```
/* ------------------------------------------------------------
// map.h
// ------------------------------------------------------------ */
#ifndef MAP_H
#define MAP_H

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

constexpr int MAX_NAME_LENGTH = 10;
constexpr int MAX_PLAYER = 4;
constexpr int MAX_LEVEL = 5;
constexpr int MIN_LEVEL = 1;

constexpr int UPGRADABLEUNIT = 1;
constexpr int COLLECTABLEUNIT = 2;
constexpr int RANDOMCOSTUNIT = 3;
constexpr int JAILUNIT = 4;

// ============ MapUnit ============
class MapUnit {
public:
    // Constructor & Destructor
    MapUnit(int id, const std::string &name, int price, WorldMap* worldMap);
    virtual ~MapUnit() = default;

    virtual int event(Player &player) = 0;

    void addPlayer(Player *player);
    void removePlayer(Player *player);

    int getId() const;
    const std::string& getName() const;
    Player* getOwner() const;
    int getPrice() const;
    const std::vector<Player*>& whoishere() const;

    void setOwner(Player *owner);
    void setWorldMap(WorldMap* map);
    virtual void releaseOwner(Player *player);

    virtual void printUnit(std::ostream &os) const;

protected:
    int id_;
    std::string name_;
    int price_;
    Player *owner_;
    std::vector<Player*> whoishere_;
    WorldMap* worldMap_;
};

// ============ UpgradableUnit ============
class UpgradableUnit : public MapUnit {
public:
    // Constructor & Destructor
    UpgradableUnit(int id, const std::string &name, int price, int
upgrade_price, int base_fine, WorldMap* worldMap);
    ~UpgradableUnit() override = default;

    bool isOwned() const;
    bool isUpgradable() const;
```

```
    int calculateFine() const;

    void upgrade();
    void reset();

    int getLevel() const;
    int getUpgradePrice() const;
    int getBaseFine() const;

    void releaseOwner(Player *player) override;
    int event(Player &player) override;
    void printUnit(std::ostream &os) const override;

private:
    int level_;
    int upgradePrice_;
    int baseFine_;
};


// ============ CollectableUnit ============
class CollectableUnit : public MapUnit {
public:
    // Constructor & Destructor
    CollectableUnit(int id, const std::string &name, int price, int fine,
WorldMap* worldMap);
    ~CollectableUnit() override = default;

    void printUnit(std::ostream &os) const override;

    int getFine() const;

    int calculateFine() const;
    void releaseOwner(Player* player) override;

    int event(Player &playe) override;

private:
    int fine_;
};

// ============ RandomCostUnit ============
class RandomCostUnit : public MapUnit {
public:
    // Constructor & Destructor
    RandomCostUnit(int id, const std::string &name, int price, int fine,
WorldMap* worldMap);
    ~RandomCostUnit() override = default;

    void printUnit(std::ostream &os) const override;

    int getFine() const;

    void releaseOwner(Player* player) override;
    int calculateFine() const;
    int event(Player &player) override;

private:
    int fine_;
};
```

```
// ========== JailUnit ============
class JailUnit : public MapUnit {
public:
    // Constructor & Destructor
    JailUnit(int id, const std::string &name, WorldMap* worldMap);
    ~JailUnit() override = default;

    void printUnit(std::ostream &os) const override;
    int event(Player &player) override;
};

int rollDice();

#endif // MAP_H


/* ------------------------------------------------------------------
// map.cpp
// ------------------------------------------------------------------ */
#include <iostream>
#include <chrono>
#include <thread>
#include <ctime>
#include <random>
#include "map.h"
#include "WorldMap.h"

using namespace std;

// ============ MapUnit ============
MapUnit::MapUnit(int id, const string &name, int price, WorldMap* worldMap)
    : id_(id), name_(name), price_(price), owner_(nullptr) , worldMap_(worldMap)
{}

