndef ARGENTUM_ITEMDROP_H
#define ARGENTUM_ITEMDROP_H

#include "../Texture/Texture.h"
#include "Coordinate.h"

/*Esta clase representa la imagen particular de un item (el drop). Se renderiza
 * en el mapa cuando esta droppeado en un tile o tambien en el inventario*/

class ItemDrop {
private:
    SDL_Rect box{};
    Texture* sTexture;

public:
    explicit ItemDrop(Coordinate position);

    /*Renderiza el item*/
    void render(SDL_Rect& camera);

    /*Setea la textura del item a renderizar*/
    void setItem(Texture* itemTexture);
};


#endif //ARGENTUM_ITEMDROP_H
```

```
//
// Created by marcos on 6/13/20.
//

#include "ItemDrop.h"
#include "../Client/GameConstants.h"

ItemDrop::ItemDrop(Coordinate position) : sTexture(nullptr) {
    box = {position.j*TILE_WIDTH, position.i*TILE_HEIGHT, 0, 0};
}

void ItemDrop::render(SDL_Rect& camera) {
    if (sTexture ≠ nullptr) {
        sTexture→render(box.x − camera.x, box.y − camera.y, 0, −90);
    }
}

void ItemDrop::setItem(Texture* itemTexture) {
    if (itemTexture ≠ nullptr) {
        SpriteDimensions_t dimensions = itemTexture→getSpriteDimensions();
        box.w = dimensions.width;
        box.h = dimensions.height;
    }
    sTexture = itemTexture;
}
```

```
//
// Created by marcos on 6/27/20.
//

#ifndef ARGENTUM_COORDINATE_H
#define ARGENTUM_COORDINATE_H

struct Coordinate {
    int i;
    int j;

    bool operator≠(Coordinate& other) {
        return (i ≠ other.i ∨ j ≠ other.j);
    }
};

#endif //ARGENTUM_COORDINATE_H
```

```
//
// Created by ivan on 12/6/20.
//

#include "Font.h"
#include "../../Texture/Texture.h"

#include <string>

#ifndef ARGENTUM_TEXT_H
#define ARGENTUM_TEXT_H


class Text {
private:
    Font& font;
    std::string text;
    Texture textTexture;

public:
    Text(Font& font, SDL_Renderer& renderer, std::string∧ _text = "");

    Text(Font& font, SDL_Renderer& renderer, const std::string& _text);

    Text(Text∧ other) noexcept;

    /* Setea el texto a "newText" */
    Text& updateText(std::string∧ newText);

    /* Setea el texto a "newText" */
    Text& updateText(const std::string& newText);

    /* Agrega "newText" al final del texto */
    Text& operator+=(std::string∧ newText);

    /* Borra la ultima letra del texto */
    Text& operator−−();

    /* Me devuelve el tamaÃ±o del texto */
    int getTextLength();

    /* Me devuelve una referencia al texto */
    std::string& getText();

    /* Renderiza el texto en la posicion (x,y) con el color "color" */
    void render(int x, int y);

    /* Genera la textura en base al texto almacendo */
    Text& operator*(SDL_Color color);

    /* Es para mandarle un color default al otro, C++ no me deja ponerle default
 sino*/
    Text& operator*();

    /*Retorna el ancho en pixeles del texto, se usa para dejar lindo el nombre c
on el nivel*/
    int getTextTextureWidth();

    ~Text();
};


#endif //ARGENTUM_TEXT_H
```

```
//
// Created by ivan on 12/6/20.
//

#include "Text.h"

Text::Text(Font& font, SDL_Renderer& renderer, std::string∧ _text) : font(font)
,
                                    textTexture(renderer) {
    text = std::move(_text);
    operator*();
}

Text& Text::updateText(std::string∧ newText) {
    text = std::move(newText);
    return *this;
}

Text& Text::operator+=(std::string∧ newText) {
    text += newText;
    return *this;
}

Text& Text::operator*(SDL_Color color) {
    if (¬text.empty()) {
        textTexture.loadFromRenderedText(text, color, font.getFont());
    }
    return *this;
}

void Text::render(int x, int y) {
    if (¬text.empty()) {
        textTexture.render(x, y);
    }
}

Text& Text::operator−−() {
    if (¬text.empty()) {
        text.pop_back();
    }
    return *this;
}

int Text::getTextLength() {
    return text.length();
}

std::string &Text::getText() {
    return text;
}

Text& Text::operator*() {
    return operator*({0xFF, 0xFF, 0xFF});
}

Text &Text::updateText(const std::string &newText) {
    text = newText;
    return *this;
}

Text::Text(Font& font, SDL_Renderer& renderer, const std::string& _text) : font(
font),
                                                                        textTextur
e(renderer) {
    text = _text;
    operator*();
}

int Text::getTextTextureWidth() {
    if (¬text.empty()) {
        return textTexture.getSpriteDimensions().width;
    } else {
        return 0;
    }
}

Text::Text(Text ∧other) noexcept : font(other.font), text(std::move(other.text)
),
                                    textTexture(std::move(other.textTexture)) {}

Text::~Text() = default;
```

```
#ifndef ARGENTUM_FONT_H
#define ARGENTUM_FONT_H

#include <SDL.h>
#include <SDL_image.h>
#include <SDL_ttf.h>
#include <string>
#include <iostream>

#include "../../../libs/TPException.h"

class Font {
private:
    TTF_Font *font;
public:
    Font(const std::string& path, int fontSize);

    TTF_Font* getFont();

    ~Font();
};


#endif //ARGENTUM_FONT_H
```

```cpp
//
// Created by ivan on 7/6/20.
//

#include "Font.h"

Font::Font(const std::string& path, int fontSize) {
    //carga la font
    font = TTF_OpenFont(path.c_str(), fontSize);
    if(font ≡ nullptr ){
        throw TPException("Failed to load lazy font! SDL_ttf Error: %s\n",
                          TTF_GetError() );
    }
}

TTF_Font *Font::getFont() {
    return this→font;
}

Font::~Font() {
    TTF_CloseFont(font);
}
```

```cpp
//
// Created by ivan on 20/6/20.
//

#ifndef ARGENTUM_SELECTOR_H
#define ARGENTUM_SELECTOR_H

#include <SDL.h>
#include "../Screen/Window.h"
#include "../Map/Coordinate.h"
#include "../../libs/GameEnums.h"
#include <mutex>

//Maneja las cosas que selecciona el usuario
class Selector {
private:
    //Coordinate clickX;
    Coordinate inventorySlot;
    Coordinate selectedTile;
    GameType::EquipmentPlace selectedEquipment;
    std::mutex m;

public:
    Selector();

    //Se fija si el click fue en el mapa o en el inventario. Dependiendo el caso
    //Guarda el tile/inventorySlot/equipable que se haya seleccionado.
    void handleEvent(Coordinate click, Coordinate playerPos, Window& window);

    //Devuelve el inventorySlot actualmente seleccionado
    int getInventorySlot();

    //Devuelve la coordenada del tile actualmente seleccionado
    Coordinate getSelectedTile();

    //Me devuelve el item equipado que seleccione
    GameType::EquipmentPlace getSelectedEquipment();

    //Devuelve true si seleccione un tile
    static bool hasSelectedTile(Coordinate click);

    //Devuelve true si seleccione un slot del inventario
    static bool hasSelectedSlot(Coordinate click);

    //Devuelve true si seleccione un item equipado
    static bool hasSelectedEquipment(Coordinate click);

    //Setea el tile seleccionado a (0,0)
    void resetTileSelection();

    ~Selector();

private:
    void _verifyTileSelection(Coordinate playerPos, Coordinate click);
    void _verifyInventorySlotSelection(Coordinate click);
    static bool _isInsideRect(Coordinate click, int left, int right, int top,
            int bottom) ;
    void _verifySelectedEquipment(Coordinate click);


};


#endif //ARGENTUM_SELECTOR_H
```

```
//
// Created by ivan on 20/6/20.
//

#include "../Client/GameConstants.h"
#include "Selector.h"
#include <iostream>

#define DEFAULT_MAP_LEFT 20
#define DEFAULT_MAP_RIGHT 1044
#define DEFAULT_MAP_TOP 236
#define DEFAULT_MAP_BOTTOM 876

#define CAMERA_X_OFFSET 27
#define CAMERA_Y_OFFSET 15

#define DEFAULT_INVENTORY_LEFT 1122
#define DEFAULT_INVENTORY_RIGHT 1434
#define DEFAULT_INVENTORY_TOP 260
#define DEFAULT_INVENTORY_BOTTOM 548

#define INVENTORY_SLOT_WIDTH 78
#define INVENTORY_SLOT_HEIGHT 72



Selector::Selector() {
    inventorySlot = {0, 0};
    selectedTile = {0, 0};
}

void Selector::handleEvent(Coordinate click, Coordinate playerPos, Window& window) {
    std::lock_guard<std::mutex> l(m);
    _verifyTileSelection(playerPos, click);
    _verifyInventorySlotSelection(click);
    _verifySelectedEquipment(click);
}

void Selector::_verifyTileSelection(Coordinate playerPos, Coordinate click) {
    //Veo si clickeo adentro del mapa
    if (_isInsideRect(click, DEFAULT_MAP_LEFT, DEFAULT_MAP_RIGHT, DEFAULT_MAP_TOP,
            DEFAULT_MAP_BOTTOM)){
        //Esto es cuando no esta en los extremos
        int playerXTile = playerPos.j;
        int playerYTile = playerPos.i;
        int relativeXTile = (click.j − DEFAULT_MAP_LEFT + CAMERA_X_OFFSET) / TILE_WIDTH;
        int relativeYTile = (click.i − DEFAULT_MAP_TOP − CAMERA_Y_OFFSET) / TILE_HEIGHT;
        selectedTile.j = playerXTile + (relativeXTile − 4);
        selectedTile.i = playerYTile + (relativeYTile − 2);

        //Me fijo los extremos
        if (playerXTile < 4){
            selectedTile.j = (click.j − DEFAULT_MAP_LEFT) / TILE_WIDTH;
        } else if (playerXTile > 95){
            selectedTile.j = 92 + ((click.j − DEFAULT_MAP_LEFT) / TILE_WIDTH);
        }
        if (playerYTile < 3){
            selectedTile.i = (click.i − DEFAULT_MAP_TOP) / TILE_HEIGHT;
        }else if (playerYTile > 97){
            selectedTile.i = 95 + ((click.i − DEFAULT_MAP_TOP) / TILE_HEIGHT);
        }
    }
```

```
}

void Selector::_verifyInventorySlotSelection(Coordinate click) {
    //Veo si clickeo adentro del inventario
    if (_isInsideRect(click, DEFAULT_INVENTORY_LEFT, DEFAULT_INVENTORY_RIGHT,
            DEFAULT_INVENTORY_TOP, DEFAULT_INVENTORY_BOTTOM)){
        inventorySlot.j = (click.j − DEFAULT_INVENTORY_LEFT) / INVENTORY_SLOT_WIDTH;
        inventorySlot.i = (click.i − DEFAULT_INVENTORY_TOP) / INVENTORY_SLOT_HEIGHT;
    }
}

void Selector::_verifySelectedEquipment(Coordinate click) {
    if (_isInsideRect(click, 1320, 1392, 660, 735)){//Casco
        selectedEquipment = GameType::EQUIPMENT_PLACE_HEAD;
    } else if (_isInsideRect(click, 1397, 1469, 660, 735)){//Chest
        selectedEquipment = GameType::EQUIPMENT_PLACE_CHEST;
    } else if (_isInsideRect(click, 1320, 1392, 760, 835)){//Weapon
        selectedEquipment = GameType::EQUIPMENT_PLACE_WEAPON;
    } else if (_isInsideRect(click, 1397, 1469, 760, 835)){//Shield
        selectedEquipment = GameType::EQUIPMENT_PLACE_SHIELD;
    }
}

bool Selector::hasSelectedTile(Coordinate click) {
    return _isInsideRect(click, DEFAULT_MAP_LEFT, DEFAULT_MAP_RIGHT, DEFAULT_MAP_TOP,
                         DEFAULT_MAP_BOTTOM);
}

bool Selector::hasSelectedSlot(Coordinate click) {
    return _isInsideRect(click, DEFAULT_INVENTORY_LEFT, DEFAULT_INVENTORY_RIGHT,
                         DEFAULT_INVENTORY_TOP, DEFAULT_INVENTORY_BOTTOM);
}

bool Selector::hasSelectedEquipment(Coordinate click) {
    //Aca me fijo si esta entre los 4 cuadrados de equipamiento
    return _isInsideRect(click, 1320, 1469,
                         660, 835);
}

int Selector::getInventorySlot() {
    std::lock_guard<std::mutex> l(m);
    return inventorySlot.j + (4 * inventorySlot.i);
}

Coordinate Selector::getSelectedTile() {
    std::lock_guard<std::mutex> l(m);
    return {selectedTile.i, selectedTile.j};
}

GameType::EquipmentPlace Selector::getSelectedEquipment() {
    std::lock_guard<std::mutex> l(m);
    return selectedEquipment;
}

void Selector::resetTileSelection() {
    std::lock_guard<std::mutex> l(m);
    selectedTile = {0, 0};
}

bool Selector::_isInsideRect(Coordinate click, int left, int right, int top, int bottom) {
    return (click.j > left ∧ click.j < right ∧ click.i > top ∧ click.i < bottom);
```

```cpp
}


Selector::~Selector() = default;
```

```cpp
//
// Created by ivan on 18/6/20.
//

#ifndef ARGENTUM_MINICHAT_H
#define ARGENTUM_MINICHAT_H

#include "../Text/Text.h"
#include "../../Screen/Window.h"
#include "../../Client/GameConstants.h"
#include <list>
#include "../../Map/Coordinate.h"
#include <mutex>
#include <atomic>

class Minichat {
private:
    Font minichatFont;
    Text input;
    std::atomic<bool> processedInput{false};
    std::list<Text> texts;
    SDL_Renderer& renderer;
    bool focusOnMinichat;
    int firstToRender;
    std::mutex generalMutex;
    std::mutex inputMutex;

public:
    explicit Minichat(SDL_Renderer& renderer);

    //Encola newText para imprimirlo en el minichat
    void queueText(std::string &newText);

    //Recibe texto y si es necesario lo separa en varias lineas del minichat
    void receiveText(const std::string &text);

    //Renderiza todos los mensajes del minichat + lo que escribio el usuario
    void render();

    //Borra la ultima letra del input
    void handleBackspace();

    //Asigna el texto del evento de input al texto de input del usuario
    void handleTextInput(SDL_Event &e);

    //Se fija si el click fue dentro del minichat. Si es asi habilita el input d
e texto
    void handleMouseButtonDown(Coordinate click, Window &window);

    //Scrollea por los mensajes del minichat
    void handleMouseWheel(SDL_Event &e);

    //Borra los mensajes del minichat
    void clearMinichat();

    /* Devuelve el input del usuario en el minichat para que sea parseado como c
omando */
    std::string handleReturnKey();

    ~Minichat();

private:
    static bool _isInsideMinichat(int xClick, int yClick);
    void _queueInputIfProcessed();
};

```

```
#endif //ARGENTUM_MINICHAT_H
```

```
//
// Created by ivan on 18/6/20.
//

#include "Minichat.h"

#define MINICHAT_X_OFFSET 15
#define MINICHAT_Y_OFFSET 15

#define MAX_TEXT_LEN 85
#define MAX_MSGS 24 //El maximo de mensajes que se van a ver al scrollear
#define MAX_MSGS_TO_RENDER 8

#define MINICHAT_FONT_PATH "/var/Argentum/Assets/Fonts/Raleway−Medium.ttf"

Minichat::Minichat(SDL_Renderer& renderer) : minichatFont(MINICHAT_FONT_PATH, 20
),
                                input(minichatFont,renderer), renderer(renderer)
 {
    focusOnMinichat = false;
    input.updateText(":");

    //Lleno el vector con mensajes vacios
    for (int i = 0; i < MAX_MSGS; ++i) {
        texts.emplace_back(minichatFont,renderer);
    }
    firstToRender = 0;
}

std::string Minichat::handleReturnKey() {
    std::lock_guard<std::mutex> l(inputMutex);
    std::string cmd = input.getText();
    if (cmd.size() > 1) {
        cmd.erase(0, 1);//Le saco ":"
        input.getText().erase(0, 1);
        processedInput = true;
        return cmd;
    }
    return "";
}

void Minichat::handleBackspace() {
    std::lock_guard<std::mutex> l(inputMutex);
    if (focusOnMinichat) {
        if (input.getTextLength() > 1) {
            −−input;
        }
    }
}

void Minichat::handleTextInput(SDL_Event &e) {
    std::lock_guard<std::mutex> l(inputMutex);
    std::string newInput = e.text.text;
    if (input.getTextLength() < MAX_TEXT_LEN) {
        input += std::move(newInput);
    }
}

void Minichat::handleMouseButtonDown(Coordinate click, Window& window) {
    focusOnMinichat = _isInsideMinichat(click.j, click.i);
    if (focusOnMinichat) {
        SDL_StartTextInput();
    } else {
        SDL_StopTextInput();
    }
}
```

```

void Minichat::handleMouseWheel(SDL_Event& e) {
    std::lock_guard<std::mutex> l(generalMutex);
    if (focusOnMinichat) {
        if (e.wheel.y > 0) // scroll up
        {
            firstToRender += 1;
            if (firstToRender > MAX_MSGS − MAX_MSGS_TO_RENDER)
                firstToRender −= 1;
        } else if (e.wheel.y < 0) // scroll down
        {
            firstToRender −= 1;
            if (firstToRender < 0)
                firstToRender = 0;
        }
    }
}

void Minichat::receiveText(const std::string& text) {
    //Separa un mensaje separado con \n en varias lineas
    int currPos = −1;
    int nextPos;
    while (currPos < (int)text.size()) {
        nextPos =  text.find('\n', currPos + 1);
        std::string substr = text.substr(currPos + 1, nextPos − currPos − 1);
        currPos = nextPos;
        queueText(substr);
        if (nextPos ≡ −1)
            break;
    }
}

