give me a basic skeleton/boiler plate for this project ,

Now I'll create a comprehensive boilerplate/skeleton for this Fantasy Game project based on the requirements.

Fantasy Game Project - Basic Skeleton/Boilerplate

Here's a complete boilerplate structure for your Fantasy Game project based on the assignment requirements. This skeleton uses object-oriented design with inheritance, polymorphism, smart pointers, and follows proper C++ practices.[1]

Project Structure

```
FantasyGame/
├── main.cpp
├── Game.h
├── Game.cpp
├── Character.h
├── Character.cpp
├── Race.h
├── Race.cpp
├── Item.h
├── Item.cpp
├── Location.h
├── Location.cpp
├── Board.h
└── Board.cpp
```

1. main.cpp

```
/**
 * @file main.cpp
 * @brief Entry point for the Fantasy Game
 * @author [Your Name]
 * @date 2025-11-10
 */

#include "Game.h"
#include <iostream>

/**
 * @brief Main function to start the game
 * @return int Exit status
```

```
 */
int main()
{
    try {
        Game game;
        game.run();
    }
    catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

2. Character.h

```
/**
 * @file Character.h
 * @brief Abstract base class for all characters in the game
 * @author [Your Name]
 * @date 2025-11-10
 */

#ifndef CHARACTER_H
#define CHARACTER_H

#include "Item.h"
#include <string>
#include <vector>
#include <memory>

/**
 * @class Character
 * @brief Abstract base class representing a game character
 */
class Character
{
protected:
    std::string name;
    int attack;
    double attackChance;
    int defence;
    double defenceChance;
    int health;
    int maxHealth;
    int strength;
    int gold;
    std::vector<std::shared_ptr<Item>> inventory;

public:
    /**
     * @brief Constructor for Character
     * @param n Character name
     * @param a Attack value
```

```
     * @param ac Attack chance
     * @param d Defence value
     * @param dc Defence chance
     * @param h Health value
     * @param s Strength value
     */
    Character(const std::string& n, int a, double ac, int d, double dc, int h, int s);

    /**
     * @brief Virtual destructor
     */
    virtual ~Character() = default;

    /**
     * @brief Pure virtual function for special defence ability
     * @param damage Incoming damage
     * @return int Actual damage taken
     */
    virtual int specialDefence(int damage) = 0;

    /**
     * @brief Get total attack including items
     * @return int Total attack value
     */
    int getTotalAttack() const;

    /**
     * @brief Get total defence including items
     * @return int Total defence value
     */
    int getTotalDefence() const;

    /**
     * @brief Add item to inventory
     * @param item Shared pointer to item
     * @return bool True if successful
     */
    bool addItem(std::shared_ptr<Item> item);

    /**
     * @brief Remove item from inventory
     * @param itemName Name of item to remove
     * @return std::shared_ptr<Item> Removed item or nullptr
     */
    std::shared_ptr<Item> removeItem(const std::string& itemName);

    /**
     * @brief Check if attack succeeds based on attack chance
     * @return bool True if attack succeeds
     */
    bool attemptAttack() const;

    /**
     * @brief Check if defence succeeds based on defence chance
     * @return bool True if defence succeeds
     */
```

```
    bool attemptDefence() const;

    /**
     * @brief Take damage and update health
     * @param damage Amount of damage
     */
    void takeDamage(int damage);

    /**
     * @brief Check if character is alive
     * @return bool True if health > 0
     */
    bool isAlive() const;

    /**
     * @brief Get current inventory weight
     * @return int Total weight
     */
    int getCurrentWeight() const;

    /**
     * @brief Display inventory contents
     */
    void displayInventory() const;

    // Getters
    std::string getName() const { return name; }
    int getHealth() const { return health; }
    int getStrength() const { return strength; }
    int getGold() const { return gold; }

    // Setters
    void addGold(int amount) { gold += amount; }
    void setHealth(int h) { health = h; }
};

#endif // CHARACTER_H
```

3. Character.cpp

```
/**
 * @file Character.cpp
 * @brief Implementation of Character class
 */

#include "Character.h"
#include <iostream>
#include <algorithm>
#include <cstdlib>

Character::Character(const std::string& n, int a, double ac, int d, double dc, int h, int
    : name(n), attack(a), attackChance(ac), defence(d), defenceChance(dc),
      health(h), maxHealth(h), strength(s), gold(0)
{
}
```