void MapUnit::addPlayer(Player *player) {
    whoishere_.push_back(player);
}

void MapUnit::removePlayer(Player *player) {
    auto it = remove(whoishere_.begin(), whoishere_.end(), player);
    if (it != whoishere_.end()) {
        whoishere_.erase(it);
    }
}

int MapUnit::getId() const {
    return id_;
}

const string& MapUnit::getName() const {
    return name_;
}

int MapUnit::getPrice() const {
    return price_;
}

Player* MapUnit::getOwner() const {
    return owner_;
}

const vector<Player*>& MapUnit::whoishere() const {
    return whoishere_;
```

```
}

void MapUnit::setOwner(Player *owner) {
    owner_ = owner;
    owner->addUnit();
}

void MapUnit::setWorldMap(WorldMap* map) {
    worldMap_ = map;
}

void MapUnit::releaseOwner(Player *player) {
    owner_ = nullptr;
}



void MapUnit::printUnit(ostream &os) const {
    os << "[MapUnit] " << getName() << "\n";
}

// ============ UpgradableUnit ============
UpgradableUnit::UpgradableUnit(int id, const string &name, int price, int
upgrade_price, int base_fine, WorldMap* worldMap)
    : MapUnit(id, name, price, worldMap), level_(MIN_LEVEL),
upgradePrice_(upgrade_price), baseFine_(base_fine) {}

bool UpgradableUnit::isOwned() const {
    return owner_ != nullptr;
}

bool UpgradableUnit::isUpgradable() const {
    return isOwned() && level_ < MAX_LEVEL;
}

int UpgradableUnit::calculateFine() const {
    return baseFine_ * level_;
}

void UpgradableUnit::upgrade() {
    if (isUpgradable()) {
        ++level_;
    }
}

void UpgradableUnit::reset() {
    level_ = 1; // Reset to initial level
    owner_ = nullptr;
}

int UpgradableUnit::getLevel() const {
    return level_;
}

int UpgradableUnit::getUpgradePrice() const {
    return upgradePrice_;
}

int UpgradableUnit::getBaseFine() const {
    return baseFine_;
}

void UpgradableUnit::releaseOwner(Player* player) {
    owner_ = nullptr;
```

```
    level_ = MIN_LEVEL;
    cout << player->getName() << " has released ownership of " << getName() <<
".\n";
}

int UpgradableUnit::event(Player &player) {
    if (!isOwned()) {
        if (player.getMoney() >= price_) {
            // Player can choose to buy the unit
            cout << "This land is unowned. Buy it for $" << price_ << "? (y/n):
";
            string answer;
            cin >> answer;

            if (checkAnswer(answer) && (answer[0] == 'y' || answer[0] == 'Y')) {
                player.deduct(price_);
                setOwner(&player);
                cout << "You bought " << getName() << " for $" << price_ << ".
\n";
            } else {
                cout << "You chose not to buy " << getName() << ".\n";
            }
        }
        else {
            cout << "You cannot afford this Upgradable Unit.\n";
        }

    }
    else if (*owner_ == player) {
        // Player owns the unit, can upgrade if possible
        if (isUpgradable()) {
            // Player can choose to upgrade
            cout << "Upgrade this land for $" << upgradePrice_ << "? (y/n): ";
            string answer;
            cin >> answer;
            if (checkAnswer(answer) && (answer[0] == 'y' || answer[0] == 'Y')) {
                if (player.deduct(upgradePrice_)) {
                    upgrade();
                    cout << "Upgraded to level " << level_ << ".\n";
                } else {
                    cout << "Not enough money to upgrade.\n";
                }
            }
        } else {
            cout << "Already at max level.\n";
        }
    }
    else {
        // Player must pay the fine
        int totalFine = calculateFine();
        int leftMoney = player.getMoney();
        cout << "Owned by " << owner_->getName() << ". Paying fine $" <<
totalFine << ".\n";
        if (player.deduct(totalFine)) {
            owner_->earnings(totalFine);
        }
        else {
            // Insufficient funds to pay the fine, player go bankrupt