//Imprime los mensajes relevantes
void Minichat::queueText(std::string& newText) {
    std::lock_guard<std::mutex> l(generalMutex);
    if (¬newText.empty()) {
        texts.pop_back();
        texts.emplace_front(minichatFont, renderer);
        *(texts.front().updateText(std::move(newText)));
    }
}

void Minichat::clearMinichat() {
    std::lock_guard<std::mutex> l(generalMutex);
    for (auto & text : texts) {
        text.updateText("");
    }
}

void Minichat::_queueInputIfProcessed() {
    if (processedInput) { /*No lo encola el thread de input porque SDL no puede
actualizar texturas desde otro thread*/
        queueText(input.getText());
        input.updateText(":");
        processedInput = false;
    }
}

void Minichat::render() {
    std::lock_guard<std::mutex> li(inputMutex);
    _queueInputIfProcessed();
    std::lock_guard<std::mutex> lg(generalMutex);
    (*input).render(0,178);
    //renderizo mensajes encolados.
    int textNum = 0;
    for (auto & text : texts) {
```

```
        if (textNum ≥ firstToRender) {
            if (¬text.getText().empty()) {
                text.render(0,140 − 20*(textNum − firstToRender));
            }
        }
        ++textNum;
        if (textNum ≥ MAX_MSGS_TO_RENDER + firstToRender) break;
    }
}

bool Minichat::_isInsideMinichat(int xClick, int yClick){
    return (xClick ≥ MINICHAT_X_OFFSET) ∧
    (xClick < (MINICHAT_X_OFFSET + DEFAULT_MINICHAT_WIDTH))
    ∧ ( yClick ≥ MINICHAT_Y_OFFSET )
    ∧ ( yClick ≤ (MINICHAT_Y_OFFSET + DEFAULT_MINICHAT_HEIGHT));
}

Minichat::~Minichat() = default;


```

```cpp
//
// Created by marcos on 7/2/20.
//

#ifndef ARGENTUM_PLAYERSTATS_H
#define ARGENTUM_PLAYERSTATS_H

#include "../../Map/Coordinate.h"

struct PlayerStats {
    std::string nickname;
    int32_t totalHealth, totalMana, nextLevelXP;
    int32_t health, mana, xp;
    int32_t level;
    int32_t constitution, strength, agility, intelligence;
    int32_t gold, safeGold;
    Coordinate position;
};
enum EquippedItems {
    Helmet,
    Armor,
    Weapon,
    Shield,
};

#endif //ARGENTUM_PLAYERSTATS_H
```

```cpp
//
// Created by ivan on 13/6/20.
//

#ifndef ARGENTUM_PLAYERINVENTORYGUI_H
#define ARGENTUM_PLAYERINVENTORYGUI_H

#include "../../Texture/TextureRepository.h"
#include "../Text/Text.h"
#include "PlayerInfoGUI.h"
#include <list>
#include <unordered_map>

struct FixedText {
    Text inventory;
    Text title;

    FixedText(SDL_Renderer& renderer, Font& font) : inventory(font, renderer, "I
NVENTORY"),
                                                     title(font, renderer, "Argentum")
 {}
};

class PlayerInventoryGUI {
private:
    Font textFont;
    Text text;
    TextureRepository& repo;
    SDL_Renderer& renderer;
    std::vector<Texture*> inventoryTextures;
    std::unordered_map<EquippedItems, Texture*> equippedTextures;
    PlayerInfoGUI& pInfo;
    FixedText fixedText;
    int32_t gold, safeGold{};

public:
    PlayerInventoryGUI(TextureRepository& repo, SDL_Renderer& renderer, PlayerIn
foGUI& playerInfo);

    /* Agrega la textura del item al slot correspondiente */
    void addInventoryItem(TextureID texture, int32_t slot);

    /* Agrega un item a la posicion correspondiente de los equipables */
    void addEquipableItem(TextureID texture, EquippedItems item);

    /* Actualiza la cantidad de oro y oro seguro */
    void updateGold(int32_t gold, int32_t _safeGold);

    /* Renderiza el inventario y la informacion del jugador */
    void render(int32_t selectedSlot);

private:
    void _drawInventoryOutlines(int32_t x);
    void _renderInventoryItems();
    void _drawEquipableOutlines();
    void _renderEquipableItems();
    void _renderText();
    void _renderSkills();
};


#endif //ARGENTUM_PLAYERINVENTORYGUI_H
```

```
//
// Created by ivan on 13/6/20.
//

#include "PlayerInventoryGUI.h"

#define INVENTORY_SIZE 16

#define INVENTORY_ITEMS_X_OFFSET 45
#define INVENTORY_ITEMS_Y_OFFSET 235

#define ITEM_WIDTH 72
#define ITEM_HEIGHT 75

#define INVENTORY_OUTLINES_X_OFFSET 73
#define INVENTORY_OUTLINES_Y_OFFSET 260

#define INVENTORY_FONT_PATH "/var/Argentum/Assets/Fonts/medieval.ttf"

PlayerInventoryGUI::PlayerInventoryGUI(TextureRepository &repo,SDL_Renderer &ren
derer,
                                       PlayerInfoGUI& playerInfo) :
                                       textFont(INVENTORY_FONT_PATH, 25),
                                            text(textFont,renderer), repo(repo),
                                            renderer(renderer), pInfo(playerInfo
),
                                            fixedText(renderer, textFont) {
    gold = 0;
    for (int i = 0; i < INVENTORY_SIZE; ++i) {//Inicializo el vector con nullptr
        inventoryTextures.push_back(nullptr);
    }
    equippedTextures[Helmet] = nullptr;
    equippedTextures[Armor] = nullptr;
    equippedTextures[Weapon] = nullptr;
    equippedTextures[Shield] = nullptr;
}

void PlayerInventoryGUI::addInventoryItem(TextureID texture, int32_t slot) {
    if (texture ≡ Nothing) {
        inventoryTextures[slot] = nullptr;
    } else {
        inventoryTextures[slot] = &repo.getTexture(texture);
    }
}

void PlayerInventoryGUI::addEquipableItem(TextureID texture, EquippedItems item)
 {
    if (texture ≠ Nothing) {
        Texture* currTexture = &repo.getTexture(texture);
        equippedTextures.at(item) = currTexture;
    } else {
        equippedTextures.at(item) = nullptr;
    }
}

void PlayerInventoryGUI::updateGold(int32_t _gold, int32_t _safeGold) {
    gold = _gold;
    safeGold = _safeGold;
}

void PlayerInventoryGUI::render(int32_t selectedSlotX) {
    _drawInventoryOutlines(selectedSlotX);
    _drawEquipableOutlines();
    _renderInventoryItems();
    _renderEquipableItems();
    _renderText();
```

```
}

void PlayerInventoryGUI::_renderText() {
    fixedText.title.render(215, 25);
    fixedText.inventory.render(160, 225);
    pInfo.getGoldText().render(140, 565);
    pInfo.getLevelText().render(70, 50);
    _renderSkills();
    pInfo.getPositionText().render(200, 880);
    pInfo.getNicknameText().render(210,95);
}

void PlayerInventoryGUI::_renderSkills() {
    pInfo.getStrengthText().render(40, 660);
    pInfo.getConstitutionText().render(40, 700);
    pInfo.getIntelligenceText().render(40, 740);
    pInfo.getAgilityText().render(40, 780);
}

void PlayerInventoryGUI::_renderInventoryItems() {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; ++j) {
            if (inventoryTextures[4*i + j] ≠ nullptr){
                inventoryTextures[4*i + j]→render(INVENTORY_ITEMS_X_OFFSET +
                                        (ITEM_WIDTH + 6) * j,INVENTORY_ITEMS_Y_O
FFSET
                                        +(i) * (ITEM_HEIGHT − 1),0);
            }
        }
    }
}

void PlayerInventoryGUI::_renderEquipableItems() {
    if (equippedTextures.at(Helmet)) {
        equippedTextures.at(Helmet)→render(250, 635,0);
    }
    if (equippedTextures.at(Armor)) {
        equippedTextures.at(Armor)→render(325, 635, 0);
    }
    if (equippedTextures.at(Weapon)) {
        equippedTextures.at(Weapon)→render(250, 735, 0);
    }
    if (equippedTextures.at(Shield)) {
        equippedTextures.at(Shield)→render(325, 735, 0);
    }
}

/* Renderiza los cuadrados en los que se divide el inventario */
void PlayerInventoryGUI::_drawInventoryOutlines(int32_t selectedSlot) {
    SDL_Rect outlineRect;
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
            outlineRect = { INVENTORY_OUTLINES_X_OFFSET + (ITEM_WIDTH + 6) * j,
                            INVENTORY_OUTLINES_Y_OFFSET + (ITEM_HEIGHT − 3) * i,
                            ITEM_WIDTH + 6, ITEM_HEIGHT − 3 };
            SDL_SetRenderDrawColor(&renderer, 0x3f, 0x2a,
                                   0x14, 0xFF);
            if ((j + 4 * i) ≡ selectedSlot){
                SDL_SetRenderDrawColor(&renderer, 0xff, 0xff,
                                       0xff, 0xFF);
            }
            SDL_RenderDrawRect( &renderer, &outlineRect );
        }
    }
}

```

```cpp
/* Renderiza los cuadrados donde se muestran los items equipados */
void PlayerInventoryGUI::_drawEquipableOutlines() {
    SDL_Rect outlineRect;
    for (int j = 0; j < 2; ++j) {
        outlineRect = { 275 + 75 * j, 660, ITEM_WIDTH, ITEM_HEIGHT };
        SDL_SetRenderDrawColor( &renderer, 0x3f,0x2a,
                                0x14, 0xFF );
        SDL_RenderDrawRect( &renderer, &outlineRect );
    }
    for (int j = 0; j < 2; ++j) {
        outlineRect = { 275 + 75 * j, 760, ITEM_WIDTH, ITEM_HEIGHT };
        SDL_SetRenderDrawColor( &renderer, 0x3f,0x2a,
                                0x14, 0xFF );
        SDL_RenderDrawRect( &renderer, &outlineRect );
    }

}
```

```cpp
//
// Created by ivan on 13/6/20.
//

#ifndef ARGENTUM_PLAYERINFOGUI_H
#define ARGENTUM_PLAYERINFOGUI_H

#include "../Text/Text.h"
#include "PlayerStats.h"
#include "../../Map/Coordinate.h"
#include "../../Sound/SoundPlayer.h"

struct GUIInfoText {
    Text nickname;
    Text health;
    Text mana;
    Text xp;
    Text level;
    Text constitution, strength, agility, intelligence;
    Text gold;
    Text position;

    GUIInfoText(SDL_Renderer& renderer, Font& font) : nickname(font, renderer),
    health(font, renderer), mana(font, renderer),
    xp(font, renderer),level(font, renderer),
    constitution(font, renderer),
    strength(font, renderer), agility(font, renderer),
    intelligence(font, renderer), gold(font, renderer),
    position(font, renderer) {}
};

class PlayerInfoGUI {
private:
    Font infoFont;
    Text info;
    SDL_Renderer& renderer;
    PlayerStats pInfo{};
    SoundPlayer& soundPlayer;
    GUIInfoText infoText;

public:
    PlayerInfoGUI(SDL_Renderer& renderer, SoundPlayer& soundPlayer);

    Text& getLevelText();
    int32_t getXPos() const;
    int32_t getYPos() const;
    std::string& getNickname();
    Text& getStrengthText();
    Text& getConstitutionText();
    Text& getAgilityText();
    Text& getIntelligenceText();
    Text& getPositionText();
    Text& getNicknameText();
    Text& getGoldText();

    /* Actualiza todas las stats del jugador */
    void update(PlayerStats& generalInfo);

    /* Renderiza las barras de vida, xp y mana */
    void render();

private:
    void _renderInfoBar(Text& textToRender, int32_t infoCurr, int32_t infoTotal,
 int32_t xOffset,
                       int32_t barLen, SDL_Color color);
    void _updateHealth(int32_t currHealth, int32_t totalHealth);
    void _updateMana(int32_t currMana, int32_t totalMana);
    void _updateXP(int32_t currXP,int32_t nextLevelXP);
    void _updateLevel(int32_t newLevel);
    void _updatePosition(Coordinate position);
    void _updateStrength(int32_t strength);
    void _updateConstitution(int32_t constitution);
    void _updateAgility(int32_t agility);
    void _updateIntelligence(int32_t intelligence);
    void _updateNickname(std::string∧ name);
    void _updateGold(int32_t gold, int32_t safeGold);
};


#endif //ARGENTUM_PLAYERINFOGUI_H
```

```
//
// Created by ivan on 13/6/20.
//

#include "PlayerInfoGUI.h"

#include <utility>
#include "../../Client/GameConstants.h"

#define PLAYER_INFO_FONT_PATH "/var/Argentum/Assets/Fonts/medieval.ttf"
#define HEALTH_TEXT "HEALTH: " + std::to_string(pInfo.health) + "/" + std::to_st
ring(pInfo.totalHealth)
#define MANA_TEXT "MANA: " + std::to_string(pInfo.mana) + "/" + std::to_string(p
Info.totalMana)
#define XP_TEXT "XP: " + std::to_string(pInfo.xp) + "/" + std::to_string(pInfo.ne
xtLevelXP)
#define POSITION_TEXT "X: " + std::to_string(getXPos()) + "  " + "Y: " + std::to_s
tring(getYPos())
#define GOLD_TEXT "GOLD: " + std::to_string(pInfo.gold) + "(" + std::to_string(p
Info.safeGold) + ")"

PlayerInfoGUI::PlayerInfoGUI(SDL_Renderer &renderer, SoundPlayer& soundPlayer) :
infoFont(PLAYER_INFO_FONT_PATH, 25), info(infoFont, renderer), renderer(renderer
),
soundPlayer(soundPlayer), infoText(renderer, infoFont) {

    pInfo = {};
    *(infoText.health.updateText(HEALTH_TEXT));
    *(infoText.mana.updateText(MANA_TEXT));
    *(infoText.xp.updateText(XP_TEXT));
}

void PlayerInfoGUI::_updateHealth(int32_t currHealth, int32_t totalHealth) {
    if (pInfo.health ≠ currHealth ∨ pInfo.totalHealth ≠ totalHealth) {
        pInfo.health = currHealth;
        pInfo.totalHealth = totalHealth;
        *(infoText.health.updateText(HEALTH_TEXT));
    }
}

void PlayerInfoGUI::_updateMana(int32_t currMana, int32_t totalMana) {
    if (pInfo.mana ≠ currMana ∨ pInfo.totalMana ≠ totalMana) {
        pInfo.mana = currMana;
        pInfo.totalMana = totalMana;
        *(infoText.mana.updateText(MANA_TEXT));
    }
}

void PlayerInfoGUI::_updateXP(int32_t currXP, int32_t nextLevelXP) {
    if (pInfo.xp ≠ currXP ∨ pInfo.nextLevelXP ≠ nextLevelXP) {
        pInfo.xp = currXP;
        pInfo.nextLevelXP = nextLevelXP;
        *(infoText.xp.updateText(XP_TEXT));
    }
}

void PlayerInfoGUI::_updateLevel(int32_t newLevel) {
    if (pInfo.level ≠ newLevel) {
        if (newLevel > pInfo.level) {
            soundPlayer.queueSound(LevelUpSound);
        }
        pInfo.level = newLevel;
        *(infoText.level.updateText(std::to_string(pInfo.level)));
    }
}

```

```
void PlayerInfoGUI::_updateStrength(int32_t strength) {
    if (pInfo.strength ≠ strength) {
        pInfo.strength = strength;
        *(infoText.strength.updateText("STRENGTH : "
                                + std::to_string(pInfo.strength)));
    }
}

void PlayerInfoGUI::_updateAgility(int32_t agility) {
    if (pInfo.agility ≠ agility) {
        pInfo.agility = agility;
        *(infoText.agility.updateText("AGILITY : "
                                + std::to_string(pInfo.agility)));
    }
}

void PlayerInfoGUI::_updateConstitution(int32_t constitution) {
    if (pInfo.constitution ≠ constitution) {
        pInfo.constitution = constitution;
        *(infoText.constitution.updateText("CONSTITUTION : "
                            + std::to_string(pInfo.constitution)));
    }
}

void PlayerInfoGUI::_updateIntelligence(int32_t intelligence) {
    if (pInfo.intelligence ≠ intelligence) {
        pInfo.intelligence = intelligence;
        *(infoText.intelligence.updateText("INTELLIGENCE : "
                                   + std::to_string(pInfo.intelligence))
);
    }
}

void PlayerInfoGUI::_updatePosition(Coordinate position) {
    if (pInfo.position ≠ position) {
        pInfo.position = position;
        *(infoText.position.updateText(POSITION_TEXT));
    }
}

void PlayerInfoGUI::_updateNickname(std::string∧ name) {
    if (pInfo.nickname ≠ name) {
        pInfo.nickname = std::move(name);
        *(infoText.nickname.updateText(pInfo.nickname));
    }
}

void PlayerInfoGUI::_updateGold(int32_t gold, int32_t safeGold) {
    if (pInfo.gold ≠ gold ∨ pInfo.safeGold ≠ safeGold) {
        pInfo.gold = gold;
        pInfo.safeGold = safeGold;
        (infoText.gold.updateText(GOLD_TEXT)).operator*(
                                SDL_Color{0xFF,0xFF,0x00});
    }
}

void PlayerInfoGUI::render() {
    _renderInfoBar(infoText.health, pInfo.health, pInfo.totalHealth,
            HEALTH_BAR_X_OFFSET, 265,{0x99, 0x00,0x00});

    _renderInfoBar(infoText.mana,
            pInfo.mana, pInfo.totalMana, MANA_BAR_X_OFFSET, 265,{0x00, 0x33, 0x6
6});

    _renderInfoBar(infoText.xp,
            pInfo.xp, pInfo.nextLevelXP, XP_BAR_X_OFFSET, 265,{0x00, 0x66, 0x00}
```

```
);
}

void PlayerInfoGUI::_renderInfoBar(Text& textToRender, int32_t infoCurr, int32_t
 infoTotal,
                                  int32_t xOffset, int32_t barLen, SDL_Color co
lor) {
    float bar = 0;
    if (infoTotal ≠ 0){
        bar = barLen * ((float)infoCurr / (float)infoTotal);
    }

    //Barra
    SDL_Rect fillRect = {xOffset, 10, (int)bar, BAR_HEIGHT};
    SDL_SetRenderDrawColor(&renderer, color.r, color.g, color.b, 0xFF);
    SDL_RenderFillRect( &renderer, &fillRect);

    //outline de la barra
    SDL_Rect outlineRect = {xOffset, 10, (int)barLen, BAR_HEIGHT };
    SDL_SetRenderDrawColor( &renderer, 0x00,0x00,0x00, 0xFF );
    SDL_RenderDrawRect( &renderer, &outlineRect );
    //Texto de la barra
    textToRender.render(xOffset, 10);
}