```
int Character::getTotalAttack() const
{
    /* Pseudo-code:
     * 1. Initialize total with base attack
     * 2. For each item in inventory
     *    a. Add item's attack modifier
     * 3. Return total
     */
    int total = attack;
    for (const auto& item : inventory) {
        total += item->getAttackModifier();
    }
    return total;
}

int Character::getTotalDefence() const
{
    /* Pseudo-code:
     * 1. Initialize total with base defence
     * 2. For each item in inventory
     *    a. Add item's defence modifier
     * 3. Return total
     */
    int total = defence;
    for (const auto& item : inventory) {
        total += item->getDefenceModifier();
    }
    return total;
}

bool Character::addItem(std::shared_ptr<Item> item)
{
    /* Pseudo-code:
     * 1. Calculate current weight + new item weight
     * 2. If exceeds strength, return false
     * 3. Check if item category already exists (except rings)
     * 4. If exists and not ring, return false
     * 5. Add item to inventory
     * 6. Apply item modifiers to character stats
     * 7. Return true
     */
    if (getCurrentWeight() + item->getWeight() > strength) {
        return false;
    }

    // Check for category restriction (one per category except rings)
    if (item->getCategory() != "Ring") {
        for (const auto& existingItem : inventory) {
            if (existingItem->getCategory() == item->getCategory()) {
                return false;
            }
        }
    }

    inventory.push_back(item);
```

```
    // Apply modifiers
    attack += item->getAttackModifier();
    defence += item->getDefenceModifier();
    health += item->getHealthModifier();
    strength += item->getStrengthModifier();

    return true;
}

std::shared_ptr<Item> Character::removeItem(const std::string& itemName)
{
    /* Pseudo-code:
     * 1. Search inventory for item by name
     * 2. If found:
     *    a. Remove item modifiers from character stats
     *    b. Remove item from inventory
     *    c. Return shared pointer to item
     * 3. Return nullptr if not found
     */
    auto it = std::find_if(inventory.begin(), inventory.end(),
        [&itemName](const std::shared_ptr<Item>& item) {
            return item->getName() == itemName;
        });

    if (it != inventory.end()) {
        std::shared_ptr<Item> item = *it;

        // Remove modifiers
        attack -= item->getAttackModifier();
        defence -= item->getDefenceModifier();
        health -= item->getHealthModifier();
        strength -= item->getStrengthModifier();

        inventory.erase(it);
        return item;
    }

    return nullptr;
}

bool Character::attemptAttack() const
{
    double random = static_cast<double>(rand()) / RAND_MAX;
    return random < attackChance;
}

bool Character::attemptDefence() const
{
    double random = static_cast<double>(rand()) / RAND_MAX;
    return random < defenceChance;
}

void Character::takeDamage(int damage)
{
    health -= damage;
```

```
    if (health < 0) {
        health = 0;
    }
}

bool Character::isAlive() const
{
    return health > 0;
}

int Character::getCurrentWeight() const
{
    int total = 0;
    for (const auto& item : inventory) {
        total += item->getWeight();
    }
    return total;
}

void Character::displayInventory() const
{
    std::cout << "\n=== Inventory ===" << std::endl;
    std::cout << "Gold: " << gold << std::endl;
    std::cout << "Current Weight: " << getCurrentWeight() << "/" << strength << std::endl
    std::cout << "\nItems:" << std::endl;

    if (inventory.empty()) {
        std::cout << "  (empty)" << std::endl;
    } else {
        for (const auto& item : inventory) {
            std::cout << "  - " << item->getName() << " (Weight: " << item->getWeight() <
        }
    }
}
```

4. Race.h

```
/**
 * @file Race.h
 * @brief Concrete character race classes
 * @author [Your Name]
 * @date 2025-11-10
 */

#ifndef RACE_H
#define RACE_H

#include "Character.h"

/**
 * @class Human
 * @brief Human race character
 */
class Human : public Character
{
```

```
public:
    Human(const std::string& name);
    int specialDefence(int damage) override;
};

/**
 * @class Elf
 * @brief Elf race character
 */
class Elf : public Character
{
public:
    Elf(const std::string& name);
    int specialDefence(int damage) override;
};

/**
 * @class Dwarf
 * @brief Dwarf race character
 */
class Dwarf : public Character
{
public:
    Dwarf(const std::string& name);
    int specialDefence(int damage) override;
};

/**
 * @class Hobbit
 * @brief Hobbit race character
 */
class Hobbit : public Character
{
public:
    Hobbit(const std::string& name);
    int specialDefence(int damage) override;
};

/**
 * @class Orc
 * @brief Orc race character with day/night abilities
 */
class Orc : public Character
{
private:
    bool isNightTime;

public:
    Orc(const std::string& name);
    int specialDefence(int damage) override;
    void setTimeOfDay(bool night);
    bool getIsNightTime() const { return isNightTime; }

    // Override attack/defence getters for day/night variation
    int getTotalAttack() const;
    int getTotalDefence() const;
```

```
    double getAttackChance() const;
    double getDefenceChance() const;
};

