***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №8**

**По дисциплине ППвИС за III семестр**

**Тема: «Потоки ввода-вывода»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Михняев А.Л.

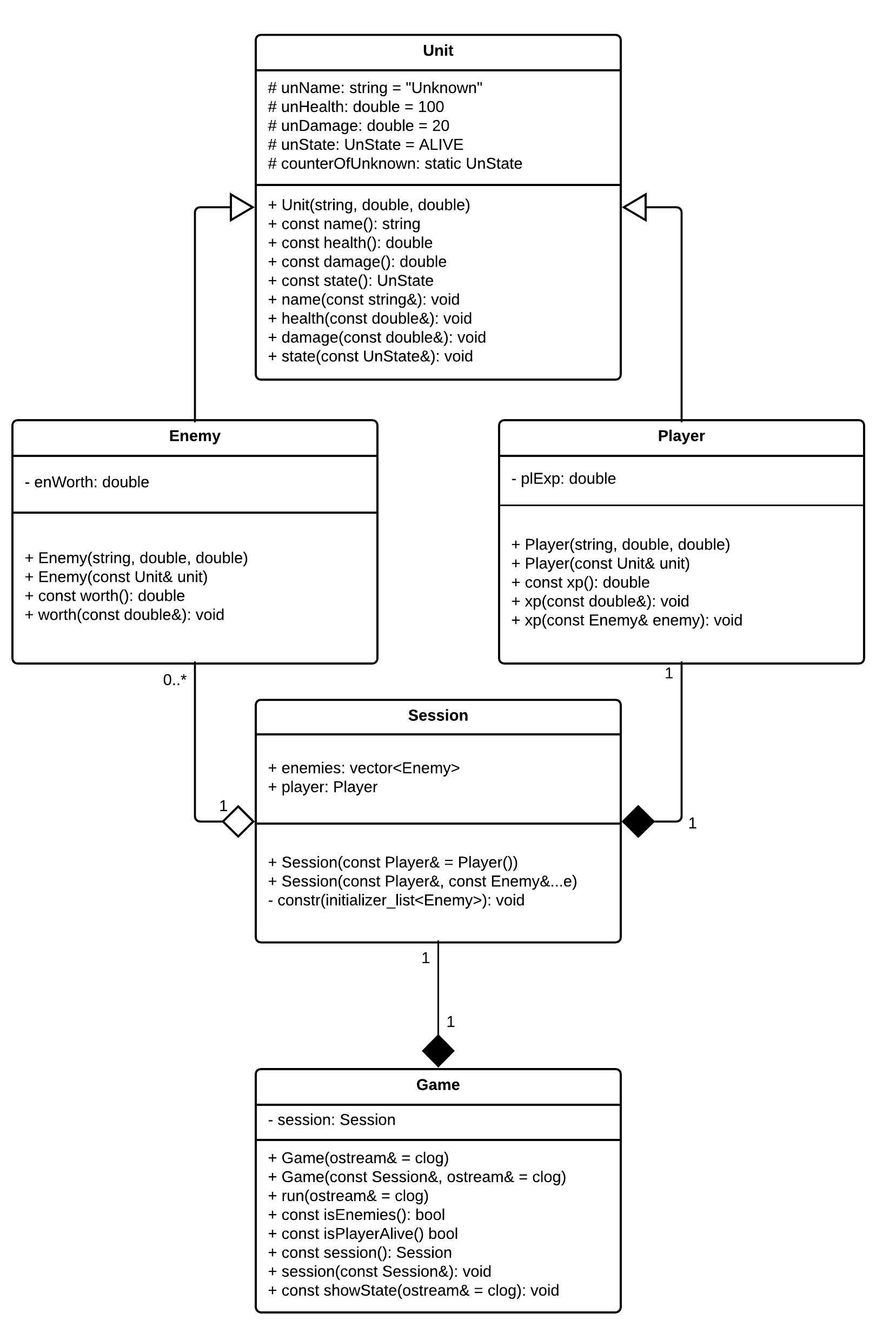
Брест 2019

ЦЕЛЬ РАБОТЫ

Знакомство с объектно-ориентированной библиотекой ввода-вывода C++ (иерархия классов iostream).

Выбранная предметная область: игра.

UML:



Листинг программы:

#include <iostream>

#include <fstream>

#include <streambuf>

#include <sstream>

#include <string>

#include <queue>

#include <chrono>

#include <ctime>

#include <iomanip>

//--------------- **Classes** ---------------//

class Unit;

class Enemy;

class Player;

struct Session;

class Game;

//--------------- **Enums** ---------------//

enum UnState

{

DEAD = 0,

ALIVE

};

//--------------- **Definitions** ---------------//

#define UnName std::string

#define UnHealth double

#define UnDamage double

#define XP double

#define DEFAULT\_NAME "Unknown"