Text& PlayerInfoGUI::getLevelText() {
    return infoText.level;
}

int32_t PlayerInfoGUI::getXPos() const {
    return pInfo.position.j;
}

int32_t PlayerInfoGUI::getYPos() const {
    return pInfo.position.i;
}

Text& PlayerInfoGUI::getStrengthText() {
    return infoText.strength;
}

Text& PlayerInfoGUI::getConstitutionText() {
    return infoText.constitution;
}

Text& PlayerInfoGUI::getAgilityText() {
    return infoText.agility;
}

Text& PlayerInfoGUI::getIntelligenceText() {
    return infoText.intelligence;
}

Text& PlayerInfoGUI::getNicknameText() {
    return infoText.nickname;
}

std::string& PlayerInfoGUI::getNickname() {
    return pInfo.nickname;
}

void PlayerInfoGUI::update(PlayerStats &generalInfo) {
    _updateHealth(generalInfo.health, generalInfo.totalHealth);
    _updateMana(generalInfo.mana, generalInfo.totalMana);
    _updateXP(generalInfo.xp, generalInfo.nextLevelXP);
    _updateLevel(generalInfo.level);
```

```cpp
    _updatePosition(generalInfo.position);
    _updateStrength(generalInfo.strength);
    _updateConstitution(generalInfo.constitution);
    _updateAgility(generalInfo.agility);
    _updateIntelligence(generalInfo.intelligence);
    _updateGold(generalInfo.gold, generalInfo.safeGold);
    _updateNickname(std::move(generalInfo.nickname));
}

Text &PlayerInfoGUI::getGoldText() {
    return infoText.gold;
}

Text &PlayerInfoGUI::getPositionText() {
    return infoText.position;
}
```

```cpp
//
// Created by marcos on 6/29/20.
//

#ifndef ARGENTUM_UPDATERECEIVER_H
#define ARGENTUM_UPDATERECEIVER_H

#include "../../libs/Thread.h"
#include "Update.h"
#include "ClientProtocol.h"
#include <msgpack.hpp>

class UpdateEvent;
class Socket;
class UpdateManager;

class UpdateReceiver : public Thread {
private:
    ClientProtocol& protocol;
    UpdateManager& updateManager;
    Update currentUpdate;
    msgpack::object_handle handler;
    std::size_t offset{0};
    Socket& socket;
    std::vector<char> buffer;
    bool& quit;

public:
    UpdateReceiver(ClientProtocol& protocol, UpdateManager& _updateManager,
                   Socket& _socket, bool& _quit) : protocol(protocol),
                   updateManager(_updateManager), socket(_socket), quit(_quit) {}

    void run() override;

private:
    void _processAttack();
    void _processCreateItem();
    void _processUnequip();
    void _processUpdate(uint32_t msgLength);
    void _processRemoveEntity();
    void _processMoveUpdate();
    void _receivePlayerData();
    void _processCreateEntity();
    void _processEquipped();
    void _processPlayerDeath();
    void _processDestroyItem();
    void _processTeleportEntity();
    void _processPlayerResurrect();
    void _processPlayerLevelUp();
};


#endif //ARGENTUM_UPDATERECEIVER_H
```

```
//
// Created by marcos on 6/29/20.
//
#include <netinet/in.h>
#include "UpdateReceiver.h"
#include "../../libs/Socket.h"
#include "../UpdateEvents/UpdateMove.h"
#include "../UpdateEvents/UpdateCreatePlayer.h"
#include "../UpdateEvents/UpdateCreateNPC.h"
#include "../UpdateEvents/UpdateGUI.h"
#include "../UpdateEvents/UpdateRemoveEntity.h"
#include "../UpdateEvents/UpdateEquip.h"
#include "../UpdateEvents/UpdateCreateItem.h"
#include "../UpdateEvents/UpdatePlayerDeath.h"
#include "../UpdateEvents/UpdateAttack.h"
#include "../UpdateEvents/UpdateDestroyItem.h"
#include "../UpdateEvents/UpdateTeleportEntity.h"
#include "../UpdateEvents/UpdatePlayerResurrect.h"
#include "../UpdateEvents/UpdateLevelUp.h"
#include "UpdateManager.h"

MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::Direction)
MSGPACK_ADD_ENUM(GameType::Entity)
MSGPACK_ADD_ENUM(GameType::EquipmentPlace)
MSGPACK_ADD_ENUM(GameType::ItemType)
MSGPACK_ADD_ENUM(GameType::Weapon)

/* Recibe un update del server, lo procesa y lo encola en una queue de functors
 * para que sea ejecutado en el thread principal */
void UpdateReceiver::run() {
    try {
        uint32_t msgLength = 0;
        while (¬quit) {
            offset = 0;
            socket.receive((char *) (&msgLength), sizeof(uint32_t));
            msgLength = ntohl(msgLength);
            buffer.clear();
            buffer.resize(msgLength);
            socket.receive(buffer.data(), msgLength);
            _processUpdate(msgLength);
            updateManager.push(currentUpdate);
        }
    } catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    } catch (...) {
        std::cerr << "Unkown error in UpdateReceiver" << std::endl;
    }
    quit = true;
}

/* Chequea que tipo de evento recibio y lo procesa. Luego recibe toda la informa
cion del jugador */
void UpdateReceiver::_processUpdate(uint32_t msgLength) {
    msgpack::type::tuple<GameType::EventID> id;
    while (offset < msgLength) {
        handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
        handler→convert(id);
        switch (std::get<0>(id)) {
            case GameType::MOVED:
                _processMoveUpdate();
                break;
            case GameType::ATTACK:
                _processAttack();
                break;
            case GameType::UNEQUIP:
```

```
                _processUnequip();
                break;
            case GameType::EQUIPPED:
                _processEquipped();
                break;
            case GameType::CREATE_ENTITY:
                _processCreateEntity();
                break;
            case GameType::CREATE_ITEM:
                _processCreateItem();
                break;
            case GameType::REMOVE_ENTITY:
                _processRemoveEntity();
                break;
            case GameType::PLAYER_DEATH:
                _processPlayerDeath();
                break;
            case GameType::DESTROY_ITEM:
                _processDestroyItem();
                break;
            case GameType::TELEPORTED:
                _processTeleportEntity();
                break;
            case GameType::RESURRECTED:
                _processPlayerResurrect();
                break;
            case GameType::PLAYER_LEVEL_UP:
                _processPlayerLevelUp();
                break;
            default:
                std::cerr << std::get<0>(id) << " is an unknown command" << std::end
l;
                break;
        }
    }
    _receivePlayerData();
}

void UpdateReceiver::_processPlayerLevelUp() {
    msgpack::type::tuple<std::string, int32_t> playerData;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(playerData);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdateLevelUp(
                                        std::move(std::get<0>(playerData)),
                                                std::get<1>(playerData))));
}

void UpdateReceiver::_processTeleportEntity() {
    msgpack::type::tuple<std::string, int32_t, int32_t> teleportData;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(teleportData);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdateTeleportEntity(
                        std::move(std::get<0>(teleportData)),
                                {std::get<1>(teleportData),
                                std::get<2>(teleportData)})));
}

void UpdateReceiver::_processDestroyItem() {
    msgpack::type::tuple<int32_t, int32_t> itemPosition;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(itemPosition);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdateDestroyItem({std::
get<0>(itemPosition),
                                                std::get<1>(itemPosition)})));
}

```

```
void UpdateReceiver::_processAttack() {
    msgpack::type::tuple<std::string, int32_t, int32_t, GameType::Weapon,
                                              GameType::Direction> entity;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(entity);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(
            new UpdateAttack(std::get<0>(entity),
                             {std::get<1>(entity), std::get<2>(entity)},
                    std::get<3>(entity), std::get<4>(entity))));
}


void UpdateReceiver::_processPlayerDeath() {
    msgpack::type::tuple<std::string> player;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(player);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(
            new UpdatePlayerDeath(std::move(std::get<0>(player)))));
}

void UpdateReceiver::_processPlayerResurrect() {
    msgpack::type::tuple<std::string> player;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(player);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdatePlayerResurrect(
                                            std::move(std::get<0>(player)))));
}

void UpdateReceiver::_processCreateItem() {
    msgpack::type::tuple<GameType::ItemType, int32_t, int32_t, int32_t> itemData
;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(itemData);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdateCreateItem(std::ge
t<0>(itemData),
                std::get<1>(itemData), {std::get<2>(itemData), std::get<3>(itemD
ata)})));
}

void UpdateReceiver::_processUnequip() {
    msgpack::type::tuple<std::string, GameType::EquipmentPlace> data;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(data);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdateEquip(std::move(st
d::get<0>(data)),
                                       std::get<1>(data), UNEQUIP)));
}

void UpdateReceiver::_processEquipped() {
    msgpack::type::tuple<std::string, GameType::EquipmentPlace, int32_t> data;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(data);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdateEquip(std::move(st
d::get<0>(data)),
                std::get<1>(data), std::get<2>(data))));
}

void UpdateReceiver::_processMoveUpdate() {
    msgpack::type::tuple<GameType::Direction, int32_t, std::string, bool> moveIn
fo;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(moveInfo);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdateMove(std::move(std
::get<2>(moveInfo)),
            std::get<0>(moveInfo), std::get<1>(moveInfo), std::get<3>(moveInfo))
));
```

```
}

void UpdateReceiver::_processRemoveEntity() {
    msgpack::type::tuple<std::string> nickname;
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    handler→convert(nickname);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdateRemoveEntity(
                                    std::move(std::get<0>(nickname)))));
}

void UpdateReceiver::_processCreateEntity() {
    handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
    msgpack::type::tuple<GameType::Entity, std::string, int32_t> entityData;
    handler→convert(entityData);
    if (std::get<0>(entityData) ≠ GameType::PLAYER) {
        EntityData data = protocol.processAddNPC(&buffer, entityData, offset);
        currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdateCreateNPC(data
)));
    } else {
        MapPlayerData data = protocol.processAddPlayer(&buffer, entityData, offs
et);
        currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdateCreatePlayer(d
ata)));
    }
}

/* Recibe la informacion del jugador para poder mostrarla en la interfaz grafica
 */
void UpdateReceiver::_receivePlayerData() {
    uint32_t length = 0;
    socket.receive(reinterpret_cast<char*>(&length), sizeof(uint32_t));
    length = ntohl(length);
    buffer.clear();
    buffer.resize(length);
    socket.receive(buffer.data(), length);
    PlayerData data = protocol.processAddPlayerData(&buffer);
    currentUpdate.push(std::unique_ptr<UpdateEvent>(new UpdateGUI(std::move(data
))));
}
```

```cpp
//
// Created by marcos on 13/7/20.
//

#ifndef ARGENTUM_UPDATEMANAGER_H
#define ARGENTUM_UPDATEMANAGER_H

#include "Update.h"
#include "../UpdateEvents/UpdateEvent.h"
#include <mutex>
#include <memory>

/*Almacena los udpates que recibe del server, los cuales serian consumidos
 * por el thread principal en el main game loop*/

class UpdateManager {
private:
    std::mutex m;
    std::queue<Update> updates;

public:
    /*Agrega un update*/
    void push(Update& update);

    /*Retorna el siguiente update, eliminandolo de la cola de updates*/
    Update pop();

    /*Retorna la cantidad de updates disponibles (cantidad en la cola)*/
    int updatesAvailable();
};


#endif //ARGENTUM_UPDATEMANAGER_H
```

```cpp
//
// Created by marcos on 13/7/20.
//

#include "UpdateManager.h"
#include "../../libs/TPException.h"

void UpdateManager::push(Update& update) {
    std::lock_guard<std::mutex> l(m);
    updates.emplace(std::move(update));
}

Update UpdateManager::pop() {
    std::lock_guard<std::mutex> l(m);
    if (¬updates.empty()) {
        auto update = std::move(updates.front());
        updates.pop();
        return update;
    }
    throw TPException("An update was requested from an empty queue!");
}

int UpdateManager::updatesAvailable() {
    std::lock_guard<std::mutex> l(m);
    return updates.size();
}
```

```
//
// Created by marcos on 6/29/20.
//

#ifndef ARGENTUM_UPDATE_H
#define ARGENTUM_UPDATE_H

#include <queue>
#include <memory>
#include "../UpdateEvents/UpdateEvent.h"

class Update {
private:
    std::queue<std::unique_ptr<UpdateEvent>> events;

public:
    Update() = default;
    Update(Update∧ other) noexcept;
    void push(std::unique_ptr<UpdateEvent> element);
    std::unique_ptr<UpdateEvent> pop();
    bool empty();
};

#endif //ARGENTUM_UPDATE_H
```

```
//
// Created by marcos on 19/7/20.
//

#include "Update.h"

void Update::push(std::unique_ptr<UpdateEvent> element) {
    events.push(std::move(element));
}

std::unique_ptr<UpdateEvent> Update::pop() {
    std::unique_ptr<UpdateEvent> element = std::move(events.front());
    events.pop();
    return element;
}

bool Update::empty() {
    return events.empty();
}

Update::Update(Update∧ other) noexcept {
    events = std::move(other.events);
}
```

```cpp
//
// Created by ivan on 25/6/20.
//

#ifndef ARGENTUM_PROTOCOLENUMTRANSLATOR_H
#define ARGENTUM_PROTOCOLENUMTRANSLATOR_H

#include "../../libs/GameEnums.h"
#include "../Texture/TextureRepository.h"
#include <unordered_map>

struct FloorTypeTexture{
    TextureID texture;
    int index;
};

class ProtocolEnumTranslator {
private:
    std::unordered_map<GameType::Entity,TextureID> entitiesMap;
    std::unordered_map<GameType::FloorType,FloorTypeTexture> floorTypesMap;
    std::unordered_map<GameType::Structure,TextureID> structuresMap;
    std::unordered_map<GameType::Race,TextureID> racesMap;
    std::unordered_map<GameType::Weapon,TextureID> weaponsMap;
    std::unordered_map<GameType::Weapon,TextureID> weaponDropsMap;
    std::unordered_map<GameType::Clothing,TextureID> clothingMap;
    std::unordered_map<GameType::Clothing,TextureID> clothingDropsMap;
    std::unordered_map<GameType::Potion,TextureID>potionsMap;

public:
    ProtocolEnumTranslator();

    /* Devuelve el id de la textura perteneciente a "entity" */
    TextureID getEntityTexture(GameType::Entity entity);

    /* Devuelve el id de la textura perteneciente a "floorType". Tambien recibe el
      * indice para saber que tile de la textura usar */
    FloorTypeTexture getFloorTypeTexture(GameType::FloorType floorType);

    /* Devuelve el id de la textura perteneciente a "structure" */
    TextureID getStructureTexture(GameType::Structure structure);

    /* Devuelve el id de la textura perteneciente a la raza "race" */
    TextureID getRaceTexture(GameType::Race race);

    /* Devuelve el id de la textura perteneciente a "weapon". Esta es la textura
      * que se usa para mostrar equipada en el jugador */
    TextureID getWeaponTexture(GameType::Weapon weapon);

    /* Devuelve el id de la textura perteneciente a "weapon". Esta es la textura
      * que se usa para mostrar como drop y en el inventario */
    TextureID getWeaponDropTexture(GameType::Weapon weapon);

    /* Devuelve el id de la textura perteneciente a la vestimenta "clothing". Esta es la textura
      * que se usa para mostrar equipada en el jugador */
    TextureID getClothingTexture(GameType::Clothing clothing);

    /* Devuelve el id de la textura perteneciente a "clothing". Esta es la textura
    * que se usa para mostrar como drop y en el inventario */
    TextureID getClothingDropTexture(GameType::Clothing clothing);

    /* Devuelve el id de la textura perteneciente a "potion" */
    TextureID getPotionTexture(GameType::Potion potion);

```

```cpp
    ~ProtocolEnumTranslator();

private:
    void _translateEntities();
    void _translateFloorTypes();
    void _translateStructures();
    void _translateRaces();
    void _translateWeapons();
    void _translateWeaponDrops();
    void _translateClothing();
    void _translateClothingDrops();
    void _translatePotions();
};


#endif //ARGENTUM_PROTOCOLENUMTRANSLATOR_H
```

```cpp
//
// Created by ivan on 25/6/20.
//

#include "ProtocolEnumTranslator.h"

ProtocolEnumTranslator::ProtocolEnumTranslator() {
    _translateEntities();
    _translateFloorTypes();
    _translateStructures();
    _translateRaces();
    _translateWeapons();
    _translateWeaponDrops();
    _translateClothing();
    _translateClothingDrops();
    _translatePotions();
}

void ProtocolEnumTranslator::_translateEntities(){
    entitiesMap.emplace(GameType::Entity::SKELETON, Skeleton);
    entitiesMap.emplace(GameType::Entity::ZOMBIE, Zombie);
    entitiesMap.emplace(GameType::Entity::SPIDER, Spider);
    entitiesMap.emplace(GameType::Entity::GOBLIN, Goblin);
    entitiesMap.emplace(GameType::Entity::BANKER, Banker);
    entitiesMap.emplace(GameType::Entity::GUARD, Guard);
    entitiesMap.emplace(GameType::Entity::TRADER, Trader);
    entitiesMap.emplace(GameType::Entity::PRIEST, Priest);
    entitiesMap.emplace(GameType::Entity::NOTHING, Nothing);
}

void ProtocolEnumTranslator::_translateFloorTypes() {
    floorTypesMap.emplace(GameType::FloorType::GRASS0,FloorTypeTexture{Grass,0});
    floorTypesMap.emplace(GameType::FloorType::GRASS1, FloorTypeTexture{Grass,1});
    floorTypesMap.emplace(GameType::FloorType::GRASS2, FloorTypeTexture{Grass,2});
    floorTypesMap.emplace(GameType::FloorType::GRASS3, FloorTypeTexture{Grass,3});
    floorTypesMap.emplace(GameType::FloorType::SAND, FloorTypeTexture{Sand,0});
    floorTypesMap.emplace(GameType::FloorType::WATER0, FloorTypeTexture{Water,0});
    floorTypesMap.emplace(GameType::FloorType::WATER1, FloorTypeTexture{Water,1});
    floorTypesMap.emplace(GameType::FloorType::WATER2, FloorTypeTexture{Water,2});
    floorTypesMap.emplace(GameType::FloorType::WATER3, FloorTypeTexture{Water,3});
    floorTypesMap.emplace(GameType::FloorType::PRETTY_ROAD0, FloorTypeTexture{PrettyRoad,0});
    floorTypesMap.emplace(GameType::FloorType::PRETTY_ROAD1, FloorTypeTexture{PrettyRoad,1});
    floorTypesMap.emplace(GameType::FloorType::PRETTY_ROAD2, FloorTypeTexture{PrettyRoad,2});
    floorTypesMap.emplace(GameType::FloorType::PRETTY_ROAD3, FloorTypeTexture{PrettyRoad,3});
    floorTypesMap.emplace(GameType::FloorType::PRETTY_GRASS0, FloorTypeTexture{PrettyGrass,0});
    floorTypesMap.emplace(GameType::FloorType::PRETTY_GRASS1, FloorTypeTexture{PrettyGrass,1});
    floorTypesMap.emplace(GameType::FloorType::PRETTY_GRASS2, FloorTypeTexture{PrettyGrass,2});
    floorTypesMap.emplace(GameType::FloorType::PRETTY_GRASS3, FloorTypeTexture{PrettyGrass,3});
    floorTypesMap.emplace(GameType::FloorType::DEAD_GRASS0, FloorTypeTexture{DeadGrass,0});
```