#endif // RACE_H
```

5. Race.cpp

```
/**
 * @file Race.cpp
 * @brief Implementation of race classes
 */

#include "Race.h"
#include <cstdlib>

// Human: Attack 30, AttackChance 2/3, Defence 20, DefenceChance 1/2, Health 60, Strengt
Human::Human(const std::string& name)
    : Character(name, 30, 2.0/3.0, 20, 1.0/2.0, 60, 100)
{
}

int Human::specialDefence(int damage)
{
    // Successful defences never cause damage
    return 0;
}

// Elf: Attack 40, AttackChance 1/1, Defence 10, DefenceChance 1/4, Health 40, Strength 7
Elf::Elf(const std::string& name)
    : Character(name, 40, 1.0, 10, 1.0/4.0, 40, 70)
{
}

int Elf::specialDefence(int damage)
{
    // Successful defences increase health by 1
    health += 1;
    return 0;
}

// Dwarf: Attack 30, AttackChance 2/3, Defence 20, DefenceChance 2/3, Health 50, Strengt
Dwarf::Dwarf(const std::string& name)
    : Character(name, 30, 2.0/3.0, 20, 2.0/3.0, 50, 130)
{
}

int Dwarf::specialDefence(int damage)
{
    // Successful defences never cause damage
    return 0;
}

// Hobbit: Attack 25, AttackChance 1/3, Defence 20, DefenceChance 2/3, Health 70, Strengt
Hobbit::Hobbit(const std::string& name)
```

```
    : Character(name, 25, 1.0/3.0, 20, 2.0/3.0, 70, 85)
{
}

int Hobbit::specialDefence(int damage)
{
    // Successful defences cause 0-5 damage
    return rand() % 6;
}

// Orc: Attack 25(45), AttackChance 1/4(1/1), Defence 10(25), DefenceChance 1/4(1/2), Hea
Orc::Orc(const std::string& name)
    : Character(name, 25, 1.0/4.0, 10, 1.0/4.0, 50, 130), isNightTime(false)
{
}

int Orc::specialDefence(int damage)
{
    if (isNightTime) {
        // Night: increase health by 1
        health += 1;
        return 0;
    } else {
        // Day: 1/4 of adjusted damage
        return damage / 4;
    }
}

void Orc::setTimeOfDay(bool night)
{
    isNightTime = night;

    if (night) {
        attack = 45;
        attackChance = 1.0;
        defence = 25;
        defenceChance = 1.0/2.0;
    } else {
        attack = 25;
        attackChance = 1.0/4.0;
        defence = 10;
        defenceChance = 1.0/4.0;
    }
}
```

6. Item.h

```
/**
 * @file Item.h
 * @brief Abstract base class and concrete classes for items
 * @author [Your Name]
 * @date 2025-11-10
 */

#ifndef ITEM_H
```

```
#define ITEM_H

#include <string>

/**
 * @class Item
 * @brief Abstract base class for all items
 */
class Item
{
protected:
    std::string name;
    std::string category;
    int weight;
    int attackModifier;
    int defenceModifier;
    int healthModifier;
    int strengthModifier;

public:
    /**
     * @brief Constructor for Item
     * @param n Item name
     * @param c Item category
     * @param w Weight
     * @param a Attack modifier
     * @param d Defence modifier
     * @param h Health modifier
     * @param s Strength modifier
     */
    Item(const std::string& n, const std::string& c, int w, int a, int d, int h, int s);

    virtual ~Item() = default;

    // Getters
    std::string getName() const { return name; }
    std::string getCategory() const { return category; }
    int getWeight() const { return weight; }
    int getAttackModifier() const { return attackModifier; }
    int getDefenceModifier() const { return defenceModifier; }
    int getHealthModifier() const { return healthModifier; }
    int getStrengthModifier() const { return strengthModifier; }

    /**
     * @brief Pure virtual function to get item description
     * @return std::string Item description
     */
    virtual std::string getDescription() const = 0;
};

// Weapon classes
class Sword : public Item
{
public:
    Sword();
    std::string getDescription() const override;
```

```
};

class Dagger : public Item
{
public:
    Dagger();
    std::string getDescription() const override;
};

// Armour classes
class PlateArmour : public Item
{
public:
    PlateArmour();
    std::string getDescription() const override;
};

class LeatherArmour : public Item
{
public:
    LeatherArmour();
    std::string getDescription() const override;
};

// Shield classes
class LargeShield : public Item
{
public:
    LargeShield();
    std::string getDescription() const override;
};

class SmallShield : public Item
{
public:
    SmallShield();
    std::string getDescription() const override;
};

// Ring classes
class RingOfLife : public Item
{
public:
    RingOfLife();
    std::string getDescription() const override;
};

class RingOfStrength : public Item
{
public:
    RingOfStrength();
    std::string getDescription() const override;
};

#endif // ITEM_H
```

7. Item.cpp

```
/**
 * @file Item.cpp
 * @brief Implementation of Item classes
 */

#include "Item.h"