            owner_->earnings(leftMoney); // Owner still earns the fine
            player.changeStatus(dead);
            // Handle player bankruptcy (e.g., remove from game, transfer units)
            for (int i = 0; i < worldMap_->size(); ++i) {
                MapUnit* unit = worldMap_->getUnit(i);
                if (unit && unit->getOwner() == &player) {
```

```
                    unit->releaseOwner(&player);
                }
            }
        }
    }
    std::this_thread::sleep_for(std::chrono::seconds(2));
    return UPGRADABLEUNIT;
}

void UpgradableUnit::printUnit(ostream &os) const {
    os << "[UpgradableUnit] " << getName() << " | Price: $" << getPrice()
       << " | Owner: " << (owner_ ? owner_->getName() : "None")
       << " | Level: " << level_
       << " | Fine: $" << calculateFine() << "\n";
}



// ============ CollectableUnit ============
CollectableUnit::CollectableUnit(int id, const string &name, int price, int
fine, WorldMap* worldMap)
    : MapUnit(id, name, price, worldMap), fine_(fine) {}

int CollectableUnit::getFine() const {
    return fine_;
}

int CollectableUnit::calculateFine() const {
    if (!owner_) {
        return 0;
    }
    // Count the number of Collectable units owned by the player
    return owner_->getNumberOfCollectableUnits() * fine_;
}

void CollectableUnit::releaseOwner(Player* player) {
    owner_ = nullptr;
}

int CollectableUnit::event(Player &player) {
    if (!owner_) {
        if (player.getMoney() >= price_) {
            // Player can choose to buy the unit
            cout << "This land is unowned. Buy it for $" << price_ << "? (y/n):
";
            string answer;
            cin >> answer;

            if (checkAnswer(answer) && (answer[0] == 'y' || answer[0] == 'Y')) {
                if (player.deduct(price_)) {
                    setOwner(&player);
                    player.addCollectableUnit();
                    cout << "Purchased successfully.\n";
                } else {
                    cout << "Insufficient funds.\n";
                }
            }
        }
        else
        {
            cout << "You cannot afford this Collectable Unit.\n";
        }
    }
    else if (owner_ != &player) {
```

```
        // Player must pay the fine to owner
        int totalFine = calculateFine();
        int leftMoney = player.getMoney();

        cout << "Owned by " << owner_->getName() << ". Pay fine $" << totalFine
<< ".\n";

        if (player.deduct(totalFine)) {
            owner_->earnings(totalFine);
        }
        else {
            owner_->earnings(leftMoney); // Owner still earns the fine
            player.changeStatus(dead);
            // Handle player bankruptcy (e.g., remove from game, transfer units)
            for (int i = 0; i < worldMap_->size(); ++i) {
                MapUnit* unit = worldMap_->getUnit(i);
                if (unit && unit->getOwner() == &player) {
                    unit->releaseOwner(&player);
                }
            }

        }
    } else {
        cout << "You own this Collectable Unit.\n";
    }
    std::this_thread::sleep_for(std::chrono::seconds(2));
    return COLLECTABLEUNIT;
}

void CollectableUnit::printUnit(ostream &os) const {
    os << "[CollectableUnit] " << getName() << " | Price: $" << getPrice()
       << " | Owner: " << (owner_ ? owner_->getName() : "None")
       << " | Fine per unit: $" << fine_ << "\n";
}


// ============ RandomCostUnit ============
RandomCostUnit::RandomCostUnit(int id, const string &name, int price, int fine,
WorldMap* worldMap)
    : MapUnit(id, name, price, worldMap), fine_(fine) {}

int RandomCostUnit::getFine() const {
    return fine_;
}

int RandomCostUnit::calculateFine() const {
    return rollDice() * fine_;
}

int RandomCostUnit::event(Player &player) {
    if (!owner_) {
        if (player.getMoney() >= price_) {
            // Player can choose to buy the unit.