```cpp
    floorTypesMap.emplace(GameType::FloorType::DEAD_GRASS1, FloorTypeTexture{DeadGrass,1});
    floorTypesMap.emplace(GameType::FloorType::DEAD_GRASS2, FloorTypeTexture{DeadGrass,2});
    floorTypesMap.emplace(GameType::FloorType::DEAD_GRASS3, FloorTypeTexture{DeadGrass,3});
    floorTypesMap.emplace(GameType::FloorType::DARK_WATER0, FloorTypeTexture{DarkWater,0});
    floorTypesMap.emplace(GameType::FloorType::DARK_WATER1, FloorTypeTexture{DarkWater,1});
    floorTypesMap.emplace(GameType::FloorType::DARK_WATER2, FloorTypeTexture{DarkWater,2});
    floorTypesMap.emplace(GameType::FloorType::DARK_WATER3, FloorTypeTexture{DarkWater,3});
}

void ProtocolEnumTranslator::_translateStructures() {
    structuresMap.emplace(GameType::Structure::BONE_GUY, BoneGuy);
    structuresMap.emplace(GameType::Structure::BROKEN_RIP_STONE, BrokenRipStone);
    structuresMap.emplace(GameType::Structure::BUSH, Bush);
    structuresMap.emplace(GameType::Structure::DEAD_BUSH, DeadBush);
    structuresMap.emplace(GameType::Structure::DEAD_GUY, DeadGuy);
    structuresMap.emplace(GameType::Structure::DEAD_TREE, DeadTree);
    structuresMap.emplace(GameType::Structure::FAT_TREE, FatTree);
    structuresMap.emplace(GameType::Structure::HANGED_GUY, HangedGuy);
    structuresMap.emplace(GameType::Structure::HOUSE1, House1);
    structuresMap.emplace(GameType::Structure::HOUSE2, House2);
    structuresMap.emplace(GameType::Structure::HOUSE3, House3);
    structuresMap.emplace(GameType::Structure::LONG_TREE, LongTree);
    structuresMap.emplace(GameType::Structure::PALM_TREE, PalmTree);
    structuresMap.emplace(GameType::Structure::RIP_STONE, RipStone);
    structuresMap.emplace(GameType::Structure::TREE, Tree);
    structuresMap.emplace(GameType::Structure::VERY_DEAD_GUY, VeryDeadGuy);
    structuresMap.emplace(GameType::Structure::SUNKEN_COLUMN, SunkenColumn);
    structuresMap.emplace(GameType::Structure::SUNKEN_SHIP, SunkenShip);
    structuresMap.emplace(GameType::Structure::NO_STRUCTURE, Nothing);
}

void ProtocolEnumTranslator::_translateRaces() {
    racesMap.emplace(GameType::Race::HUMAN, HumanHead);
    racesMap.emplace(GameType::Race::ELF, ElfHead);
    racesMap.emplace(GameType::Race::DWARF, DwarfHead);
    racesMap.emplace(GameType::Race::GNOME, GnomeHead);

}

void ProtocolEnumTranslator::_translateWeapons() {
    weaponsMap.emplace(GameType::Weapon::LONGSWORD, LongSword);
    weaponsMap.emplace(GameType::Weapon::AXE, Axe);
    weaponsMap.emplace(GameType::Weapon::WARHAMMER, WarHammer);
    weaponsMap.emplace(GameType::Weapon::ASH_ROD, AshRod);
    weaponsMap.emplace(GameType::Weapon::ELVEN_FLUTE, Nothing);
    weaponsMap.emplace(GameType::Weapon::LINKED_STAFF, LinkedStaff);
    weaponsMap.emplace(GameType::Weapon::SIMPLE_BOW, SimpleBow);
    weaponsMap.emplace(GameType::Weapon::COMPOSITE_BOW, CompositeBow);
    weaponsMap.emplace(GameType::Weapon::GNARLED_STAFF, GnarledStaff);
    weaponsMap.emplace(GameType::Weapon::FIST, Nothing);
}

void ProtocolEnumTranslator::_translateWeaponDrops() {
    weaponDropsMap.emplace(GameType::Weapon::LONGSWORD, LongSwordDrop);
    weaponDropsMap.emplace(GameType::Weapon::AXE, AxeDrop);
    weaponDropsMap.emplace(GameType::Weapon::WARHAMMER, WarHammerDrop);
    weaponDropsMap.emplace(GameType::Weapon::ASH_ROD, AshRodDrop);
    weaponDropsMap.emplace(GameType::Weapon::ELVEN_FLUTE, ElvenFluteDrop);
    weaponDropsMap.emplace(GameType::Weapon::LINKED_STAFF, LinkedStaffDrop);
    weaponDropsMap.emplace(GameType::Weapon::SIMPLE_BOW, SimpleBowDrop);
    weaponDropsMap.emplace(GameType::Weapon::COMPOSITE_BOW, CompositeBowDrop);
    weaponDropsMap.emplace(GameType::Weapon::GNARLED_STAFF, GnarledStaffDrop);
    weaponDropsMap.emplace(GameType::Weapon::FIST, Nothing);
}

void ProtocolEnumTranslator::_translateClothing() {
    clothingMap.emplace(GameType::Clothing::COMMON_CLOTHING, CommonClothing);
    clothingMap.emplace(GameType::Clothing::LEATHER_ARMOR, LeatherArmor);
    clothingMap.emplace(GameType::Clothing::PLATE_ARMOR, PlateArmor);
    clothingMap.emplace(GameType::Clothing::KING_ARMOR, KingArmor);
    clothingMap.emplace(GameType::Clothing::BLUE_TUNIC, BlueTunic);
    clothingMap.emplace(GameType::Clothing::HOOD, Hood);
    clothingMap.emplace(GameType::Clothing::IRON_HELMET, IronHelmet);
    clothingMap.emplace(GameType::Clothing::TURTLE_SHIELD, TurtleShield);
    clothingMap.emplace(GameType::Clothing::IRON_SHIELD, IronShield);
    clothingMap.emplace(GameType::Clothing::MAGIC_HAT, MagicHat);
    clothingMap.emplace(GameType::Clothing::NO_HELMET, Nothing);
    clothingMap.emplace(GameType::Clothing::NO_SHIELD, Nothing);
}

void ProtocolEnumTranslator::_translateClothingDrops() {
    clothingDropsMap.emplace(GameType::Clothing::COMMON_CLOTHING,CommonClothingD
rop);
    clothingDropsMap.emplace(GameType::Clothing::LEATHER_ARMOR, LeatherArmorDrop
);
    clothingDropsMap.emplace(GameType::Clothing::PLATE_ARMOR, PlateArmorDrop);
    clothingDropsMap.emplace(GameType::Clothing::KING_ARMOR, KingArmorDrop);
    clothingDropsMap.emplace(GameType::Clothing::BLUE_TUNIC, BlueTunicDrop);
    clothingDropsMap.emplace(GameType::Clothing::HOOD, HoodDrop);
    clothingDropsMap.emplace(GameType::Clothing::IRON_HELMET, IronHelmetDrop);
    clothingDropsMap.emplace(GameType::Clothing::TURTLE_SHIELD, TurtleShieldDrop
);
    clothingDropsMap.emplace(GameType::Clothing::IRON_SHIELD, IronShieldDrop);
    clothingDropsMap.emplace(GameType::Clothing::MAGIC_HAT, MagicHatDrop);
    clothingDropsMap.emplace(GameType::Clothing::NO_HELMET, Nothing);
    clothingDropsMap.emplace(GameType::Clothing::NO_SHIELD, Nothing);
}

void ProtocolEnumTranslator::_translatePotions() {
    potionsMap.emplace(GameType::Potion::HEALTH_POTION, HealthPotion);
    potionsMap.emplace(GameType::Potion::MANA_POTION, ManaPotion);
}

TextureID ProtocolEnumTranslator::getEntityTexture(GameType::Entity entity) {
    return entitiesMap.at(entity);
}

TextureID ProtocolEnumTranslator::getStructureTexture(GameType::Structure struct
ure){
    return structuresMap.at(structure);
}
TextureID ProtocolEnumTranslator::getRaceTexture(GameType::Race race){
    return racesMap.at(race);
}
TextureID ProtocolEnumTranslator::getWeaponTexture(GameType::Weapon weapon){
    return weaponsMap.at(weapon);
}
TextureID ProtocolEnumTranslator::getWeaponDropTexture(GameType::Weapon weapon){
    return weaponDropsMap.at(weapon);
}
TextureID ProtocolEnumTranslator::getClothingTexture(GameType::Clothing clothing
){
    return clothingMap.at(clothing);
}
```

```
TextureID ProtocolEnumTranslator::getClothingDropTexture(GameType::Clothing clot
hing){
    return clothingDropsMap.at(clothing);
}
TextureID ProtocolEnumTranslator::getPotionTexture(GameType::Potion potion){
    return potionsMap.at(potion);
}

FloorTypeTexture ProtocolEnumTranslator::getFloorTypeTexture(GameType::FloorType
 floorType) {
    return floorTypesMap.at(floorType);
}

ProtocolEnumTranslator::~ProtocolEnumTranslator() = default;
```

```cpp
#include "Client/ArgentumClientSide.h"
#include "../libs/TPException.h"
#include <iostream>

int main(int argc, char** argv) {
    try {
        ArgentumClientSide::run(argc);
    } catch (TPException& e) {
        std::cerr << e.what() << " in Client!" << std::endl;
    } catch (...) {
        std::cerr << "Uknown error in Client!" << std::endl;
    }
}
```

```cpp
//
// Created by ivan on 10/7/20.
//

#ifndef ARGENTUM_WITHDRAWCOMMAND_H
#define ARGENTUM_WITHDRAWCOMMAND_H

#include "InputCommand.h"
#include "../../Map/Coordinate.h"

class WithdrawCommand : public InputCommand{
private:
    Coordinate tile;
    std::string item;

public:
    explicit WithdrawCommand(Coordinate tile, std::string^ item) : tile(tile),
item(item) {}
    void operator()(std::stringstream& msgBuffer) override;
};


#endif //ARGENTUM_WITHDRAWCOMMAND_H
```

```
//
// Created by ivan on 10/7/20.
//

#include "WithdrawCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)


void WithdrawCommand::operator()(std::stringstream &msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_WITHDRAW)
;
    msgpack::type::tuple<std::string, int32_t, int32_t> depositInfo;
    depositInfo = {item, tile.i, tile.j};
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, depositInfo);
}
```

```
//
// Created by ivan on 9/7/20.
//

#ifndef ARGENTUM_SELLCOMMAND_H
#define ARGENTUM_SELLCOMMAND_H

#include "InputCommand.h"
#include "../../Map/Coordinate.h"

class SellCommand : public InputCommand {
private:
    Coordinate tile;
    std::string item;

public:
    explicit SellCommand(Coordinate tile, std::string^ item) : tile(tile), item
(item) {}
    void operator()(std::stringstream& msgBuffer) override;
};



#endif //ARGENTUM_SELLCOMMAND_H
```

```
//
// Created by ivan on 9/7/20.
//

#include "SellCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)


void SellCommand::operator()(std::stringstream &msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_SELL);
    msgpack::type::tuple<std::string, int32_t, int32_t> saleInfo;
    saleInfo = {item, tile.i, tile.j};
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, saleInfo);
}
```

```
//
// Created by ivan on 9/7/20.
//

#ifndef ARGENTUM_RESURRECTCOMMAND_H
#define ARGENTUM_RESURRECTCOMMAND_H

#include "InputCommand.h"
#include "../../Map/Coordinate.h"

class ResurrectCommand : public InputCommand {
private:
    Coordinate tile;

public:
    explicit ResurrectCommand(Coordinate tile) : tile(tile) {}
    void operator()(std::stringstream& msgBuffer) override;
};

#endif //ARGENTUM_RESURRECTCOMMAND_H
```

```
//
// Created by ivan on 9/7/20.
//

#include "ResurrectCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)


void ResurrectCommand::operator()(std::stringstream &msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_RESURRECT);
    msgpack::type::tuple<int32_t, int32_t> selectedTile;
    selectedTile = {tile.i, tile.j};
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, selectedTile);
}
```

```
//
// Created by ivan on 13/7/20.
//

#ifndef ARGENTUM_REQUESTINVENTORYNAMESCOMMAND_H
#define ARGENTUM_REQUESTINVENTORYNAMESCOMMAND_H

#include "InputCommand.h"

class RequestInventoryNamesCommand : public InputCommand{
public:
    void operator()(std::stringstream& msgBuffer) override;
};


#endif //ARGENTUM_REQUESTINVENTORYNAMESCOMMAND_H
```

```
//
// Created by ivan on 13/7/20.
//

#include "RequestInventoryNamesCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)

void RequestInventoryNamesCommand::operator()(std::stringstream &msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_REQUEST_I
NVENTORY_NAMES);
    msgpack::pack(msgBuffer, event);
}
```

```
//
// Created by ivan on 9/7/20.
//

#ifndef ARGENTUM_PICKUPCOMMAND_H
#define ARGENTUM_PICKUPCOMMAND_H

#include "InputCommand.h"
#include "../../Map/Coordinate.h"

class PickUpCommand : public InputCommand{
public:
    void operator()(std::stringstream& msgBuffer) override;
};


#endif //ARGENTUM_PICKUPCOMMAND_H
```

```cpp
//
// Created by ivan on 9/7/20.
//

#include "PickUpCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)

void PickUpCommand::operator()(std::stringstream &msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_PICK_UP);
    msgpack::pack(msgBuffer, event);
}
```

```cpp
//
// Created by ivan on 10/7/20.
//

#ifndef ARGENTUM_MESSAGETOPLAYERCOMMAND_H
#define ARGENTUM_MESSAGETOPLAYERCOMMAND_H
#include "InputCommand.h"
#include "../../Map/Coordinate.h"

class MessageToPlayerCommand : public InputCommand{
private:
    std::string nickname, msg;
public:
    MessageToPlayerCommand(std::string∧ nickname, std::string∧ msg) : nickname(
nickname),
    msg(msg) {}
    void operator()(std::stringstream& msgBuffer) override;

};


#endif //ARGENTUM_MESSAGETOPLAYERCOMMAND_H
```

jul 21, 20 15:47 **MessageToPlayerCommand.cpp** Page 1/1

```cpp
//
// Created by ivan on 10/7/20.
//

#include "MessageToPlayerCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)


void MessageToPlayerCommand::operator()(std::stringstream &msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_SEND_MSG)
;
    msgpack::type::tuple<std::string, std::string> msgInfo;
    msgInfo = {nickname, msg};
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, msgInfo);
}
```

jul 21, 20 15:47 **MeditateCommand.h** Page 1/1

```cpp
//
// Created by ivan on 7/7/20.
//

#ifndef ARGENTUM_MEDITATECOMMAND_H
#define ARGENTUM_MEDITATECOMMAND_H

#include "InputCommand.h"

class MeditateCommand : public InputCommand{
public:
    void operator()(std::stringstream& msgBuffer) override;
};


#endif //ARGENTUM_MEDITATECOMMAND_H
```

```
//
// Created by ivan on 7/7/20.
//

#include "MeditateCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)


void MeditateCommand::operator()(std::stringstream& msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_MEDITATE)
;
    msgpack::pack(msgBuffer, event);
}
```

```
//
// Created by ivan on 9/7/20.
//

#ifndef ARGENTUM_LISTCOMMAND_H
#define ARGENTUM_LISTCOMMAND_H

#include "InputCommand.h"
#include "../../Map/Coordinate.h"

class ListCommand : public InputCommand{
private:
    Coordinate tile;

public:
    explicit ListCommand(Coordinate tile) : tile(tile) {}
    void operator()(std::stringstream& msgBuffer) override;
};


#endif //ARGENTUM_LISTCOMMAND_H
```

```cpp
//
// Created by ivan on 9/7/20.
//

#include "ListCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)

void ListCommand::operator()(std::stringstream &msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_LIST);
    msgpack::type::tuple<int32_t, int32_t> selectedTile;
    selectedTile = {tile.i, tile.j};
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, selectedTile);
}
```

```cpp
//
// Created by ivan on 7/7/20.
//

#ifndef ARGENTUM_INPUTCOMMAND_H
#define ARGENTUM_INPUTCOMMAND_H

#include <sstream>
#include <msgpack.hpp>
#include "../../../libs/GameEnums.h"

// Interfaz de los comandos del minichat.

class InputCommand {
public:
    virtual void operator()(std::stringstream& msgBuffer) = 0;
    virtual ~InputCommand() = default;
};

#endif //ARGENTUM_INPUTCOMMAND_H
```

```
//
// Created by ivan on 10/7/20.
//

#ifndef ARGENTUM_HEALCOMMAND_H
#define ARGENTUM_HEALCOMMAND_H

#include "InputCommand.h"
#include "../../Map/Coordinate.h"

class HealCommand : public InputCommand{
private:
    Coordinate tile;

public:
    explicit HealCommand(Coordinate tile) : tile(tile) {}
    void operator()(std::stringstream& msgBuffer) override;
};


#endif //ARGENTUM_HEALCOMMAND_H
```

```
//
// Created by ivan on 10/7/20.
//

#include "HealCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)


void HealCommand::operator()(std::stringstream &msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_HEAL);
    msgpack::type::tuple<int32_t, int32_t> selectedTile;
    selectedTile = {tile.i, tile.j};
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, selectedTile);
}
```