Item::Item(const std::string& n, const std::string& c, int w, int a, int d, int h, int s)
    : name(n), category(c), weight(w), attackModifier(a), defenceModifier(d),
      healthModifier(h), strengthModifier(s)
{
}

// Sword: weight 10, +10 attack
Sword::Sword() : Item("Sword", "Weapon", 10, 10, 0, 0, 0) {}
std::string Sword::getDescription() const { return "A sharp sword (+10 Attack)"; }

// Dagger: weight 5, +5 attack
Dagger::Dagger() : Item("Dagger", "Weapon", 5, 5, 0, 0, 0) {}
std::string Dagger::getDescription() const { return "A quick dagger (+5 Attack)"; }

// Plate Armour: weight 40, +10 defence, -5 attack
PlateArmour::PlateArmour() : Item("Plate Armour", "Armour", 40, -5, 10, 0, 0) {}
std::string PlateArmour::getDescription() const { return "Heavy plate armour (+10 Defence

// Leather Armour: weight 20, +5 defence
LeatherArmour::LeatherArmour() : Item("Leather Armour", "Armour", 20, 0, 5, 0, 0) {}
std::string LeatherArmour::getDescription() const { return "Light leather armour (+5 Defe

// Large Shield: weight 30, +10 defence, -5 attack
LargeShield::LargeShield() : Item("Large Shield", "Shield", 30, -5, 10, 0, 0) {}
std::string LargeShield::getDescription() const { return "A large protective shield (+10

// Small Shield: weight 10, +5 defence
SmallShield::SmallShield() : Item("Small Shield", "Shield", 10, 0, 5, 0, 0) {}
std::string SmallShield::getDescription() const { return "A small shield (+5 Defence)"; }

// Ring of Life: weight 1, +10 health
RingOfLife::RingOfLife() : Item("Ring of Life", "Ring", 1, 0, 0, 10, 0) {}
std::string RingOfLife::getDescription() const { return "A magical ring (+10 Health)"; }

// Ring of Strength: weight 1, +50 strength, -10 health
RingOfStrength::RingOfStrength() : Item("Ring of Strength", "Ring", 1, 0, 0, -10, 50) {}
std::string RingOfStrength::getDescription() const { return "A powerful ring (+50 Strengt
```

8. Location.h

```
/**
 * @file Location.h
 * @brief Class representing a board location/square
 * @author [Your Name]
 * @date 2025-11-10
```

```
 */

#ifndef LOCATION_H
#define LOCATION_H

#include "Character.h"
#include "Item.h"
#include <memory>

/**
 * @class Location
 * @brief Represents a single square on the game board
 */
class Location
{
private:
    int x;
    int y;
    std::shared_ptr<Character> enemy;
    std::shared_ptr<Item> item;
    bool isStartLocation;

public:
    /**
     * @brief Constructor for Location
     * @param xPos X coordinate
     * @param yPos Y coordinate
     */
    Location(int xPos, int yPos);

    /**
     * @brief Destructor
     */
    ~Location() = default;

    /**
     * @brief Set enemy at this location
     * @param e Shared pointer to enemy character
     */
    void setEnemy(std::shared_ptr<Character> e);

    /**
     * @brief Remove enemy from this location
     */
    void removeEnemy();

    /**
     * @brief Set item at this location
     * @param i Shared pointer to item
     */
    void setItem(std::shared_ptr<Item> i);

    /**
     * @brief Remove and return item from this location
     * @return std::shared_ptr<Item> The item or nullptr
     */
```

```
    std::shared_ptr<Item> removeItem();

    /**
     * @brief Check if location has enemy
     * @return bool True if enemy present
     */
    bool hasEnemy() const;

    /**
     * @brief Check if location has item
     * @return bool True if item present
     */
    bool hasItem() const;

    /**
     * @brief Get enemy at this location
     * @return std::shared_ptr<Character> Enemy or nullptr
     */
    std::shared_ptr<Character> getEnemy() const;

    /**
     * @brief Get item at this location
     * @return std::shared_ptr<Item> Item or nullptr
     */
    std::shared_ptr<Item> getItem() const;

    /**
     * @brief Display location information
     */
    void displayInfo() const;

    // Getters/Setters
    int getX() const { return x; }
    int getY() const { return y; }
    void setStartLocation(bool start) { isStartLocation = start; }
    bool getIsStartLocation() const { return isStartLocation; }
};

#endif // LOCATION_H
```

9. Location.cpp

```
/**
 * @file Location.cpp
 * @brief Implementation of Location class
 */

#include "Location.h"
#include <iostream>

Location::Location(int xPos, int yPos)
    : x(xPos), y(yPos), enemy(nullptr), item(nullptr), isStartLocation(false)
{
}
```