            cout << "This land is unowned. Buy it for $" << price_ << "? (y/n):
";
            string answer;
            cin >> answer;

            if (checkAnswer(answer) && (answer[0] == 'y' || answer[0] == 'Y')) {
                if (player.deduct(price_)) {
                    setOwner(&player);
                    cout << "Purchased successfully.\n";
                } else {
                    cout << "Insufficient funds.\n";
```

```
                }
            }
        }
        else {
            cout << "You cannot afford this Random Cost Unit.\n";
        }
    }
    else if (owner_ != &player) {
        // Player must pay the fine to owner
        int totalFine = calculateFine();
        int leftMoney = player.getMoney();

        cout << "Random fine rolled. Pay $" << totalFine << ".\n";

        if (player.deduct(totalFine)) {
            owner_->earnings(totalFine);
        }
        else {
            owner_->earnings(leftMoney); // Owner still earns the fine
            player.changeStatus(dead);
            // Handle player bankruptcy (e.g., remove from game, transfer units)
            for (int i = 0; i < worldMap_->size(); ++i) {
                MapUnit* unit = worldMap_->getUnit(i);
                if (unit && unit->getOwner() == &player) {
                    unit->releaseOwner(&player);
                }
            }
        }
    } else {
        cout << "You own this RandomCost Unit.\n";
    }
    std::this_thread::sleep_for(std::chrono::seconds(2));
    return RANDOMCOSTUNIT;
}

int rollDice() {
    static mt19937 gen(time(0));
    uniform_int_distribution<> dist(1, 6);
    return dist(gen);
}

void RandomCostUnit::releaseOwner(Player* player) {
    owner_ = nullptr;
}

void RandomCostUnit::printUnit(ostream &os) const {
    os << "[RandomCostUnit] " << getName() << " | Price: $" << getPrice()
       << " | Owner: " << (owner_ ? owner_->getName() : "None")
       << " | Fine Multiplier: $" << fine_ << " per dice roll\n";
}

// ========== JailUnit ============
JailUnit::JailUnit(int id, const string &name, WorldMap* worldMap)
    : MapUnit(id, name, 0, worldMap) {}

int JailUnit::event(Player &player) {
    // Handle jail event for player
    cout << player.getName() << " landed in JAIL! You will miss the next round.
\n";
    player.changeStatus(jail);
    return JAILUNIT;
}

void JailUnit::printUnit(ostream &os) const {
```

```
    os << "[JailUnit] " << getName() << " | JAIL\n";
}


/* ----------------------------------------------------------------
// player.h
// ---------------------------------------------------------------- */
#ifndef PLAYER_H
#define PLAYER_H
#include <string>
#include <vector>

constexpr int initDeposit=30000;
constexpr int maxPlayersNum=4;
constexpr int minPlayersNum=1;
constexpr int alive=1;
constexpr int jail=2;
constexpr int dead=3;

class WorldMap;

class Player{
    public:
    Player(int id, std::string name);
    bool operator==(const Player& other) const;

    void changeName(const std::string& newName);
    bool deduct(int const cost);
    void earnings(int const toll);
    void move(int const rolledNum, WorldMap& map_);
    void addUnit();
    void addCollectableUnit();
    void changeStatus(int const status);

    std::string getName() const;
    int getID() const;
    int getLocation() const;
    int getMoney() const;
    int getNumberOfUnits() const;
    int getNumberOfCollectableUnits() const;
    int getStatus() const;

    private:
        const int id_;
        std::string name_="";
        int location_=0;
        int money_=initDeposit;
        int numUnits_=0;
        int numCollectableUnits_=0;
        int status_=alive;
};
class WorldPlayer{
    public:
        WorldPlayer(int numPlayers, WorldMap* map);
        WorldPlayer& operator++();
        WorldPlayer operator++(int);

        bool Action1();//new round
        int Action2();//after rolled the dice
        bool gameOver();

        Player& getPlayer(int index);
        int currentPlayerIs() const;
        int getCurrentPlayerID() const;
```

```
        int getNumPlayers() const;

    private:
        int numPlayers_=1;
        std::vector<Player> players_;
        const std::vector<std::string> defaultName_={"Frieren", "Himmel",
"Heiter", "Eisen"};
        int currentPlayer_=0;
        WorldMap *map_=nullptr;
};

bool checkAnswer(const std::string& answer);