```cpp
//
// Created by ivan on 9/7/20.
//

#ifndef ARGENTUM_DROPCOMMAND_H
#define ARGENTUM_DROPCOMMAND_H

#include "InputCommand.h"
#include "../../Map/Coordinate.h"

class DropCommand : public InputCommand{
private:
    int32_t slot;
public:
    explicit DropCommand(int32_t slot) : slot(slot) {}
    void operator()(std::stringstream& msgBuffer) override;
};


#endif //ARGENTUM_DROPCOMMAND_H
```

```cpp
//
// Created by ivan on 9/7/20.
//

#include "DropCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)

void DropCommand::operator()(std::stringstream &msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_DROP);
    msgpack::type::tuple<int32_t> inventorySlot;
    inventorySlot = slot;
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, inventorySlot);
}
```

jul 21, 20 15:47 **DepositCommand.h** Page 1/1

```cpp
//
// Created by ivan on 10/7/20.
//

#ifndef ARGENTUM_DEPOSITCOMMAND_H
#define ARGENTUM_DEPOSITCOMMAND_H

#include "InputCommand.h"
#include "../../Map/Coordinate.h"

class DepositCommand : public InputCommand{
private:
    Coordinate tile;
    std::string item;

public:
    explicit DepositCommand(Coordinate tile, std::string^ item) : tile(tile), item(item) {}
    void operator()(std::stringstream& msgBuffer) override;
};


#endif //ARGENTUM_DEPOSITCOMMAND_H
```

jul 21, 20 15:47 **DepositCommand.cpp** Page 1/1

```cpp
//
// Created by ivan on 10/7/20.
//

#include "DepositCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)

void DepositCommand::operator()(std::stringstream &msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_DEPOSIT);
    msgpack::type::tuple<std::string, int32_t, int32_t> depositInfo;
    depositInfo = {item, tile.i, tile.j};
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, depositInfo);
}
```

```
//
// Created by ivan on 7/7/20.
//

#ifndef ARGENTUM_COMMANDVERIFIER_H
#define ARGENTUM_COMMANDVERIFIER_H

#include "../../../libs/GameEnums.h"
#include "InputCommand.h"
#include <memory>
#include <unordered_map>

class GameGUI;

class CommandVerifier {
private:
    std::unordered_map<std::string, GameType::PlayerEvent> commands;
    std::string input;

public:
    /* Constructor */
    CommandVerifier();
    /* Verifica que comando ingrese y devuelve el functor */
    std::unique_ptr<InputCommand> verifyCommand(GameGUI& game, std::string∧ cmd
);

private:
    void _initCommands();
    std::unique_ptr<InputCommand> _processMeditate();
    std::unique_ptr<InputCommand> _processPickUp();
    std::unique_ptr<InputCommand> _processDrop(GameGUI& game);
    std::unique_ptr<InputCommand> _processList(GameGUI &game);
    std::unique_ptr<InputCommand> _processResurrect(GameGUI &game);
    std::unique_ptr<InputCommand> _processSell(GameGUI &game);
    std::unique_ptr<InputCommand> _processBuy(GameGUI &game);
    std::unique_ptr<InputCommand> _processHeal(GameGUI& game);
    std::unique_ptr<InputCommand> _processDeposit(GameGUI &game);
    std::unique_ptr<InputCommand> _processWithdraw(GameGUI &game);
    std::unique_ptr<InputCommand> _processSendMessageToPlayer();
    std::unique_ptr<InputCommand> _processRequestInventoryNames();

    static void _processGold(std::string &parameter);

};


#endif //ARGENTUM_COMMANDVERIFIER_H
```

```
//
// Created by ivan on 7/7/20.
//

#include <sstream>
#include "CommandVerifier.h"
#include "MeditateCommand.h"
#include "PickUpCommand.h"
#include "DropCommand.h"
#include "ListCommand.h"
#include "SellCommand.h"
#include "BuyCommand.h"
#include "ResurrectCommand.h"
#include "HealCommand.h"
#include "DepositCommand.h"
#include "WithdrawCommand.h"
#include "MessageToPlayerCommand.h"
#include "RequestInventoryNamesCommand.h"
#include "../GameGUI.h"

CommandVerifier::CommandVerifier() {
    _initCommands();
}

/* Inicializa el unordered_map de comandos */
void CommandVerifier::_initCommands() {
    commands.emplace("/meditate", GameType::PLAYER_MEDITATE);
    commands.emplace("/revive", GameType::PLAYER_RESURRECT);
    commands.emplace("/heal", GameType::PLAYER_HEAL);
    commands.emplace("/deposit", GameType::PLAYER_DEPOSIT);
    commands.emplace("/withdraw", GameType::PLAYER_WITHDRAW);
    commands.emplace("/list", GameType::PLAYER_LIST);
    commands.emplace("/buy", GameType::PLAYER_BUY);
    commands.emplace("/sell", GameType::PLAYER_SELL);
    commands.emplace("/take", GameType::PLAYER_PICK_UP);
    commands.emplace("/drop", GameType::PLAYER_DROP);
    commands.emplace("/inventory", GameType::PLAYER_REQUEST_INVENTORY_NAMES);
}

std::unique_ptr<InputCommand> CommandVerifier::verifyCommand(GameGUI& game,
        std::string∧ inputCmd) {
    std::unique_ptr<InputCommand> command;
    input = inputCmd;

    //Agarro lo que tenga antes de un espacio. Eso deberia ser el comando
    std::string cmd = input.substr(0, input.find(' ', 0));
    GameType::PlayerEvent event;
    if (cmd.front() ≡ '@') {//Antes de ver si es un comando veo si es un nickna
me
        command = _processSendMessageToPlayer();
    } else {
        try {
            event = commands.at(cmd);
            switch (event) {
                case GameType::PLAYER_PICK_UP:
                    command = _processPickUp();
                    break;
                case GameType::PLAYER_DROP:
                    command = _processDrop(game);
                    break;
                case GameType::PLAYER_LIST:
                    command = _processList(game);
                    break;
                case GameType::PLAYER_RESURRECT:
                    command = _processResurrect(game);
                    break;
                case GameType::PLAYER_HEAL:
                    command = _processHeal(game);
                    break;
                case GameType::PLAYER_BUY:
                    command = _processBuy(game);
                    break;
                case GameType::PLAYER_SELL:
                    command = _processSell(game);
                    break;
                case GameType::PLAYER_WITHDRAW:
                    command = _processWithdraw(game);
                    break;
                case GameType::PLAYER_DEPOSIT:
                    command = _processDeposit(game);
                    break;
                case GameType::PLAYER_MEDITATE:
                    command = _processMeditate();
                    break;
                case GameType::PLAYER_REQUEST_INVENTORY_NAMES:
                    command = _processRequestInventoryNames();
                    break;
                default:
                    break;
            }
        } catch (std::exception& e) {
            //Si no encuentra el comando en el unordered_map es que no es un com
ando
            //valido asi que devuelvo nullptr
            return nullptr;
        }
    }
    return command;
}

std::unique_ptr<InputCommand> CommandVerifier::_processRequestInventoryNames() {
    //Chequeo que no haya nada escrito despues del comando
    if (input.size() > input.find(' ', 0)) {
        return nullptr;
    }
    return std::unique_ptr<InputCommand>(new RequestInventoryNamesCommand());
}

std::unique_ptr<InputCommand> CommandVerifier::_processMeditate() {
    //Chequeo que no haya nada escrito despues del comando
    if (input.size() > input.find(' ', 0)) {
        return nullptr;
    }
    return std::unique_ptr<InputCommand>(new MeditateCommand());
}

std::unique_ptr<InputCommand> CommandVerifier::_processPickUp() {
    //Chequeo que no haya nada escrito despues del comando
    if (input.size() > input.find(' ', 0)) {
        return nullptr;
    }
    return std::unique_ptr<InputCommand>(new PickUpCommand());
}

std::unique_ptr<InputCommand> CommandVerifier::_processDrop(GameGUI& game) {
    //Chequeo que no haya nada escrito despues del comando
    if (input.size() > input.find(' ', 0)) {
        return nullptr;
    }
    return std::unique_ptr<InputCommand>(new DropCommand(game.getSelector().getI
nventorySlot()));
}
```

```

std::unique_ptr<InputCommand> CommandVerifier::_processList(GameGUI& game) {
    //Chequeo que no haya nada escrito despues del comando
    if (input.size() > input.find(' ', 0)) {
        return nullptr;
    }
    return std::unique_ptr<InputCommand>(new ListCommand(game.getSelector().getS
electedTile()));
}

std::unique_ptr<InputCommand> CommandVerifier::_processResurrect(GameGUI& game)
{
    //Chequeo que no haya nada escrito despues del comando
    if (input.size() > input.find(' ', 0)) {
        return nullptr;
    }
    return std::unique_ptr<InputCommand>(new ResurrectCommand(game.getSelector()
.getSelectedTile()));
}

std::unique_ptr<InputCommand> CommandVerifier::_processHeal(GameGUI& game) {
    //Chequeo que no haya nada escrito despues del comando
    if (input.size() > input.find(' ', 0)) {
        return nullptr;
    }
    return std::unique_ptr<InputCommand>(new HealCommand(game.getSelector().getS
electedTile()));
}

std::unique_ptr<InputCommand> CommandVerifier::_processSell(GameGUI& game) {
    std::string parameters;
    if (input.size() > input.find(' ', 0)) {
        //Agarro lo que haya dsps del espacio que deberia ser el item que quiero
 vender
        parameters = input.substr(input.find(' ', 0) + 1, input.size());
        if (¬parameters.empty()) {
            return std::unique_ptr<InputCommand>(new SellCommand(
                    game.getSelector().getSelectedTile(), std::move(parameters))
);
        }
    }
    return nullptr;
}

std::unique_ptr<InputCommand> CommandVerifier::_processBuy(GameGUI& game) {
    std::string parameters;
    if (input.size() > input.find(' ', 0)) {
        //Agarro lo que haya dsps del espacio que deberian ser los parametros
        parameters = input.substr(input.find(' ', 0) + 1, input.size());
        if (¬parameters.empty()) {
            return std::unique_ptr<InputCommand>(new BuyCommand(
                    game.getSelector().getSelectedTile(), std::move(parameters))
);
        }
    }
    return nullptr;
}

std::unique_ptr<InputCommand> CommandVerifier::_processDeposit(GameGUI& game) {
    std::string parameters;
    int separator = input.find(' ', 0);
    if ((int)input.size() > separator ∧ separator ≠ −1) {
        //Agarro lo que haya dsps del espacio que deberian ser los parametros
        parameters = input.substr(separator + 1, input.size());
        //Como para el gold tambien necesito una cantidad lo proceso distinto a
un item
```

```
        if (parameters.find("Gold", 0) ≠ std::string::npos) {
            _processGold(parameters);
        }
        if (¬parameters.empty()){
            return std::unique_ptr<InputCommand>(new DepositCommand(
                    game.getSelector().getSelectedTile(), std::move(parameters))
);
        }
    }
    return nullptr;
}

std::unique_ptr<InputCommand> CommandVerifier::_processWithdraw(GameGUI& game) {
    std::string parameters;
    int separator = input.find(' ', 0);
    if ((int)input.size() > separator ∧ separator ≠ −1) {
        //Agarro lo que haya dsps del espacio que deberian ser los parametros
        parameters = input.substr(separator + 1, input.size());
        //Como para el gold tambien necesito una cantidad lo proceso distinto a
un item
        if (parameters.find("Gold", 0) ≠ std::string::npos) {
            _processGold(parameters);
        }
        if (¬parameters.empty()){
            return std::unique_ptr<InputCommand>(new WithdrawCommand(
                    game.getSelector().getSelectedTile(), std::move(parameters))
);
        }
    }
    return nullptr;
}

void CommandVerifier::_processGold(std::string& parameter) {
    int separator = parameter.find(' ', 0);
    if ((int)parameter.size() > separator ∧ separator ≠ −1) {
        //Agarro la parte del string que deberia tener la cantidad de gold
        std::string goldAmount = parameter.substr(parameter.find(' ', 0) + 1,
                                                  parameter.size());
        try {
            std::stoi(goldAmount);
        } catch (std::exception &e) {
            parameter = "";//Si la cantidad no es un numero
        }
    } else {
        parameter = "";//Si no tengo una cantidad de oro
    }
}

std::unique_ptr<InputCommand> CommandVerifier::_processSendMessageToPlayer() {
    int separator = input.find(' ');
    if ((int)input.size() > separator ∧ separator ≠ −1) {
        std::string nickname = input.substr(1, separator−1);
        std::string msg = input.substr(separator + 1, input.size());
        if (¬msg.empty()) {
            return std::unique_ptr<InputCommand>(new MessageToPlayerCommand(
                    std::move(nickname), std::move(msg)));
        }
    }
    return nullptr;
}
```

```
//
// Created by ivan on 9/7/20.
//

#ifndef ARGENTUM_BUYCOMMAND_H
#define ARGENTUM_BUYCOMMAND_H

#include "InputCommand.h"
#include "../../Map/Coordinate.h"

class BuyCommand : public InputCommand{
private:
    Coordinate tile;
    std::string item;

public:
    explicit BuyCommand(Coordinate tile, std::string∧ item) : tile(tile), item(
item) {}
    void operator()(std::stringstream& msgBuffer) override;
};


#endif //ARGENTUM_BUYCOMMAND_H
```

```cpp
//
// Created by ivan on 9/7/20.
//

#include "BuyCommand.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)

void BuyCommand::operator()(std::stringstream &msgBuffer) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_BUY);
    msgpack::type::tuple<std::string, int32_t, int32_t> buyInfo;
    buyInfo = {item, tile.i, tile.j};
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, buyInfo);
}
```

```cpp
//
// Created by marcos on 7/1/20.
//

#ifndef ARGENTUM_GAMEINITIALIZER_H
#define ARGENTUM_GAMEINITIALIZER_H

#include <vector>
#include <string>
#include "../../libs/GameEnums.h"

class GameGUI;
class Socket;
class ClientProtocol;

class GameInitializer {
private:
    GameGUI& game;
    Socket& socket;
    ClientProtocol& protocol;

public:
    GameInitializer(GameGUI& _game, Socket& _socket, ClientProtocol& _protocol)
:
                    game(_game), socket(_socket), protocol(_protocol) {}

    /* Manda al servidor la informacion para crear un nuevo jugador */
    void createPlayer(const std::string &nickname, GameType::Race race,
                      GameType::Class _class);

    /* Manda al servidor la informacion para cargar un jugador */
    void loadPlayer(const std::string &nickname);

    /* Recibe el estado inicial del juego */
    void initializeGame();

private:
    void _receiveMapInfo();
    void _receiveCurrentGameState();
    void _loadMap(std::vector<char>& buffer);
    void _processAddEntity(std::vector<char>& buffer, std::size_t& offset);
    void _receivePlayerData();
};


#endif //ARGENTUM_GAMEINITIALIZER_H
```

```cpp
//
// Created by marcos on 7/1/20.
//

#include "GameInitializer.h"
#include <cstdint>
#include "../../libs/Socket.h"
#include <msgpack.hpp>
#include "GameGUI.h"
#include "ProtocolEnumTranslator.h"
#include "ClientProtocol.h"
#include "CitizenData.h"

MSGPACK_ADD_ENUM(GameType::PlayerEvent)
MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::Race)
MSGPACK_ADD_ENUM(GameType::Class)
MSGPACK_ADD_ENUM(GameType::Entity)
MSGPACK_ADD_ENUM(GameType::Structure)
MSGPACK_ADD_ENUM(GameType::FloorType)

void GameInitializer::initializeGame() {
    _receiveMapInfo();
    _receiveCurrentGameState();
}

/* Recibe la informacion del mapa */
void GameInitializer::_receiveMapInfo() {
    int32_t msgLength;
    socket.receive((char*)(&msgLength), sizeof(msgLength));
    msgLength = ntohl(msgLength);
    std::vector<char> buffer(msgLength);
    socket.receive(buffer.data(), buffer.size());
    _loadMap(buffer);
}

/* Recibe el estado inicial del juego */
void GameInitializer::_receiveCurrentGameState() {
    int32_t msgLength;
    socket.receive((char*)(&msgLength), sizeof(msgLength));
    msgLength = ntohl(msgLength);
    std::vector<char> buffer(msgLength);
    socket.receive(buffer.data(), msgLength);
    std::size_t offset = 0;
    msgpack::object_handle handler;

    while (offset < static_cast<size_t>(msgLength)) {
        handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
        msgpack::type::tuple<GameType::EventID> id;
        handler→convert(id);
        if (std::get<0>(id) ≡ GameType::CREATE_ITEM) {
            ItemData data = protocol.processAddItem(&buffer, offset);
            game.createItem(data.position, data.texture);
        } else if (std::get<0>(id) ≡ GameType::CREATE_ENTITY) {
            _processAddEntity(buffer, offset);
        }
    }
    game.getMap().update();
    _receivePlayerData();
}

/* Procesa la entidad que recibe del server y la agrega al juego */
void GameInitializer::_processAddEntity(std::vector<char>& buffer, std::size_t&
offset) {
    msgpack::object_handle handler = msgpack::unpack(buffer.data(), buffer.size(
), offset);
```

```cpp
    msgpack::type::tuple<GameType::Entity, std::string, int32_t> entityData;
    handler→convert(entityData);
    if (std::get<0>(entityData) ≠ GameType::PLAYER) {
        EntityData data = protocol.processAddNPC(&buffer, entityData, offset);
        game.addNPC(data);
    } else {
        MapPlayerData data = protocol.processAddPlayer(&buffer, entityData, offs
et);
        game.addPlayer(data);
    }
}