```
void Location::setEnemy(std::shared_ptr<Character> e)
{
    enemy = e;
}

void Location::removeEnemy()
{
    enemy.reset();
}

void Location::setItem(std::shared_ptr<Item> i)
{
    item = i;
}

std::shared_ptr<Item> Location::removeItem()
{
    std::shared_ptr<Item> temp = item;
    item.reset();
    return temp;
}

bool Location::hasEnemy() const
{
    return enemy != nullptr;
}

bool Location::hasItem() const
{
    return item != nullptr;
}

std::shared_ptr<Character> Location::getEnemy() const
{
    return enemy;
}

std::shared_ptr<Item> Location::getItem() const
{
    return item;
}

void Location::displayInfo() const
{
    std::cout << "\n=== Location (" << x << ", " << y << ") ===" << std::endl;

    if (hasEnemy()) {
        std::cout << "Enemy: " << enemy->getName()
                  << " (Health: " << enemy->getHealth() << ")" << std::endl;
    }

    if (hasItem()) {
        std::cout << "Item: " << item->getName() << std::endl;
        std::cout << "  " << item->getDescription() << std::endl;
    }
```

```
    if (!hasEnemy() && !hasItem()) {
        std::cout << "This location is empty." << std::endl;
    }
}
```

10. Board.h

```
/**
 * @file Board.h
 * @brief Class managing the game board
 * @author [Your Name]
 * @date 2025-11-10
 */

#ifndef BOARD_H
#define BOARD_H

#include "Location.h"
#include <vector>
#include <memory>

/**
 * @class Board
 * @brief Manages the game board with smart pointers
 */
class Board
{
private:
    int width;
    int height;
    std::vector<std::vector<std::shared_ptr<Location>>> grid;

public:
    /**
     * @brief Constructor for Board
     * @param w Width of board
     * @param h Height of board
     */
    Board(int w, int h);

    /**
     * @brief Initialize board with enemies and items
     */
    void initialize();

    /**
     * @brief Get location at coordinates
     * @param x X coordinate
     * @param y Y coordinate
     * @return std::shared_ptr<Location> Location at coordinates or nullptr
     */
    std::shared_ptr<Location> getLocation(int x, int y);

    /**
     * @brief Check if coordinates are valid
```

```
     * @param x X coordinate
     * @param y Y coordinate
     * @return bool True if valid
     */
    bool isValidLocation(int x, int y) const;

    /**
     * @brief Get starting location
     * @return std::shared_ptr<Location> Start location
     */
    std::shared_ptr<Location> getStartLocation();

    // Getters
    int getWidth() const { return width; }
    int getHeight() const { return height; }
};

#endif // BOARD_H
```

11. Board.cpp

```
/**
 * @file Board.cpp
 * @brief Implementation of Board class
 */

#include "Board.h"
#include "Race.h"
#include <cstdlib>
#include <ctime>

Board::Board(int w, int h) : width(w), height(h)
{
    /* Pseudo-code:
     * 1. Resize grid to width x height
     * 2. For each position (x, y):
     *    a. Create new Location using smart pointer
     *    b. Store in grid
     */
    grid.resize(height);
    for (int y = 0; y < height; ++y) {
        grid[y].resize(width);
        for (int x = 0; x < width; ++x) {
            grid[y][x] = std::make_shared<Location>(x, y);
        }
    }
}

void Board::initialize()
{
    /* Pseudo-code:
     * 1. Set random seed
     * 2. Set start location (e.g., 0,0)
     * 3. For each location:
     *    a. Generate random number
```

```
     *    b. If < 0.2, create random enemy
     *    c. Else if < 0.4, create random item
     *    d. Else leave empty
     */
    srand(time(nullptr));

    // Set start location
    grid[^0][^0]->setStartLocation(true);

    // Populate board
    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            // Skip start location
            if (x == 0 && y == 0) continue;

            double random = static_cast<double>(rand()) / RAND_MAX;

            if (random < 0.2) {
                // Create enemy (20% chance)
                int raceChoice = rand() % 5;
                std::shared_ptr<Character> enemy;

                switch (raceChoice) {
                    case 0: enemy = std::make_shared<Human>("Enemy Human"); break;
                    case 1: enemy = std::make_shared<Elf>("Enemy Elf"); break;
                    case 2: enemy = std::make_shared<Dwarf>("Enemy Dwarf"); break;
                    case 3: enemy = std::make_shared<Hobbit>("Enemy Hobbit"); break;
                    case 4: enemy = std::make_shared<Orc>("Enemy Orc"); break;
                }

                grid[y][x]->setEnemy(enemy);