std::ostream& operator<<(std::ostream& os,const Player& player);
std::ostream& operator<<(std::ostream& os,WorldPlayer& players);
std::istream& operator>>(std::istream& is, Player& player);
bool wantExit();
bool checkNum(const std::string& answer);
void displayScreen(WorldMap &map, WorldPlayer &players);


#endif


/* ------------------------------------------------------------
// player.cpp
// ------------------------------------------------------------ */
#include <iostream>
#include "player.h"
#include <string>
#include <vector>
#include <set>
#include <sstream>
#include "map.h"
#include "WorldMap.h"

using namespace std;

Player::Player(int id, string name):id_(id), name_(name){}
bool Player::operator==(const Player& other) const {
    return id_ == other.id_;
}
void Player::changeName(const string& newName)
{
    name_ = newName;
}
bool Player::deduct(int const cost)
{
    if (cost>money_)
    {
        cout<<name_<<", you're bankrupt!"<<endl;
        status_=dead;
        return false;
    }
    money_-=cost;
    return true;
}
void Player::earnings(int const toll)
{
    money_+=toll;
}
void Player::move(int const rolledNum, WorldMap& map_)
{
    int originLoc=location_;
    location_=(location_+rolledNum)%map_.size();
```

```
    if (location_<originLoc)
    {
        earnings(2000);
    }
}
void Player::addUnit()
{
    numUnits_++;
}
void Player::addCollectableUnit()
{
    numCollectableUnits_++;
}
void Player::changeStatus(int const status)
{
    status_=status;
}

std::string Player::getName() const
{
    return name_;
}
int Player::getID() const
{
    return id_;
}
int Player::getLocation() const
{
    return location_;
}
int Player::getMoney() const
{
    return money_;
}
int Player::getNumberOfUnits() const
{
    return numUnits_;
}
int Player::getNumberOfCollectableUnits() const
{
    return numCollectableUnits_;
}
int Player::getStatus() const
{
    return status_;
}
WorldPlayer::WorldPlayer(int numPlayers, WorldMap* map):numPlayers_(numPlayers),
map_(map)
{
    for (int i = 0; i < numPlayers_; ++i)
    {
        players_.emplace_back(i, defaultName_[i]);
    }
}
WorldPlayer& WorldPlayer::operator++()
{
    for (int i = 1; i < maxPlayersNum; i++)
    {
        if (players_[(currentPlayer_+i)%numPlayers_].getStatus()!=dead)
        {
            if (players_[(currentPlayer_+i)%numPlayers_].getStatus()==jail)
            {
                players_[(currentPlayer_+i)%numPlayers_].changeStatus(alive);
                continue;
```

```
            }
            currentPlayer_=(currentPlayer_+i)%numPlayers_;
            break;
        }

    }
    return *this;
}
WorldPlayer WorldPlayer::operator++(int)
{
    WorldPlayer temp= *this;
    currentPlayer_=(currentPlayer_+1)%numPlayers_;
    return temp;
}
bool WorldPlayer::Action1()//new round
{
    cout<<players_[currentPlayer_].getName()<<", it's your turn. Do you want to
roll the dice?(y/n) ";
    string answer;
    cin>>answer;cin.ignore();
    if (checkAnswer(answer))
    {
        if (answer[0]=='n'||answer[0]=='N')
        {
            return false;
        }
        else
        {
            int rolledNum=rollDice();
            players_[currentPlayer_].move(rolledNum, *map_);
            displayScreen(*map_, *this);
            cout<<players_[currentPlayer_].getName()<<", you
rolled:"<<rolledNum<<endl;
            return true;
        }
    }
    cerr<<"Invalid input."<<endl;
    return Action1();
}
int WorldPlayer::Action2()//after rolled the dice
{
    int type=(*map_).getUnit(players_[currentPlayer_].getLocation())-
>event(players_[currentPlayer_]);

    return type;
}
bool WorldPlayer::gameOver()
{
    int count=0;
    int winner=-1;
    for (int i = 0; i < numPlayers_; i++)
    {
        if (players_[i].getStatus()==dead)
        {
            count++;
        }
        else
        {
            winner=i;
        }
    }
    if (count>=(numPlayers_-1))
    {
```

```
        cout<<players_[winner].getName()<<", congratulations on your
victory!"<<endl;
        return true;
    }
    return false;
}
Player& WorldPlayer::getPlayer(int index)
{
    return players_.at(index);
}
int WorldPlayer::currentPlayerIs() const
{
    cout<<"It's "<<players_[currentPlayer_].getName()<<"'s turn."<<endl;
    return currentPlayer_;
}
int WorldPlayer::getCurrentPlayerID() const
{
    return currentPlayer_;
}
int WorldPlayer::getNumPlayers() const
{
    return numPlayers_;
}