/* Carga las texturas de cada tile del mapa*/
void GameInitializer::_loadMap(std::vector<char>& buffer) {
    std::size_t offset = 0;
    msgpack::object_handle handler = msgpack::unpack(buffer.data(), buffer.size(
), offset);
    msgpack::type::tuple<int32_t, int32_t> mapSize;
    ProtocolEnumTranslator translator;
    handler→convert(mapSize);
    int rows = std::get<0>(mapSize);
    int columns = std::get<1>(mapSize);
    game.setMapSize(rows, columns);
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < columns; ++j) {
            handler = msgpack::unpack(buffer.data(), buffer.size(), offset);
            msgpack::type::tuple<GameType::FloorType, GameType::Structure,
            GameType::Entity, std::string> tileInfo;
            handler→convert(tileInfo);
            CitizenData citizen = {translator.getEntityTexture(std::get<2>(tileI
nfo)),
                                   std::get<3>(tileInfo)};
            game.loadTileData({i, j}, translator.getFloorTypeTexture(std::get<0>
(tileInfo)),
                              translator.getStructureTexture(std::get<1>(tileInf
o)),
                              citizen);
        }
    }
}

void GameInitializer::createPlayer(const std::string& nickname, GameType::Race r
ace,
                                   GameType::Class _class) {
    game.getMap().setPlayerNickname(nickname);//Para despues poder buscar la pos
icion del player en Map
    std::stringstream msgBuffer;
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::CREATE_PLAYER);
    msgpack::type::tuple<std::string, GameType::Race, GameType::Class> playerInf
o;
    playerInfo = {nickname, race, _class};
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, playerInfo);
    std::string aux = msgBuffer.str();
    uint32_t length = aux.size();
    length = htonl(aux.size());
    std::vector<char> sendBuffer(sizeof(uint32_t));
    ClientProtocol::loadBytes(sendBuffer, &length, sizeof(uint32_t));
    std::copy(aux.begin(), aux.end(), std::back_inserter(sendBuffer));
    socket.send(sendBuffer.data(), sendBuffer.size());
}

void GameInitializer::loadPlayer(const std::string& nickname) {
    game.getMap().setPlayerNickname(nickname);//Para despues poder buscar la pos
icion del player en Map
    std::stringstream msgBuffer;
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::LOAD_PLAYER);
    msgpack::type::tuple<std::string> playerInfo;
    playerInfo = {nickname};
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, playerInfo);
    std::string aux = msgBuffer.str();
    uint32_t length = aux.size();
    length = htonl(aux.size());
    std::vector<char> sendBuffer(sizeof(uint32_t));
    ClientProtocol::loadBytes(sendBuffer, &length, sizeof(uint32_t));
    std::copy(aux.begin(), aux.end(), std::back_inserter(sendBuffer));
    socket.send(sendBuffer.data(), sendBuffer.size());
}

/* Recibe toda la informacion inicial del jugador */
void GameInitializer::_receivePlayerData() {
    uint32_t length = 0;
    socket.receive(reinterpret_cast<char*>(&length), sizeof(uint32_t));
    length = ntohl(length);
    std::vector<char> buffer(length);
    socket.receive(buffer.data(), length);
    PlayerData data = protocol.processAddPlayerData(&buffer);
    game.getPlayerInventory().updateGold(data.generalInfo.gold, data.generalInfo
.safeGold);
    for (const auto & item : data.equippedItems) {
        game.getPlayerInventory().addEquipableItem(std::get<0>(item),
                                                   std::get<1>(item));
    }
    for (const auto & item : data.inventoryItems) {
        game.getPlayerInventory().addInventoryItem(std::get<0>(item),
                                                   std::get<1>(item));
    }
    game.setCameraOn(data.generalInfo.nickname);
    game.getPlayerInfo().update(data.generalInfo);
    game.getMinichat().receiveText(data.minichatText);
}
```

```
//
// Created by marcos on 6/25/20.
//

#ifndef ARGENTUM_GAMEGUI_H
#define ARGENTUM_GAMEGUI_H

#include "../Screen/Window.h"
#include "../Texture/TextureRepository.h"
#include "GameConstants.h"
#include "../Map/Map.h"
#include "../Graphics/Minichat/Minichat.h"
#include "../Graphics/Selector.h"
#include "../Graphics/GUI/PlayerInventoryGUI.h"
#include "../Graphics/GUI/PlayerInfoGUI.h"
#include "../Texture/PlayerEquipment.h"
#include "../Sound/SoundPlayer.h"
#include "EntityData.h"
#include "CitizenData.h"

class GameGUI {
private:
    Window screen;
    SDL_Rect camera{0, 0, DEFAULT_MAP_WIDTH, DEFAULT_MAP_HEIGHT};
    TextureRepository repo;
    SoundPlayer soundPlayer;
    Map map;
    Minichat minichat;
    Selector selector;
    PlayerInfoGUI infoGUI;
    PlayerInventoryGUI inventoryGUI;
    Texture& background;//Esto capaz es mejor ponerlo en window u otra clase

public:
    GameGUI();

    /* Setea el tamaÃ±o del mapa */
    void setMapSize(int rows, int columns);

    /* Carga la informacion del tile en "position" */
    void loadTileData(Coordinate position, FloorTypeTexture floor, TextureID str
ucture,
                      CitizenData& citizen);

    /* Ubica un item en el tile en "position" */
    void createItem(Coordinate position, TextureID itemTexture);

    /* Agrega un NPC al mapa*/
    void addNPC(EntityData& entityData);

    /* Agrega un NPC al mapa*/
    void addPlayer(MapPlayerData& playerData);

    /* Setea la camara en el jugador controlado por el usuario */
    void setCameraOn(std::string& playerNickname);

    /* Renderiza toda la interfaz grafica */
    void render();

    /* Actualiza el estado del mapa */
    void update(double timeStep);

    Window& getWindow();
    Minichat& getMinichat();
    Selector& getSelector();
    PlayerInfoGUI& getPlayerInfo();
    PlayerInventoryGUI& getPlayerInventory();
    Map& getMap();
    SoundPlayer& getSoundPlayer();
    TextureRepository &getTextureRepo();
};


#endif //ARGENTUM_GAMEGUI_H
```

```
//
// Created by marcos on 6/25/20.
//

#include "GameGUI.h"

void GameGUI::setMapSize(int rows, int columns) {
    map.setSize(rows, columns);
}

GameGUI::GameGUI() : repo(screen.getRenderer()), map(repo, camera, soundPlayer),
                    minichat(screen.getRenderer()), infoGUI(screen.getRenderer()
, soundPlayer)
                    ,inventoryGUI(repo, screen.getRenderer(), infoGUI),
                    background(repo.getTexture(Background)) {
}

void GameGUI::loadTileData(Coordinate position, FloorTypeTexture floor,
                                TextureID structure, CitizenData& citizen) {
    map.loadTileData(position, floor, structure, citizen);
}

void GameGUI::update(double timeStep) {
    map.update(timeStep);
}

void GameGUI::render() {
    screen.clear();
    screen.setViewport(ScreenViewport);
    background.render(0, 0);

    //Mapa
    screen.setViewport(MapViewport);
    map.render();

    //Inventario
    screen.setViewport(InventoryViewport);
    inventoryGUI.render(selector.getInventorySlot());

    //PlayerStats
    screen.setViewport(PlayerInfoViewport);
    infoGUI.render();

    //Minichat
    screen.setViewport(MinichatViewport);
    minichat.render();
    screen.show();
}

Window& GameGUI::getWindow() {
    return screen;
}

Minichat &GameGUI::getMinichat() {
    return minichat;
}

Selector &GameGUI::getSelector() {
    return selector;
}

PlayerInfoGUI &GameGUI::getPlayerInfo() {
    return infoGUI;
}

void GameGUI::addNPC(EntityData& entityData) {
```

```cpp
    map.addNPC(entityData);
}

void GameGUI::createItem(Coordinate position, TextureID itemTexture) {
    map.createItem(position, itemTexture);
}

void GameGUI::addPlayer(MapPlayerData& playerData) {
    map.addPlayer(playerData);
}

PlayerInventoryGUI &GameGUI::getPlayerInventory() {
    return inventoryGUI;
}

Map &GameGUI::getMap() {
    return map;
}

SoundPlayer &GameGUI::getSoundPlayer() {
    return soundPlayer;
}

TextureRepository& GameGUI::getTextureRepo() {
    return repo;
}

void GameGUI::setCameraOn(std::string& playerNickname) {
    map.setCameraOn(playerNickname);
}



```

```cpp
//
// Created by marcos on 6/7/20.
//

#ifndef ARGENTUM_GAMECONSTANTS_H
#define ARGENTUM_GAMECONSTANTS_H

//Dimension de los tiles
const int TILE_WIDTH = 128;
const int TILE_HEIGHT = 128;

const int TOTAL_HORIZONTAL_TILES = 100;
const int TOTAL_VERTICAL_TILES = 100;

const int VISIBLE_HORIZONTAL_TILES = 8;
const int VISIBLE_VERTICAL_TILES = 5;

//Map Viewport
const int DEFAULT_MAP_WIDTH = TILE_WIDTH*VISIBLE_HORIZONTAL_TILES;
const int DEFAULT_MAP_HEIGHT = TILE_HEIGHT*VISIBLE_VERTICAL_TILES;

//UI Viewport
const int DEFAULT_INVENTORY_WIDTH = 470;
const int DEFAULT_INVENTORY_HEIGHT = DEFAULT_MAP_HEIGHT + 296;

//Minichat Viewport
const int DEFAULT_MINICHAT_WIDTH = DEFAULT_MAP_WIDTH + 5;
const int DEFAULT_MINICHAT_HEIGHT = 210;

//PlayerStats Viewport
const int DEFAULT_PLAYER_INFO_WIDTH = DEFAULT_MAP_WIDTH;
const int DEFAULT_PLAYER_INFO_HEIGHT = 45;

//Dimension de la ventana
const int DEFAULT_SCREEN_WIDTH = DEFAULT_MAP_WIDTH + DEFAULT_INVENTORY_WIDTH;
const int DEFAULT_SCREEN_HEIGHT = DEFAULT_INVENTORY_HEIGHT;

//General para todas las barras
const int BAR_HEIGHT = 30;

//Barras individuales.
const int HEALTH_BAR_X_OFFSET = 25;
const int MANA_BAR_X_OFFSET = 385;
const int XP_BAR_X_OFFSET = 725;

//Dimension de lo que se muestra del mapa
const int LEVEL_WIDTH = TOTAL_HORIZONTAL_TILES * TILE_WIDTH;
const int LEVEL_HEIGHT = TOTAL_VERTICAL_TILES * TILE_HEIGHT;


#endif //ARGENTUM_GAMECONSTANTS_H
```

```
//
// Created by marcos on 7/2/20.
//

#ifndef ARGENTUM_ENTITYDATA_H
#define ARGENTUM_ENTITYDATA_H

#include <string>
#include "../Texture/TextureRepository.h"
#include "../Map/Coordinate.h"
#include "../../libs/GameEnums.h"
#include "../Graphics/GUI/PlayerStats.h"
#include "../Texture/PlayerEquipment.h"

/*La info de un player para cargar en el mapa, esto me llega cuando se
 * crea un nuevo player*/

struct EntityData {
    TextureID texture;
    std::string nickname;
    Coordinate pos;
    GameType::Direction currentDir;
    int32_t distanceMoved;
    int32_t level;
};

struct PlayerData {
    PlayerStats generalInfo;
    std::vector<std::tuple<TextureID, EquippedItems>> equippedItems;
    std::vector<std::tuple<TextureID, int>> inventoryItems;
    std::string minichatText;

    PlayerData(PlayerData∧ other) noexcept {
        this→generalInfo = other.generalInfo;
        this→inventoryItems = std::move(other.inventoryItems);
        this→equippedItems = std::move(other.equippedItems);
        this→minichatText = std::move(other.minichatText);
        other.generalInfo = {};
        other.inventoryItems.clear();
        other.equippedItems.clear();
    }

    PlayerData() = default;
};

struct MapPlayerData {
    EntityData entityData;
    GameType::Race race{};
    PlayerEquipment equipment{};
    bool isAlive{};
};

struct ItemData {
    Coordinate position;
    TextureID texture;
};

#endif //ARGENTUM_ENTITYDATA_H
```

```
//
// Created by ivan on 24/6/20.
//

#ifndef ARGENTUM_CLIENTPROTOCOL_H
#define ARGENTUM_CLIENTPROTOCOL_H

#include "../Graphics/GUI/PlayerInventoryGUI.h"
#include "../../libs/GameEnums.h"
#include <vector>
#include <msgpack.hpp>
#include "ProtocolEnumTranslator.h"
#include "../Texture/PlayerEquipment.h"
#include "../Map/Coordinate.h"
#include "../Graphics/GUI/PlayerStats.h"
#include "EntityData.h"

class Socket;

class ClientProtocol {
private:
    Socket& socket;
    ProtocolEnumTranslator translator;
    msgpack::object_handle handler;
    std::vector<char>* buffer{};

public:
    /* Constructor */
    explicit ClientProtocol(Socket& _socket) : socket(_socket) {}
    /* Procesa la informacion del jugador recibida por el server y la asigna a u
n MapPlayerData */
    MapPlayerData processAddPlayer(std::vector<char>* _buffer,
            msgpack::type::tuple<GameType::Entity, std::string, int32_t>& entity
Data,
            std::size_t& offset);
    /* Procesa la informacion de un entity recibida por el server y la asigna a
un EntityData */
    EntityData processAddNPC(std::vector<char>* _buffer, msgpack::type::tuple<Ga
meType::Entity,
            std::string, int32_t> &entityData, size_t &offset);
    /* Procesa la informacion de un Item recibida por el server y la asigna a un
 ItemData*/
    ItemData processAddItem(std::vector<char>* _buffer, std::size_t& offset);
    /* Procesa la informacion del inventario y las stats del jugador recibida po
r
     * el server y la asigna a un PlayerData */
    PlayerData processAddPlayerData(std::vector<char>* _buffer);
    /* Carga "loadBuffer" con "data" */
    static void loadBytes(std::vector<char> &loadBuffer, void *data, unsigned in
t size);


private:
    void _addManaData(PlayerData& data, size_t& offset);
    void _addHealthData(PlayerData& data, size_t& offset);
    void _addXPData(PlayerData& data, size_t& offset);
    void _addEquippedItems(PlayerData& info, size_t &offset);
    void _addClothing(PlayerData& info, size_t &offset, EquippedItems item);
    void _addWeapon(PlayerData& info, size_t &offset);
    void _fillInventory(PlayerData& info, size_t &offset);
    void _addItem(PlayerData& info, GameType::ItemType type, int32_t id, int pos
ition);
    void _addSkills(PlayerData& data, size_t &offset);
    void _addPosition(PlayerData& data, size_t &offset);
    void _addPlayerStats(PlayerData& data, size_t& offset);
    void _addInventoryItems(PlayerData& data, size_t& offset);
    void _addMinichatText(PlayerData &data, size_t &offset);
    void _addNickname(PlayerData &data, size_t &offset);
    void _loadAddPlayerGeneralInfo(msgpack::type::tuple<GameType::Entity,
            std::string, int32_t>& entityData, MapPlayerData& pData, std::size_t
& offset);
    void _loadAddPlayerEquipmentInfo(MapPlayerData& pData,
                                                        std::size_t& offset);
};


#endif //ARGENTUM_CLIENTPROTOCOL_H
```

```
//
// Created by ivan on 24/6/20.
//

#include "ClientProtocol.h"
#include "GameGUI.h"
#include "../../libs/Socket.h"

MSGPACK_ADD_ENUM(GameType::EventID)
MSGPACK_ADD_ENUM(GameType::Race)
MSGPACK_ADD_ENUM(GameType::FloorType)
MSGPACK_ADD_ENUM(GameType::Structure)
MSGPACK_ADD_ENUM(GameType::Entity)
MSGPACK_ADD_ENUM(GameType::Weapon)
MSGPACK_ADD_ENUM(GameType::Clothing)
MSGPACK_ADD_ENUM(GameType::Potion)
MSGPACK_ADD_ENUM(GameType::ItemType)
MSGPACK_ADD_ENUM(GameType::Class)
MSGPACK_ADD_ENUM(GameType::PlayerEvent)
MSGPACK_ADD_ENUM(GameType::Direction)


void ClientProtocol::loadBytes(std::vector<char>& loadBuffer, void* data, unsign
ed int size) {
    for (unsigned int i = 0; i < size; ++i) {
        loadBuffer[i] = *(reinterpret_cast<char *>(data) + i);
    }
}

ItemData ClientProtocol::processAddItem(std::vector<char>* _buffer, std::size_t&
 offset) {
    buffer = _buffer;
    TextureID itemTexture = Nothing;
    handler = msgpack::unpack(buffer→data(), buffer→size(), offset);
    //Tupla itemData: ItemType, Item, positionI, positionJ
    msgpack::type::tuple<GameType::ItemType, int32_t, int32_t , int32_t> itemDat
a;
    handler→convert(itemData);
    GameType::ItemType itemType = std::get<0>(itemData);//Veo que tipo de item e
s

    //Asigno la textura al item
    if (itemType ≡ GameType::ITEM_TYPE_WEAPON) {
        itemTexture = translator.getWeaponDropTexture(
                static_cast<GameType::Weapon>(std::get<1>(itemData)));

    } else if (itemType ≡ GameType::ITEM_TYPE_CLOTHING) {
        itemTexture = translator.getClothingDropTexture(
                static_cast<GameType::Clothing>(std::get<1>(itemData)));

    } else if (itemType ≡ GameType::ITEM_TYPE_POTION) {
        itemTexture = translator.getPotionTexture(
                static_cast<GameType::Potion>(std::get<1>(itemData)));
    } else if (itemType ≡ GameType::ITEM_TYPE_GOLD) {
        itemTexture = Gold;
    }
    return {{std::get<2>(itemData), std::get<3>(itemData)}, itemTexture};
}

EntityData ClientProtocol::processAddNPC(std::vector<char>* _buffer, msgpack::ty
pe::tuple<GameType::Entity,
        std::string, int32_t>& entityData, std::size_t& offset) {
    buffer = _buffer;
    EntityData npcData;
    npcData.texture = translator.getEntityTexture(std::get<0>(entityData));
    npcData.nickname = std::get<1>(entityData);
```