            } else if (random < 0.4) {
                // Create item (20% chance)
                int itemChoice = rand() % 8;
                std::shared_ptr<Item> item;

                switch (itemChoice) {
                    case 0: item = std::make_shared<Sword>(); break;
                    case 1: item = std::make_shared<Dagger>(); break;
                    case 2: item = std::make_shared<PlateArmour>(); break;
                    case 3: item = std::make_shared<LeatherArmour>(); break;
                    case 4: item = std::make_shared<LargeShield>(); break;
                    case 5: item = std::make_shared<SmallShield>(); break;
                    case 6: item = std::make_shared<RingOfLife>(); break;
                    case 7: item = std::make_shared<RingOfStrength>(); break;
                }

                grid[y][x]->setItem(item);
            }
            // Otherwise leave empty (60% chance)
        }
    }
}

std::shared_ptr<Location> Board::getLocation(int x, int y)
```

```
{
    if (isValidLocation(x, y)) {
        return grid[y][x];
    }
    return nullptr;
}

bool Board::isValidLocation(int x, int y) const
{
    return x >= 0 && x < width && y >= 0 && y < height;
}

std::shared_ptr<Location> Board::getStartLocation()
{
    return grid[^0][^0];
}
```

12. Game.h

```
/**
 * @file Game.h
 * @brief Main game controller class
 * @author [Your Name]
 * @date 2025-11-10
 */

#ifndef GAME_H
#define GAME_H

#include "Board.h"
#include "Character.h"
#include <memory>

/**
 * @class Game
 * @brief Controls game flow and logic
 */
class Game
{
private:
    std::unique_ptr<Board> board;
    std::shared_ptr<Character> player;
    int currentX;
    int currentY;
    bool isRunning;
    int commandCount;
    bool isNightTime;

public:
    /**
     * @brief Constructor
     */
    Game();

    /**
```

```
     * @brief Main game loop
     */
    void run();

private:
    /**
     * @brief Setup game (board size, character selection)
     */
    void setup();

    /**
     * @brief Process player command
     * @param command Command string
     */
    void processCommand(const std::string& command);

    /**
     * @brief Move player in direction
     * @param dx Change in x
     * @param dy Change in y
     */
    void movePlayer(int dx, int dy);

    /**
     * @brief Handle combat between player and enemy
     * @param enemy Enemy character
     */
    void handleCombat(std::shared_ptr<Character> enemy);

    /**
     * @brief Player attacks enemy
     * @param attacker Attacking character
     * @param defender Defending character
     * @return bool True if defender defeated
     */
    bool performAttack(std::shared_ptr<Character> attacker,
                       std::shared_ptr<Character> defender);

    /**
     * @brief Pick up item at current location
     */
    void pickUpItem();

    /**
     * @brief Drop item from inventory
     */
    void dropItem();

    /**
     * @brief Display current location info
     */
    void look();

    /**
     * @brief Update time of day
     */
```

```
    void updateTime();

    /**
     * @brief Display game over screen
     */
    void gameOver();
};

#endif // GAME_H
```

13. Game.cpp

```
/**
 * @file Game.cpp
 * @brief Implementation of Game class
 */

#include "Game.h"
#include "Race.h"
#include <iostream>
#include <algorithm>
#include <cctype>

Game::Game()
    : board(nullptr), player(nullptr), currentX(0), currentY(0),
      isRunning(true), commandCount(0), isNightTime(false)
{
}

void Game::run()
{
    std::cout << "================================" << std::endl;
    std::cout << "    FANTASY GAME" << std::endl;
    std::cout << "================================" << std::endl;

    setup();

    std::cout << "\nGame started! Type 'help' for commands." << std::endl;

    while (isRunning && player->isAlive()) {
        std::cout << "\n> ";
        std::string command;
        std::getline(std::cin, command);

        // Convert to lowercase
        std::transform(command.begin(), command.end(), command.begin(),
                       [](unsigned char c){ return std::tolower(c); });

        processCommand(command);
    }

    if (!player->isAlive()) {
        gameOver();
    }
}
```

```
void Game::setup()
{
    /* Pseudo-code:
     * 1. Get board dimensions from player
     * 2. Create board with smart pointer
     * 3. Initialize board
     * 4. Display race options
     * 5. Get player race choice
     * 6. Create player character
     * 7. Set starting position
     */
    int width, height;

    std::cout << "\nEnter board width: ";
    std::cin >> width;
    std::cout << "Enter board height: ";
    std::cin >> height;
    std::cin.ignore(); // Clear newline

    board = std::make_unique<Board>(width, height);
    board->initialize();

    std::cout << "\nSelect your race:" << std::endl;
    std::cout << "1. Human" << std::endl;
    std::cout << "2. Elf" << std::endl;
    std::cout << "3. Dwarf" << std::endl;
    std::cout << "4. Hobbit" << std::endl;
    std::cout << "5. Orc" << std::endl;

    int choice;
    std::cout << "Enter choice (1-5): ";
    std::cin >> choice;
    std::cin.ignore();