bool checkAnswer(const std::string& answer)
{
    static const std::set<char> valid{'y','Y','n','N'};
    return answer.size() == 1 && valid.count(answer[0]);
}
bool checkNum(const std::string& answer)
{
    static const std::set<char> valid{'1','2','3','4'};
    return answer.size() == 1 && valid.count(answer[0]);
}
std::ostream& operator<<(std::ostream& os,const Player& player)
{
    os<<"["<<player.getID()<<"]"<<player.getName()<<" $"<<player.getMoney()<<"
with "<<player.getNumberOfUnits()<<" Units"<<endl;
    return os;
}
std::ostream& operator<<(std::ostream& os, WorldPlayer& players)
{
    for (int i = 0; i < players.getNumPlayers(); i++)
    {
        if (players.getPlayer(i).getStatus()!=dead)
        {
            os << players.getPlayer(i);
        }
    }
    return os;
}
std::istream& operator>>(std::istream& is,Player& player)
{
    string a;
    is>>a;
    player.changeName(a);
    return is;
}
bool wantExit()
{
    cout<<"End Game?(y/n)";
    string answer;
    cin>>answer;cin.ignore();
    if (checkAnswer(answer))
    {
```

```
        if (answer[0]=='y'||answer[0]=='Y') return true;
        if (answer[0]=='n'||answer[0]=='N') return false;
    }
    cout<<"Invalid input."<<endl;
    return wantExit();
}
void displayScreen(WorldMap &map, WorldPlayer &players)
{
    system("clear");
    map.display(players);
    cout<<players;
}
```

```
/* ----------------------------------------------------------------
// WorldMap.h
// ---------------------------------------------------------------- */
#ifndef WORLD_MAP_H
#define WORLD_MAP_H
#include <string>
#include "map.h"

class WorldMap {
public:
    WorldMap() {
        for (int i = 0; i < 20; i++) {
            units_[i] = nullptr;
        }
    };

    ~WorldMap() {
        for (int i = 0; i < 20; ++i) {
            delete units_[i];
            units_[i] = nullptr;
        }
    }
    size_t size() const;
    void loadFromFile(const std::string& filename);
    MapUnit* getUnit(int index) const;
    void display(WorldPlayer& worldPlayer) const;
private:
    MapUnit* units_[20];

    std::string formatUnitDisplay(int i,  WorldPlayer& worldPlayer) const;
};

#endif
```

```
/* ----------------------------------------------------------------
// WorldMap.cpp
// ---------------------------------------------------------------- */
#include "WorldMap.h"
#include "player.h"
#include <fstream>
#include <sstream>
#include <iostream>
using namespace std;

void WorldMap::loadFromFile(const std::string& filename) {
    std::ifstream in(filename);
    int idx = 0;
```

```
    if (in.is_open()) {
        std::string line;
        while (getline(in, line)) {
            if (idx >= 20) break;  // Prevent overflow
            std::istringstream iss(line);
            char type;
            std::string name;
            iss >> type >> name;

            if (type == 'U') {
                int price, upgrade_price;
                iss >> price >> upgrade_price;
                std::vector<int> fines(5);
                for (int& fine : fines) iss >> fine;
                units_[idx++] = new UpgradableUnit(idx - 1, name, price,
upgrade_price, fines[0], this);
                std::cout << "Loaded UpgradableUnit: " << name << " with price:
" << price << " and upgrade price: " << upgrade_price << "\n";
            }
            else if (type == 'C') {
                int price, fine;
                iss >> price >> fine;
                units_[idx++] = new CollectableUnit(idx - 1, name, price, fine,
this);
                std::cout << "Loaded CollectableUnit: " << name << " with price:
" << price << " and fine: " << fine << "\n";
            }
            else if (type == 'R') {
                int price, fine;
                iss >> price >> fine;
                units_[idx++] = new RandomCostUnit(idx - 1, name, price, fine,
this);
                std::cout << "Loaded RandomCostUnit: " << name << " with price:
" << price << " and fine: " << fine << "\n";
            }
            else if (type == 'J') {
                units_[idx++] = new JailUnit(idx - 1, name, this);
                std::cout << "Loaded JailUnit: " << name << "\n";
            }
        }
        in.close();
    }
}

MapUnit* WorldMap::getUnit(int index) const {
    if (index < 0 || index >= 20) return nullptr;
    return units_[index];
}

size_t WorldMap::size() const {
    size_t count = 0;
    for (const auto& unit : units_) {