```cpp
    npcData.level = std::get<2>(entityData);
    //Tupla position: positionI, positionJ, direccion, distancia movida
    msgpack::type::tuple<int32_t, int32_t, GameType::Direction, int32_t> position;
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    handler->convert(position);
    npcData.pos = {std::get<0>(position), std::get<1>(position)};
    npcData.currentDir = static_cast<GameType::Direction>(std::get<2>(position));
    npcData.distanceMoved = std::get<3>(position);
    return npcData;
}

void ClientProtocol::_loadAddPlayerGeneralInfo(msgpack::type::tuple<GameType::Entity,
                            std::string, int32_t>& entityData, MapPlayerData& pData, std::size_t& offset) {
    pData.entityData.texture = Nothing;
    pData.entityData.nickname = std::get<1>(entityData);
    pData.entityData.level = std::get<2>(entityData);
    //Tupla position: positionI, positionJ, direccion, distancia movida
    msgpack::type::tuple<int32_t, int32_t, GameType::Direction, int32_t> position;
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    handler->convert(position);
    pData.entityData.pos = {std::get<0>(position), std::get<1>(position)};
    pData.entityData.currentDir = std::get<2>(position);
    pData.entityData.distanceMoved = std::get<3>(position);
    msgpack::type::tuple<GameType::Race> playerRace;
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    handler->convert(playerRace);
    pData.race = std::get<0>(playerRace);
    msgpack::type::tuple<bool> isAlive;
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    handler->convert(isAlive);
    pData.isAlive = std::get<0>(isAlive);
}

void ClientProtocol::_loadAddPlayerEquipmentInfo(MapPlayerData& pData,
                                                        std::size_t& offset) {
    msgpack::type::tuple<int32_t> item;
    PlayerEquipment equipment{};
    equipment.head = translator.getRaceTexture(
            static_cast<GameType::Race>(pData.race));
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    handler->convert(item); /*Recibo en orden el helmet, armor, shield y weapon*/
    equipment.helmet = translator.getClothingTexture(
            static_cast<GameType::Clothing>(std::get<0>(item)));
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    handler->convert(item);
    equipment.body = translator.getClothingTexture(
            static_cast<GameType::Clothing>(std::get<0>(item)));
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    handler->convert(item);
    equipment.shield = translator.getClothingTexture(
            static_cast<GameType::Clothing>(std::get<0>(item)));
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    handler->convert(item);
    equipment.weapon = translator.getWeaponTexture(
            static_cast<GameType::Weapon>(std::get<0>(item)));
    pData.equipment = equipment;
}


MapPlayerData ClientProtocol::processAddPlayer(std::vector<char>* _buffer,
```

```cpp
                                                msgpack::type::tuple<GameType::Entity,
                                                std::string, int32_t>& entityData, std::size_t& offset) {
    buffer = _buffer;
    MapPlayerData pData;
    _loadAddPlayerGeneralInfo(entityData, pData, offset);
    _loadAddPlayerEquipmentInfo(pData, offset);
    return pData;
}

/* Agrega la informacion correspondiente al inventario a PlayerData*/
void ClientProtocol::_addInventoryItems(PlayerData& data, size_t& offset) {
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    //Tupla gold: cantidad de oro, cantidad de oro seguro
    msgpack::type::tuple<int32_t, int32_t> gold;
    handler->convert(gold);
    data.generalInfo.gold = std::get<0>(gold);
    data.generalInfo.safeGold = std::get<1>(gold);
    //Aca recibe los items del inventario
    _addEquippedItems(data, offset);
    _fillInventory(data, offset);
}

/* Agrega los items equipados a PlayerData */
void ClientProtocol::_addEquippedItems(PlayerData& info, size_t& offset){
    _addClothing(info, offset, Helmet);//Esto carga el helmet
    _addClothing(info, offset, Armor);//Esto carga la armadura
    _addClothing(info, offset, Shield);//Esto carga el shield
    _addWeapon(info, offset);
}

/* Llena el inventario con los items recibidos por el server */
void ClientProtocol::_fillInventory(PlayerData& info, size_t& offset){
    for (int i = 0; i < 16; ++i) {
        handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
        msgpack::type::tuple<GameType::ItemType, int32_t> item;
        handler->convert(item);
        _addItem(info, std::get<0>(item), std::get<1>(item), i);
    }
}

/* Agrega un item al inventario */
void ClientProtocol::_addItem(PlayerData& info, GameType::ItemType type, int32_t id, int position) {
    TextureID texture;
    switch (type) {
        case GameType::ITEM_TYPE_WEAPON:
            texture = translator.getWeaponDropTexture(static_cast<GameType::Weapon>(id));
            break;
        case GameType::ITEM_TYPE_CLOTHING:
            texture = translator.getClothingDropTexture(static_cast<GameType::Clothing>(id));
            break;
        case GameType::ITEM_TYPE_POTION:
            texture = translator.getPotionTexture(static_cast<GameType::Potion>(id));
            break;
        case GameType::ITEM_TYPE_NONE:
            texture = Nothing;
            break;
        default:
            break;
    }
    info.inventoryItems.emplace_back(texture, position);
```

```cpp
}

/* Agrega las stats del jugador recibida por el server a PlayerData */
void ClientProtocol::_addPlayerStats(PlayerData& data, size_t& offset) {
    _addXPData(data, offset);
    _addManaData(data, offset);
    _addHealthData(data, offset);
    _addSkills(data, offset);
    _addPosition(data, offset);
    _addMinichatText(data, offset);
    _addNickname(data, offset);
}

/* Agrega el nickname a PlayerData */
void ClientProtocol::_addNickname(PlayerData& data, size_t& offset) {
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    msgpack::type::tuple<std::string> name;
    handler->convert(name);
    data.generalInfo.nickname = std::get<0>(name);
}

/* Agrega un item de vestimenta a PlayerData */
void ClientProtocol::_addClothing(PlayerData& info, size_t& offset, EquippedItems item) {
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    msgpack::type::tuple<int32_t> equippedClothing;
    handler->convert(equippedClothing);
    info.equippedItems.emplace_back(translator.getClothingDropTexture
            (static_cast<GameType::Clothing>(std::get<0>(equippedClothing))), item);
}

/* Agrega el arma equipada a PlayerData */
void ClientProtocol::_addWeapon(PlayerData& info, size_t& offset){
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    msgpack::type::tuple<int32_t> equippedWeapon;
    handler->convert(equippedWeapon);
    info.equippedItems.emplace_back(translator.getWeaponDropTexture(
            static_cast<GameType::Weapon>(std::get<0>(equippedWeapon))), Weapon);
}

void ClientProtocol::_addXPData(PlayerData& data, size_t& offset) {
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    //Tupla xpData: xp actual, xp para siguiente nivel, nivel actual
    msgpack::type::tuple<int32_t, int32_t, int32_t> xpData;
    handler->convert(xpData);
    data.generalInfo.xp = std::get<0>(xpData);
    data.generalInfo.nextLevelXP = std::get<1>(xpData);
    data.generalInfo.level = std::get<2>(xpData);
}

void ClientProtocol::_addHealthData(PlayerData& data, size_t& offset) {
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    //Tupla healthData: vida actual, vida total
    msgpack::type::tuple<int32_t, int32_t> healthData;
    handler->convert(healthData);
    data.generalInfo.health = std::get<0>(healthData);
    data.generalInfo.totalHealth = std::get<1>(healthData);
}

void ClientProtocol::_addManaData(PlayerData& data, size_t& offset) {
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    //Tupla manaData: mana actual, mana total
    msgpack::type::tuple<int32_t, int32_t> manaData;
    handler->convert(manaData);
```

```cpp
    data.generalInfo.mana = std::get<0>(manaData);
    data.generalInfo.totalMana = std::get<1>(manaData);
}

void ClientProtocol::_addSkills(PlayerData& data, size_t& offset){
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    //Tupla skills: strength, cosntitution, intelligence, agility
    msgpack::type::tuple<int32_t, int32_t, int32_t, int32_t> skills;
    handler->convert(skills);
    data.generalInfo.strength = std::get<0>(skills);
    data.generalInfo.constitution = std::get<1>(skills);
    data.generalInfo.intelligence = std::get<2>(skills);
    data.generalInfo.agility = std::get<3>(skills);
}

void ClientProtocol::_addPosition(PlayerData& data, size_t& offset) {
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    //Tupla pos: i, j
    msgpack::type::tuple<int32_t, int32_t> pos;
    handler->convert(pos);
    data.generalInfo.position = {std::get<0>(pos), std::get<1>(pos)};
}

void ClientProtocol::_addMinichatText(PlayerData& data, size_t& offset){
    handler = msgpack::unpack(buffer->data(), buffer->size(), offset);
    msgpack::type::tuple<std::string> text;
    handler->convert(text);
    data.minichatText = std::get<0>(text);
}


PlayerData ClientProtocol::processAddPlayerData(std::vector<char>* _buffer) {
    buffer = _buffer;
    std::size_t offset = 0;
    PlayerData data;
    _addInventoryItems(data, offset);
    _addPlayerStats(data, offset);
    return data;
}
```

```
//
// Created by ivan on 26/6/20.
//

#ifndef ARGENTUM_CLIENTEVENTHANDLER_H
#define ARGENTUM_CLIENTEVENTHANDLER_H

#include <SDL.h>
#include "../../libs/Thread.h"
#include "GameGUI.h"
#include <sstream>
#include "BlockingQueue.hpp"
#include "InputCommands/CommandVerifier.h"

class Socket;

/*Esta clase procesa los eventos del player y los envia al server acorde
 * al protocolo correspondiente*/

class ClientEventHandler : public Thread {
private:
    Socket& socket;
    bool& quit;
    GameGUI& game;
    CommandVerifier cmdVerifier;
    BlockingQueue<std::unique_ptr<SDL_Event>>& events;
    std::stringstream msgBuffer;

public:
    ClientEventHandler(Socket& _socket, bool& quit, GameGUI& game,
                            BlockingQueue<std::unique_ptr<SDL_Event>>& _events)
                        : socket(_socket), quit(quit), game(game), events(_event
s) {};

    void run() override;

private:
    void _handleKeyDown(SDL_Event& e);
    void _sendMessage();
    void _handleMouseButtonDown(SDL_Event &e);
    void _processAttack(Coordinate coordinate);
    void _processUseItem(int _inventorySlot);
    void _processUnequipItem(GameType::EquipmentPlace _equipment);
    void _processCommandInput();
    void _handleKeyUp(SDL_Event& e);
};


#endif //ARGENTUM_CLIENTEVENTHANDLER_H
```

```
//
// Created by ivan on 26/6/20.
//

#include "ClientEventHandler.h"
#include "BlockingQueue.hpp"
#include <msgpack.hpp>
#include "../../libs/Socket.h"
#include "ClientProtocol.h"
#include "InputCommands/InputCommand.h"

MSGPACK_ADD_ENUM(GameType::Direction)
MSGPACK_ADD_ENUM(GameType::PlayerEvent)

/* Procesa los eventos del usuario y manda dicho evento con su informacion corre
spondiente al server */
void ClientEventHandler::run() {
    Minichat& minichat = game.getMinichat();

    try {
        while (¬quit) {
            std::unique_ptr<SDL_Event> e = events.pop();
            if (e) {
                switch (e→type) {
                    case SDL_QUIT:
                        quit = true;
                        break;
                    case SDL_KEYDOWN:
                        _handleKeyDown(*e);
                        break;
                    case SDL_KEYUP:
                        _handleKeyUp(*e);
                        break;
                    case SDL_MOUSEBUTTONDOWN:
                        _handleMouseButtonDown(*e);
                        break;
                    case SDL_TEXTINPUT:
                        minichat.handleTextInput(*e);
                        break;
                    case SDL_MOUSEWHEEL:
                        minichat.handleMouseWheel(*e);
                        break;
                }
            }
            if (msgBuffer.rdbuf()→in_avail() ≠ 0) { /*Nos cargaron un mensaje*/
                _sendMessage();
            }
        }
    } catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

/* Porcesa el evento de cuando se deja de apretar una tecla */
void ClientEventHandler::_handleKeyUp(SDL_Event& e) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_STOP_MOVI
NG);
    switch (e.key.keysym.sym) {
        case SDLK_UP:
            msgpack::pack(msgBuffer, event);
            break;
        case SDLK_DOWN:
            msgpack::pack(msgBuffer, event);
            break;
        case SDLK_LEFT:
            msgpack::pack(msgBuffer, event);
```

```cpp
            break;
        case SDLK_RIGHT:
            msgpack::pack(msgBuffer, event);
            break;
    }
}

/* Procesa el evento de cuando se hace click */
void ClientEventHandler::_handleMouseButtonDown(SDL_Event& e){
    int clickX, clickY;
    SDL_GetMouseState(&clickX, &clickY);
    //Escalo la posicion de click
    clickX = (float)clickX * ((float)DEFAULT_SCREEN_WIDTH/(float)game.getWindow().getWidth());
    clickY = (float)clickY * ((float)DEFAULT_SCREEN_HEIGHT/(float)game.getWindow().getHeight());

    game.getMinichat().handleMouseButtonDown({clickY, clickX}, game.getWindow());

    game.getSelector().handleEvent({clickY, clickX},{game.getPlayerInfo().getYPos(),
                         game.getPlayerInfo().getXPos()},game.getWindow());

    if (e.button.button ≡ SDL_BUTTON_RIGHT) {
        if (Selector::hasSelectedTile({clickY, clickX})) {
            _processAttack(game.getSelector().getSelectedTile());
        }
        if (Selector::hasSelectedSlot({clickY, clickX})) {
            _processUseItem(game.getSelector().getInventorySlot());
        }
        if (Selector::hasSelectedEquipment({clickY, clickX})) {
            _processUnequipItem(game.getSelector().getSelectedEquipment());
        }
    }
}

/* Procesa el evento de cuando se apreta una tecla */
void ClientEventHandler::_handleKeyDown(SDL_Event& e) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_START_MOVING);
    msgpack::type::tuple<GameType::Direction> direction;
    if (e.key.repeat ≡ 0) {
        switch (e.key.keysym.sym) {
            case SDLK_UP:
                game.getSelector().resetTileSelection();
                direction = {GameType::DIRECTION_UP};
                msgpack::pack(msgBuffer, event);
                msgpack::pack(msgBuffer, direction);
                break;
            case SDLK_DOWN:
                game.getSelector().resetTileSelection();
                direction = {GameType::DIRECTION_DOWN};
                msgpack::pack(msgBuffer, event);
                msgpack::pack(msgBuffer, direction);
                break;
            case SDLK_LEFT:
                game.getSelector().resetTileSelection();
                direction = {GameType::DIRECTION_LEFT};
                msgpack::pack(msgBuffer, event);
                msgpack::pack(msgBuffer, direction);
                break;
            case SDLK_RIGHT:
                game.getSelector().resetTileSelection();
                direction = {GameType::DIRECTION_RIGHT};
                msgpack::pack(msgBuffer, event);
```

```cpp
                msgpack::pack(msgBuffer, direction);
                break;
            case SDLK_BACKSPACE:
                game.getMinichat().handleBackspace();
                break;
            case SDLK_RETURN:
                _processCommandInput();
                break;
            case SDLK_TAB:
                if (SoundPlayer::isMusicPlaying()) {
                    game.getSoundPlayer().pauseMusic();
                } else {
                    game.getSoundPlayer().playMusic();
                }
                break;
        }
    }
}

/* Procesa el evento cuando apreto enter para ejecutar un comando del minichat */
void ClientEventHandler::_processCommandInput() {
    std::string cmd = game.getMinichat().handleReturnKey();
    if (cmd ≠ ""){ //Si apreto enter y no hay texto handleReturnKey me devuelve esto
        if (cmd ≡ "/clear") {
            game.getMinichat().clearMinichat();
        } else {
            std::unique_ptr<InputCommand> inputCmd;
            inputCmd = cmdVerifier.verifyCommand(game, std::move(cmd));
            if (inputCmd) {
                (*inputCmd)(msgBuffer);//Arma el mensaje y lo packea en msgBuffer
            }
        }
    }
}

/* Arma el buffer y lo envia para el evento de desequipar un item */
void ClientEventHandler::_processUnequipItem(GameType::EquipmentPlace _equipment){
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_UNEQUIP);
    msgpack::type::tuple<int32_t> equipment;
    equipment = _equipment;
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, equipment);
}

/* Arma el buffer y lo envia para el evento de equiparse un item */
void ClientEventHandler::_processUseItem(int _inventorySlot) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_USE_ITEM);
    msgpack::type::tuple<int32_t> inventorySlot;
    inventorySlot = _inventorySlot;
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, inventorySlot);
}

/* Arma el buffer y lo envia para el evento de atacar a una posicion */
void ClientEventHandler::_processAttack(Coordinate selectedTile) {
    msgpack::type::tuple<GameType::PlayerEvent> event(GameType::PLAYER_ATTACK);
    msgpack::type::tuple<int32_t, int32_t> targetPosition;
    targetPosition = {selectedTile.i, selectedTile.j};
    msgpack::pack(msgBuffer, event);
    msgpack::pack(msgBuffer, targetPosition);
}
```

```cpp

/* Envia el mensaje con el evento y su informacion al servidor */
void ClientEventHandler::_sendMessage() {
    std::string aux = msgBuffer.str();
    uint32_t length = htonl(aux.size());
    std::vector<char> buffer(sizeof(uint32_t));
    ClientProtocol::loadBytes(buffer, &length, sizeof(uint32_t));
    std::copy(aux.begin(), aux.end(), std::back_inserter(buffer));
    socket.send(buffer.data(), buffer.size());
    msgBuffer.str(""); /*Reseteo el stringstream*/
    msgBuffer.clear();
}

```

```cpp
//
// Created by marcos on 9/7/20.
//

#ifndef ARGENTUM_CITIZENDATA_H
#define ARGENTUM_CITIZENDATA_H

#include "../Texture/TextureID.h"
#include <string>


 /*Se usa cuando cargo la data inicial de los tiles, por si tienen un citizen
  * (trader, banker, etc)*/
struct CitizenData {
    TextureID texture;
    std::string nickname;
};

#endif //ARGENTUM_CITIZENDATA_H
```

```
#ifndef TP3TALLER_ARGENTUMCLIENTSIDE_H
#define TP3TALLER_ARGENTUMCLIENTSIDE_H

class ArgentumClientSide {
public:
    /* Instancia al cliente y comienza su ejecucion */
    static int run(int argc);
};


#endif //TP3TALLER_ARGENTUMCLIENTSIDE_H
```

```
#include "ArgentumClientSide.h"
#include "ArgentumClient.h"
#include <iostream>

#define INVALID_ARGUMENTS_MESSAGE "Error: Invalid Arguments."
#define ARGUMENT_AMOUNT 1

int ArgentumClientSide::run(int argc) {
    if (argc ≠ ARGUMENT_AMOUNT) {
        std::cerr << INVALID_ARGUMENTS_MESSAGE << std::endl;
        return EXIT_FAILURE;
    }
    try {
        Client client;
        client.run();
    } catch(std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
    return EXIT_SUCCESS;
}
```

```cpp
#ifndef TP3TALLER_CLIENT_H
#define TP3TALLER_CLIENT_H

/*Esta clase se encarga de manejar la logica de la conexion y comuniacion
 * con el Server*/