    std::string playerName;
    std::cout << "Enter your character name: ";
    std::getline(std::cin, playerName);

    switch (choice) {
        case 1: player = std::make_shared<Human>(playerName); break;
        case 2: player = std::make_shared<Elf>(playerName); break;
        case 3: player = std::make_shared<Dwarf>(playerName); break;
        case 4: player = std::make_shared<Hobbit>(playerName); break;
        case 5: player = std::make_shared<Orc>(playerName); break;
        default: player = std::make_shared<Human>(playerName);
    }

    currentX = 0;
    currentY = 0;
}

void Game::processCommand(const std::string& command)
{
    /* Pseudo-code:
     * 1. Parse command (use first character)
```

```
     * 2. Switch on command type:
     *    - n/s/e/w: move
     *    - p: pick up
     *    - d: drop
     *    - a: attack
     *    - l: look
     *    - i: inventory
     *    - e: exit
     * 3. Increment command count
     * 4. Update time if needed
     */

    if (command.empty()) return;

    char cmd = command[^0];

    switch (cmd) {
        case 'n': // North
            movePlayer(0, -1);
            commandCount++;
            break;
        case 's': // South
            movePlayer(0, 1);
            commandCount++;
            break;
        case 'e': // East
            movePlayer(1, 0);
            commandCount++;
            break;
        case 'w': // West
            movePlayer(-1, 0);
            commandCount++;
            break;
        case 'p': // Pick up
            pickUpItem();
            commandCount++;
            break;
        case 'd': // Drop
            dropItem();
            commandCount++;
            break;
        case 'a': // Attack
            {
                auto location = board->getLocation(currentX, currentY);
                if (location->hasEnemy()) {
                    handleCombat(location->getEnemy());
                } else {
                    std::cout << "No enemy to attack here!" << std::endl;
                }
            }
            commandCount++;
            break;
        case 'l': // Look
            look();
            break;
        case 'i': // Inventory
```

```
            player->displayInventory();
            break;
        case 'x': // Exit
            isRunning = false;
            std::cout << "Thanks for playing! Final gold: " << player->getGold() << std::
            break;
        default:
            std::cout << "Commands: (N)orth, (S)outh, (E)ast, (W)est, " << std::endl;
            std::cout << "         (P)ick up, (D)rop, (A)ttack, (L)ook, " << std::endl;
            std::cout << "         (I)nventory, E(x)it" << std::endl;
    }

    updateTime();
}

void Game::movePlayer(int dx, int dy)
{
    int newX = currentX + dx;
    int newY = currentY + dy;

    if (board->isValidLocation(newX, newY)) {
        currentX = newX;
        currentY = newY;
        std::cout << "Moved to (" << currentX << ", " << currentY << ")" << std::endl;
        look();
    } else {
        std::cout << "Cannot move in that direction!" << std::endl;
    }
}

void Game::handleCombat(std::shared_ptr<Character> enemy)
{
    /* Pseudo-code:
     * 1. Display combat start message
     * 2. Player attacks enemy
     * 3. If enemy alive, enemy attacks player
     * 4. Display combat results
     * 5. If enemy defeated, award gold and remove from board
     * 6. If player defeated, end game
     */

    std::cout << "\n=== COMBAT ===" << std::endl;
    std::cout << "Fighting " << enemy->getName() << std::endl;

    // Player attacks
    bool enemyDefeated = performAttack(player, enemy);

    if (enemyDefeated) {
        int goldReward = enemy->getTotalDefence();
        player->addGold(goldReward);
        std::cout << "Enemy defeated! Gained " << goldReward << " gold." << std::endl;

        auto location = board->getLocation(currentX, currentY);
        location->removeEnemy();
        return;
    }
```

```
    // Enemy counter-attacks
    std::cout << "\nEnemy counter-attacks!" << std::endl;
    performAttack(enemy, player);

    if (!player->isAlive()) {
        std::cout << "You have been defeated!" << std::endl;
        isRunning = false;
    }
}

bool Game::performAttack(std::shared_ptr<Character> attacker,
                         std::shared_ptr<Character> defender)
{
    /* Pseudo-code:
     * 1. Check if attack succeeds (attack chance)
     * 2. If attack fails, return false
     * 3. Check if defence succeeds (defence chance)
     * 4. Calculate damage based on defence result
     * 5. Apply damage to defender
     * 6. Return true if defender health <= 0
     */

    std::cout << attacker->getName() << " attacks!" << std::endl;

    if (!attacker->attemptAttack()) {
        std::cout << "Attack missed!" << std::endl;
        return false;
    }

    std::cout << "Attack successful!" << std::endl;

    if (defender->attemptDefence()) {
        std::cout << defender->getName() << " defends!" << std::endl;
        int damage = defender->specialDefence(
            attacker->getTotalAttack() - defender->getTotalDefence());
        defender->takeDamage(damage);
        std::cout << "Damage: " << damage << std::endl;
    } else {
        int damage = attacker->getTotalAttack() - defender->getTotalDefence();
        if (damage < 0) damage = 0;
        defender->takeDamage(damage);
        std::cout << "Defence failed! Damage: " << damage << std::endl;
    }

    std::cout << defender->getName() << " Health: " << defender->getHealth() << std::endl

    return !defender->isAlive();
}

void Game::pickUpItem()
{
    auto location = board->getLocation(currentX, currentY);

    if (!location->hasItem()) {
        std::cout << "No item here to pick up!" << std::endl;
```