        if (unit != nullptr) ++count;
    }
    return count;
}


void WorldMap::display( WorldPlayer& worldPlayer) const {
    int total = size();
    int mid = total / 2 + total % 2;

    for (int row = 0; row < mid; ++row) {
        int leftIdx = row;
```

```
        int rightIdx = row + mid;

        std::string left = formatUnitDisplay(leftIdx, worldPlayer);
        std::string right = (rightIdx < total) ? formatUnitDisplay(rightIdx,
worldPlayer) : "";

        std::cout << std::left << std::setw(40) << left << right << "\n";
    }
}


std::string WorldMap::formatUnitDisplay(int i,  WorldPlayer& worldPlayer) const
{
    std::ostringstream oss;
    const MapUnit* unit = units_[i];

    // =players=
    oss << "=";
    for (int p = 0; p < worldPlayer.getNumPlayers(); ++p) {
        const Player& player = worldPlayer.getPlayer(p);
        if (player.getStatus() != dead && player.getLocation() == i) {
            oss << std::to_string(player.getID());
        } else {
            oss << " ";
        }
    }
    oss << "= ";

    // [i]
    oss << std::setw(4) << ("[" + std::to_string(i) + "]");

    // unit name
    oss << std::setw(10) << std::right << unit->getName();

    // <owner>
    if (unit->getOwner()) {
        oss << std::setw(4) << std::right << ("{" + std::to_string(unit-
>getOwner()->getID()) + "}");
    } else {
        oss << std::setw(4) << "  ";
    }

    // type
    std::string type = " ";
    if (auto* up = dynamic_cast<const UpgradableUnit*>(unit)) {
        if (up->getOwner()) {
            // Owned: show upgraded price and level
            oss << std::setw(3) << "U$";
            oss << std::setw(5) << up->calculateFine();
            oss << " L" << up->getLevel();
        } else {
            // Not owned: show base price
            oss << std::setw(3) << "B$";
            oss << std::setw(5) << up->getPrice();
        }
    }
    else if (auto* c = dynamic_cast<const CollectableUnit*>(unit)) {
        // Collectable: <owner> x N (no fine, no level)
        if (c->getOwner()) {
            oss << " x" << c->getOwner()->getNumberOfCollectableUnits();
        } else {
            oss <<  std::setw(3) << "C$";
            oss << std::setw(5) << c->getPrice();
        }
```

```
    }
    else if (dynamic_cast<const RandomCostUnit*>(unit)) {
        // Random: just display "?" (no fine, no level)
        oss << std::setw(3) << "?";
    }
    else if (dynamic_cast<const JailUnit*>(unit)) {
        oss << std::setw(3) << "J";
    }

    return oss.str();
}


/* ------------------------------------------------------------
// main.cpp
// ------------------------------------------------------------ */
#include <iostream>
#include "WorldMap.h"
#include "player.h"
#include <thread>
#include <chrono>

using namespace std;

int main() {
    WorldMap map;
    map.loadFromFile("map.dat");

    int numPlayers = 0;
    system("clear");
    while (true) {
        cout << "How many players are there? ";
        cin >> numPlayers;

        if (cin.fail()) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Invalid input." << endl;
        } else if (numPlayers < minPlayersNum || numPlayers > maxPlayersNum) {
            cout << "Support "<<minPlayersNum<<"-"<<maxPlayersNum<<" players
only." << endl;
        } else {
            break;
        }
    }

    WorldPlayer players(numPlayers, &map);
    for (int i = 0; i < players.getNumPlayers(); i++)
    {
        cout<<players.getPlayer(i).getName()<<", do u want to change your name?
(y/n) ";
        string answer;
        cin>>answer;cin.ignore();
        if (checkAnswer(answer))
        {
            if (answer[0]=='y'||answer[0]=='Y')
            {
                cout<<"Please enter your new name: ";
                cin>>players.getPlayer(i);cin.ignore();
            }
            continue;
        }
        cout<<"Invalid input."<<endl;
        i--;
```

```
    }
    while (true)
    {
        displayScreen(map, players);

        if (players.Action1())
        {
            int unitType=players.Action2();
            ++players;
            std::this_thread::sleep_for(std::chrono::seconds(1));
        }
        else if (wantExit())
        {
            break;
        }
        if(players.gameOver())
        {
            break;
        }
    }
    return 0;
}
```

**<< 字體大小預設 12 點，字型請用 Courier New。請適當編排以利列印與閱讀，程式碼儘量不要跨行。若有需要，可以將字體大小縮為 10 點字。 >>**