#include "../../libs/Socket.h"
#include "GameGUI.h"
#include <string>
#include "ClientEventHandler.h"

struct GameStartInfo;

class Client {
private:
    Socket socket;

public:
    Client();
    Client(const Client&) = delete; /*Borro los constructores por copia*/
    Client operator=(const Client&) = delete;

    /* Comienza la ejecucion del cliente */
    void run();
    ~Client();

private:
    void _gameLoop();
    static void _initializeSDL();
    static void _closeSDL();
    static void _setCursor();
};


#endif //TP3TALLER_CLIENT_H
```

```cpp
#include <netdb.h>
#include "ArgentumClient.h"
#include "ClientProtocol.h"
#include <vector>
#include <utility>
#include "BlockingQueue.hpp"
#include <SDL_mixer.h>
#include "../UpdateEvents/UpdateEvent.h"
#include "UpdateReceiver.h"
#include "GameInitializer.h"
#include "../Screen/MainMenu.h"
#include "UpdateManager.h"

#define FREQUENCY 44100
#define CHUNKSIZE 2048
#define CURSOR_PATH "/var/Argentum/Assets/Images/UI/Cursor.bmp"


void Client::_gameLoop() {
    bool quit = false;
    GameGUI game;
    Timer timer;
    class MainMenu mainMenu(game.getTextureRepo().getTexture(MainMenu),
                            game.getWindow());
    Window& window = game.getWindow();
    ClientProtocol protocol(socket);
    GameInitializer initializer(game, socket, protocol);

    /* Loop del menu principal */
    mainMenu.menuScreen(quit, initializer, socket);

    if (quit) return;//Si elegi salir del juego en el menu no tengo que hacer na
da mas

    initializer.initializeGame();

    BlockingQueue<std::unique_ptr<SDL_Event>> sdlEvents;
    UpdateManager updateManager;
    ClientEventHandler eventHandler(socket, quit, game, sdlEvents);
    UpdateReceiver updater(protocol, updateManager, socket, quit);
    std::unique_ptr<SDL_Event> event(new SDL_Event());

    /* Se lanzan los dos threads que van a manejar los eventos de input de usuar
io
     * y los recibidos por el server respectivamente */
    eventHandler();
    updater();

    timer.start();
    game.getSoundPlayer().playMusic();
    double timeStep;

    try {
        while (¬quit) {
            //Eventos de input del usuario
            while(SDL_PollEvent(event.get()) ≠ 0) {
                if (¬window.handleEvent(*event)) {
                    sdlEvents.push(std::move(event));
                    event.reset(new SDL_Event());
                }
            }

            //Eventos recibidos por el servidor
            int updatesAvailable = updateManager.updatesAvailable();
            if (updatesAvailable > 0 ∧ updatesAvailable < 5) {
                updatesAvailable = 1;
            } /*No updateo todas si hay menos de 5 ya que pierde fluidez la camara (poca, pero notable)
              * y considero que dado que nuestros updates son cada 16 ms unos 80 ms de atraso para este tipo de juego es imperceptible.
              * Sin embargo, si el cliente se atrasa 5 o mas updates se las aplico todas para que esto no sea un problema*/
            for (int i = 0; i < updatesAvailable; ++i) {
                auto update = updateManager.pop();
                while (¬update.empty()) {
                    auto updateEvent = update.pop();
                    (*updateEvent)(game);
                }
                timeStep = timer.getTime();
                timer.start();
                game.update(timeStep);
            }

            game.getSoundPlayer().playSounds();
            game.render(); /*No hace falta dormir al cpu ya que el juego utiliza VSYNC, por lo que el frame rate ya se cappea*/
        }
    } catch (std::exception& e) {
        std::cerr << e.what() << " in Main Game Loop" << std::endl;
    } catch (...) {
        std::cerr << "Unknown Error in Main Game Loop" << std::endl;
    }

    quit = true;
    socket.close();
    sdlEvents.doneAdding();
    eventHandler.join();
    updater.join();
}

void Client::run() {
    _gameLoop();
}

Client::Client(){
    _initializeSDL();
    _setCursor();
}

//Setea un cursor custom
void Client::_setCursor() {
    SDL_Surface *surface;
    SDL_Cursor *cursor;
    surface = SDL_LoadBMP(CURSOR_PATH);
    if (¬surface) {
        throw TPException("Could not create cursor");
    }
    cursor = SDL_CreateColorCursor(surface, 0, 0);
    SDL_FreeSurface(surface);
    if (¬cursor) {
        throw TPException("Could not create cursor");
    }
    SDL_SetCursor(cursor);
}

void Client::_initializeSDL() {
    //Inicializa audio y video
    if(SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO) < 0) {
        throw TPException("Graphics could not initialize! Graphics Error: %s\n", SDL_GetError());
    } else {
        //Setea filtrado de texturas lineal
```

```cpp
        if(¬SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "MipmapLinearNearest")) {
            std::cerr << "Warning: Linear texture filtering not enabled!" << std::endl;
        }
        //Inicializa la carga de png
        int imgFlags = IMG_INIT_PNG;
        if(¬(IMG_Init(imgFlags) & imgFlags)) {
            SDL_Quit();
            throw TPException("SDL_image could not initialize! SDL_mage Error: %s\n", IMG_GetError() );
        }
    }
    //Inicializa el audio, permite cargar la musica MP3
    if(Mix_Init(MIX_INIT_MP3) ≡ 0) {
        IMG_Quit();
        SDL_Quit();
        throw TPException("SDL_mixer could not initialize!"
                          " SDL_mixer Error: %s\n", Mix_GetError());
    }
    //Inicializa el reproductor de audio
    if(Mix_OpenAudio(FREQUENCY, MIX_DEFAULT_FORMAT, 2, CHUNKSIZE) < 0) {
        Mix_Quit();
        IMG_Quit();
        SDL_Quit();
        throw TPException("SDL_mixer could not initialize!"
                          " SDL_mixer Error: %s\n", Mix_GetError());
    }
    //Inicializa el cargado de fonts para texto
    if(TTF_Init() ≡ -1) {
        Mix_Quit();
        IMG_Quit();
        SDL_Quit();
        throw TPException("SDL_ttf could not initialize! SDL_ttf Error:"
                          " %s\n", TTF_GetError());
    }
}

void Client::_closeSDL() {
    TTF_Quit();
    Mix_CloseAudio();
    Mix_Quit();
    IMG_Quit();
    SDL_QuitSubSystem(SDL_INIT_VIDEO | SDL_INIT_AUDIO);
    SDL_Quit();
}

Client::~Client() {
    _closeSDL();
}
```

```cpp
//
// Created by marcos on 6/7/20.
//

#ifndef ARGENTUM_PLAYER_H
#define ARGENTUM_PLAYER_H

#include "Entity.h"
#include "../Texture/PlayerTexture.h"
#include "../Texture/NPCTexture.h"

/*Esta clase representa a los players en el mapa, incluido el propio*/

class Player : public Entity {
private:
    PlayerTexture pTexture;
    NPCTexture ghostTexture;
    bool isAlive{true};

public:
    Player(TextureRepository& repo, SDL_Rect& camera, float x, float y,
                                                    PlayerEquipment& images,
                                                    bool _isAlive = true,
                                                    std::string∧ level = "",
                                                    const std::string& nickname
= "");

    /* Renderiza al jugador con su equipamiento (o como fantasma si esta muerto)
 */
    void render() override;

    /* AÃ±ade la textura del item que se quiere equipar al jugador */
    void equip(GameType::EquipmentPlace place, TextureID equipment);

    /* Setea isAlive en false asi luego lo renderizo como fantasma */
    void kill();

    /* Setea isAlive en true asi luego lo renderizo normalmente */
    void revive();

    /*Aumenta el nivel que se muestra del player*/
    void updateLevel(int level) override;
};


#endif //ARGENTUM_PLAYER_H
```

```cpp
//
// Created by marcos on 6/7/20.
//

#include "Player.h"

Player::Player(TextureRepository& repo, SDL_Rect& camera, float x, float y,
        PlayerEquipment& images, bool _isAlive, std::string∧ level, const std::
string& nickname) :
        Entity(camera, x, y), pTexture(repo, images, level, nickname),
        ghostTexture(repo, PlayerGhost, "(" + level + ")", nickname) {
    isAlive = _isAlive;
}

void Player::render() {
    if (cameraFollows) {
        Entity::updateCamera();
    }
    if (isAlive) {
        Entity::render(pTexture);
    } else {
        Entity::render(ghostTexture);
    }
}

void Player::equip(GameType::EquipmentPlace place, TextureID equipment) {
    pTexture.equip(place, equipment);
}

void Player::kill() {
    isAlive = false;
}

void Player::revive() {
    isAlive = true;
}

void Player::updateLevel(int level) {
    std::string strLevel = std::to_string(level);
    pTexture.setLevel(strLevel);
    ghostTexture.setLevel("(" + strLevel + ")");
}
```

```cpp
//
// Created by marcos on 6/8/20.
//

#ifndef ARGENTUM_NPC_H
#define ARGENTUM_NPC_H

#include "../Texture/NPCTexture.h"
#include "Entity.h"

/*Esta clase representa tanto a los monstruos como a los citizen (priest, banker
, etc)*/

class NPC : public Entity {
private:
    NPCTexture npcTexture;

public:
    NPC(TextureRepository& repo, SDL_Rect& camera, float x, float y,
            TextureID texture, std::string^ level = "");

    /*Aumenta el nivel que se muestra del player*/
    void updateLevel(int level) override;

    /*Renderiza en el mapa al NPC*/
    void render() override;
};


#endif //ARGENTUM_NPC_H
```

```cpp
//
// Created by marcos on 6/8/20.
//

#include "NPC.h"

NPC::NPC(TextureRepository& repo, SDL_Rect &camera, float x, float y,
         TextureID texture, std::string^ level) : Entity(camera, x, y),
         npcTexture(repo, texture, std::move(level)) {}

void NPC::render() {
    Entity::render(npcTexture);
}

void NPC::updateLevel(int level) {
    npcTexture.setLevel(std::to_string(level));
}
```

```cpp
//
// Created by marcos on 6/9/20.
//

#ifndef ARGENTUM_ENTITY_H
#define ARGENTUM_ENTITY_H

#include "../Texture/EntityTexture.h"
#include "../../libs/GameEnums.h"
#include <list>
#include <memory>
#include "../Miscellaneous/Spell.h"

/*Esta clase encapsula el comportamiento general de las entites (personajes
 * del juego, sean npcs o players)*/

class Entity {
private:
    SDL_Rect& camera;
    int currentFrame;
    GameType::Direction moveDirection, lastDirection;
    float xPosition, width;
    float yPosition, height;
    float totalDistanceMoved{0};

protected:
    bool cameraFollows{false};
    std::weak_ptr<Spell> spell;

public:
    Entity(SDL_Rect& camera, float x, float y);

    /*Desplaza al entity en la direccion indicada la distanceTravelled indicada.
      * Si reachedDestination es true, resetea el contador y el frame de la anima
cion.
      * Este sera true cuando el entity haya terminado de desplazarse la distanci
a
      * entre un tile y otro (el server me avisa)*/
    GameType::Direction move(GameType::Direction direction, unsigned int distanc
eTravelled, bool reachedDestination);

    /*Renderiza al la textura de entity recibida*/
    void render(EntityTexture& eTexture);

    /*Metodo abstracto, deben implementarlo los hijos*/
    virtual void render() = 0;

    /*Centra la camara en el player*/
    void updateCamera();

    /*Setea que la camara siga al player, este metodo solo se debera ejecutar
      * en el player propio*/
    void activateCamera();

    /*Agrega un hechizo al entity para que lo siga*/
    void addSpell(std::shared_ptr<Spell>& _spell);

    /*Retorna el spell que guarda el entity, este metodo existe en el caso
      * donde el entity sea matado por un hechizo y el hechizo deba migrar a un t
ile
      * para no perder la animacion del hechizo*/
    std::weak_ptr<Spell>& getSpell();

    /*Setea la posicion interna del player (en pixeles)*/
    void setPosition(float _xPosition, float _yPosition);

```

```cpp
    /*Setea la direccion de renderizado del entity, se utiliza para cuando
      * un entity ataca, para que mire a la direccion en la que ataco*/
    void setLookDirection(GameType::Direction direction);

    virtual void updateLevel(int level) = 0;

    virtual ~Entity() = default;

private:
    void _renderLastDirection(EntityTexture& eTexture);
    void _modifyPosition(GameType::Direction direction, float distance);
    void _updateFrame(bool reachedDestination);
};


#endif //ARGENTUM_ENTITY_H
```

```cpp
//
// Created by marcos on 6/9/20.
//

#include "Entity.h"
#include "../Client/GameConstants.h"
#include "../../libs/SharedConstants.h"
#include "../Miscellaneous/CameraCollisionVerifier.h"

Entity::Entity(SDL_Rect &camera, float x, float y) : camera(camera) {
    currentFrame = 0;
    moveDirection = GameType::DIRECTION_STILL;
    lastDirection = GameType::DIRECTION_STILL;
    xPosition = x;
    yPosition = y;
    width = (float)TILE_WIDTH/2;
    height = (float)TILE_HEIGHT;
}

void Entity::_updateFrame(bool reachedDestination) {
    if (reachedDestination) {
        if (totalDistanceMoved < static_cast<float>(TILE_WIDTH)) {
            _modifyPosition(moveDirection, static_cast<float>(TILE_WIDTH) - totalDistanceMoved);
        }
        currentFrame = 0;
        lastDirection = moveDirection;
        moveDirection = GameType::DIRECTION_STILL;
        totalDistanceMoved = 0;
    } else {
        for (int i = 0; i < 6; ++i) { /*6 es la cantidad de frames distintos del body*/
            if (totalDistanceMoved < ((float)TILE_WIDTH/6 * (float)(i+1))) {
                currentFrame = i;
                break;
            }
        }
    }
}

void Entity::render(EntityTexture& eTexture) {
    if (CameraCollisionVerifier::isInsideCamera(camera, {(int)xPosition,
                                                         (int)yPosition, (int)width, (int)height})) {
        switch (moveDirection) {
            case GameType::DIRECTION_UP:
                eTexture.renderBack((int)(xPosition) - camera.x,
                                    (int)(yPosition) - camera.y, currentFrame);
                break;
            case GameType::DIRECTION_DOWN:
                eTexture.renderFront((int)(xPosition) - camera.x,
                                     (int)(yPosition) - camera.y, currentFrame);
                break;
            case GameType::DIRECTION_RIGHT:
                eTexture.renderRight((int)(xPosition) - camera.x,
                                     (int)(yPosition) - camera.y, currentFrame);
                break;
            case GameType::DIRECTION_LEFT:
                eTexture.renderLeft((int)(xPosition) - camera.x,
                                    (int)(yPosition) - camera.y, currentFrame);
                break;
            case GameType::DIRECTION_STILL:
                _renderLastDirection(eTexture);
        }
    }
    auto _spell = spell.lock();
```

```cpp
    if (_spell) {
        _spell->setPosition(xPosition, yPosition);
        _spell->render();
    }
}

void Entity::_renderLastDirection(EntityTexture& eTexture) {
    switch (lastDirection) {
        case GameType::DIRECTION_UP:
            eTexture.renderBack((int)(xPosition) - camera.x,
                                (int)(yPosition) - camera.y, 0);
            break;
        case GameType::DIRECTION_DOWN:
            eTexture.renderFront((int)(xPosition) - camera.x,
                                 (int)(yPosition) - camera.y, 0);
            break;
        case GameType::DIRECTION_RIGHT:
            eTexture.renderRight((int)(xPosition) - camera.x,
                                 (int)(yPosition) - camera.y, 0);
            break;
        case GameType::DIRECTION_LEFT:
            eTexture.renderLeft((int)(xPosition) - camera.x,
                                (int)(yPosition) - camera.y, 0);
            break;
        case GameType::DIRECTION_STILL:
            eTexture.renderFront((int)(xPosition) - camera.x,
                                 (int)(yPosition) - camera.y, 0);
    }
}

void Entity::updateCamera() {
    //Centro la camara sobre el jugador
    camera.x = ((int)xPosition + 55 / 2 ) - DEFAULT_MAP_WIDTH / 2;
    camera.y = ((int)yPosition + 100 / 2 ) - DEFAULT_MAP_HEIGHT / 2;

    //Mantengo la camara en los bordes
    if (camera.x < 0) {
        camera.x = 0;
    }
    if (camera.y < 0) {
        camera.y = 0;
    }
    if (camera.x > LEVEL_WIDTH - camera.w) {
        camera.x = LEVEL_WIDTH - camera.w;
    }
    if (camera.y > LEVEL_HEIGHT - camera.h) {
        camera.y = LEVEL_HEIGHT - camera.h;
    }
}

GameType::Direction Entity::move(GameType::Direction direction, unsigned int distanceTravelled,
                                    bool reachedDestination) {
    float distanceInPixels = static_cast<float>(TILE_WIDTH) *
                             static_cast<float>(distanceTravelled) / static_cast<float>(TILE_DISTANCE_IN_METERS);
    GameType::Direction previousDirection = moveDirection;
    moveDirection = direction;
    _modifyPosition(direction, distanceInPixels);
    totalDistanceMoved += distanceInPixels;
    _updateFrame(reachedDestination);
    return previousDirection;
}

void Entity::_modifyPosition(GameType::Direction direction, float distance) {
    switch (direction) {
        case GameType::DIRECTION_UP:
            yPosition -= distance;
            break;
        case GameType::DIRECTION_DOWN:
            yPosition += distance;
            break;
        case GameType::DIRECTION_LEFT:
            xPosition -= distance;
            break;
        case GameType::DIRECTION_RIGHT:
            xPosition += distance;
            break;
        case GameType::DIRECTION_STILL:
            //do nothing
            break;
    }
}

void Entity::activateCamera() {
    cameraFollows = true;
    camera.x = static_cast<int>(xPosition);
    camera.y = static_cast<int>(yPosition);
}

void Entity::addSpell(std::shared_ptr<Spell>& _spell) {
    spell = _spell;
    _spell→setPosition(xPosition, yPosition);
}

std::weak_ptr<Spell> &Entity::getSpell() {
    return spell;
}

void Entity::setPosition(float _xPosition, float _yPosition) {
    xPosition = _xPosition;
    yPosition = _yPosition;
    moveDirection = GameType::DIRECTION_STILL;
    lastDirection = moveDirection;
    currentFrame = 0;
    totalDistanceMoved = 0;
}

void Entity::setLookDirection(GameType::Direction direction) {
    lastDirection = direction;
}
```

Table of Contents