```
        return;
    }

    auto item = location->getItem();

    if (player->addItem(item)) {
        location->removeItem();
        std::cout << "Picked up: " << item->getName() << std::endl;
    } else {
        std::cout << "Cannot pick up item (too heavy or category full)!" << std::endl;
    }
}

void Game::dropItem()
{
    auto location = board->getLocation(currentX, currentY);

    if (location->hasItem()) {
        std::cout << "Location already has an item!" << std::endl;
        return;
    }

    std::cout << "Enter item name to drop: ";
    std::string itemName;
    std::getline(std::cin, itemName);

    auto item = player->removeItem(itemName);

    if (item) {
        location->setItem(item);
        std::cout << "Dropped: " << itemName << std::endl;
    } else {
        std::cout << "Item not found in inventory!" << std::endl;
    }
}

void Game::look()
{
    auto location = board->getLocation(currentX, currentY);
    location->displayInfo();

    std::cout << "\nTime: " << (isNightTime ? "Night" : "Day") << std::endl;
    std::cout << "Player Health: " << player->getHealth() << std::endl;
}

void Game::updateTime()
{
    if (commandCount > 0 && commandCount % 5 == 0) {
        isNightTime = !isNightTime;
        std::cout << "\n*** Time changes to " << (isNightTime ? "NIGHT" : "DAY") << " ***

        // Update Orc stats if player or enemies are Orcs
        auto orcPlayer = std::dynamic_pointer_cast<Orc>(player);
        if (orcPlayer) {
            orcPlayer->setTimeOfDay(isNightTime);
        }
```

```
        // Update all Orc enemies on board
        for (int y = 0; y < board->getHeight(); ++y) {
            for (int x = 0; x < board->getWidth(); ++x) {
                auto location = board->getLocation(x, y);
                if (location->hasEnemy()) {
                    auto orcEnemy = std::dynamic_pointer_cast<Orc>(location->getEnemy());
                    if (orcEnemy) {
                        orcEnemy->setTimeOfDay(isNightTime);
                    }
                }
            }
        }
    }
}

void Game::gameOver()
{
    std::cout << "\n================================" << std::endl;
    std::cout << "         GAME OVER" << std::endl;
    std::cout << "================================" << std::endl;
    std::cout << "Final Gold Collected: " << player->getGold() << std::endl;
    std::cout << "================================" << std::endl;
}
```

Additional Files Needed

14. FantasyGame.pro (Qt Project File)

```
QT -= gui

CONFIG += c++11 console
CONFIG -= app_bundle

SOURCES += \
    main.cpp \
    Game.cpp \
    Character.cpp \
    Race.cpp \
    Item.cpp \
    Location.cpp \
    Board.cpp

HEADERS += \
    Game.h \
    Character.h \
    Race.h \
    Item.h \
    Location.h \
    Board.h
```

Key Features of This Boilerplate

Object-Oriented Design[1]

- Abstract base classes (`Character`, `Item`) with pure virtual functions
- Inheritance hierarchy for races and items
- Polymorphism through virtual functions

Smart Pointers[1]

- `std::shared_ptr` for Characters and Items (multiple ownership)
- `std::unique_ptr` for Board (single ownership)
- All Location objects managed by smart pointers

Dynamic Data Structures[1]

- `std::vector` for inventory (unbounded items support)
- 2D vector for board grid
- STL containers used throughout

Complete Implementation Features[1]

- All 5 races with unique abilities (Human, Elf, Dwarf, Hobbit, Orc)
- All 8 items with proper modifiers
- Day/night cycle (every 5 commands)
- Combat system with attack/defence chances
- Movement system with boundary checking
- Inventory management with weight limits
- Category restrictions (one weapon/armor/shield, unlimited rings)

Code Quality[1]

- Doxygen-style comments on all classes and functions
- Pseudo-code for non-trivial functions
- Proper file structure with .h/.cpp separation
- K&R formatting style ready

This boilerplate provides a solid foundation to build upon and meets all the core requirements specified in the assignment.[1]

⁂

1. EE6161-2025-Assignment2FantasyGame.pdf