#define DEFAULT\_HEALTH 100

#define DEFAULT\_DAMAGE 20

#define DEFAULT\_WORTH 10

#define MAX\_HEALTH 1000

#define MAX\_DAMAGE 200

#define MAX\_XP 10000

#define LOG(message, log) { \

time\_t now = time(0); \

char now\_str[1024] = "\0"; \

ctime\_s(now\_str, 1024, &now); \

log << message << " - " << now\_str; }

//--------------- **Functions** ---------------//

void saveSize(std::ofstream& stream, size\_t \* i, std::ostream& log = std::clog);

void saveUnit(std::ofstream& stream, Unit \* i, std::ostream& log = std::clog);

void saveEnemy(std::ofstream& stream, Enemy \* i, std::ostream& log = std::clog);

void savePlayer(std::ofstream& stream, Player \* i, std::ostream& log = std::clog);

void saveSession(const std::string& filename, const Session& session, std::ostream& log = std::clog);

size\_t retrieveSize(std::ifstream& stream, std::ostream& log = std::clog);

Unit retrieveUnit(std::ifstream& stream, std::ostream& log = std::clog);

Enemy retrieveEnemy(std::ifstream& stream, std::ostream& log = std::clog);

Player retrievePlayer(std::ifstream& stream, std::ostream& log = std::clog);

bool retrieveSession(const std::string& filename, Session& session, std::ostream& log = std::clog);

bool attack(const Unit& attacker, Unit& victim, std::ostream& log = std::clog);

void clearFile(const std::string& filename, std::ostream& log = std::clog);

void showLog(std::istream& log);

//--------------- **Unit** ---------------//

class Unit

{

protected:

UnName unName;

UnHealth unHealth;

UnDamage unDamage;

UnState unState;

static int counterOfUnkown;

public:

Unit(UnName initName = DEFAULT\_NAME, UnHealth initHealth = DEFAULT\_HEALTH,

UnDamage initDamage = DEFAULT\_DAMAGE)

: unName(initName), unHealth(initHealth), unDamage(initDamage), unState(ALIVE)

{

if (initName == DEFAULT\_NAME)

{

if (counterOfUnkown != 0)

{

std::stringstream addition;

addition << counterOfUnkown;

unName = DEFAULT\_NAME + addition.str();

}

counterOfUnkown++;

}

}

// to get

UnName name() const

{

return unName;

}

UnHealth health() const

{

return unHealth;

}

UnDamage damage() const

{

return unDamage;

}

UnState state() const

{

return unState;

}

// to set

void name(const UnName& newName)

{

unName = newName;

}

void health(const UnHealth& newHealth)

{

if (newHealth < 0)

unHealth = 0;

else if (newHealth > MAX\_HEALTH)

unHealth = MAX\_HEALTH;

else

unHealth = newHealth;

}

void damage(const UnDamage& newDamage)

{

if (newDamage < 0)

unDamage = 0;

else if (newDamage > MAX\_DAMAGE)

unDamage = MAX\_DAMAGE;

else

unDamage = newDamage;

}

void state(const UnState& newState)

{

unState = newState;

}

};

int Unit::counterOfUnkown = 0;

//--------------- **Enemy** ---------------//

class Enemy : public Unit

{

XP enWorth;

public:

Enemy(UnName initName = DEFAULT\_NAME, UnHealth initHealth = DEFAULT\_HEALTH,

UnDamage initDamage = DEFAULT\_DAMAGE)

: Unit(initName, initHealth, initDamage), enWorth(0) {}

Enemy(const Unit& unit)

: Unit(unit), enWorth(0) {}

XP worth() const

{

return enWorth;

}

void worth(const XP& newWorth)

{

enWorth = newWorth;

}

};

//--------------- **Player** ---------------//

class Player : public Unit

{

XP plExp;

public:

Player(UnName initName = DEFAULT\_NAME, UnHealth initHealth = DEFAULT\_HEALTH,

UnDamage initDamage = DEFAULT\_DAMAGE)

: Unit(initName, initHealth, initDamage), plExp(0) {}

Player(const Unit& unit)

: Unit(unit), plExp(0) {}

XP xp() const

{

return plExp;

}

void xp(XP newXP)

{

if (newXP < 0)

plExp = 0;

if (newXP > MAX\_XP)

plExp = MAX\_XP;

else

plExp = newXP;

}

void xp(const Enemy& enemy)

{

plExp += enemy.worth();

}

};

//--------------- **Session** ---------------//

struct Session

{

//std::queue<Enemy> enemies;

std::vector<Enemy> enemies;

Player player;

Session(const Player& player = Player())

: player(player) {}

template<typename...E>

Session(const Player& player, const E&... e)

: player(player)

{

constr({ e... });

}

private:

void constr(std::initializer\_list<Enemy> enemies)

{

for (auto enemy : enemies)

this->enemies.push\_back(enemy);

}

};

//--------------- **Game** ---------------//

class Game

{

Session curSession;

public:

Game(std::ostream& log = std::clog)

: curSession(Player(), Enemy())

{

LOG("Game have been created!", log);

}

Game(const Session& initSession, std::ostream& log = std::clog)

: curSession(initSession)

{

LOG("Game have been created!", log);

}

void run(std::ostream& log = std::clog)

{

if (curSession.enemies.empty() || curSession.player.state() == DEAD)

return;

if (attack(curSession.player, curSession.enemies.front(), log) &&

curSession.enemies.front().state() == DEAD)

{

curSession.player.xp(curSession.enemies.front());

curSession.enemies.erase(curSession.enemies.begin());

}

else

attack(curSession.enemies.front(), curSession.player, log);

}

bool isEnemies() const

{

return curSession.enemies.size();

}

bool isPlayerAlive() const

{

return curSession.player.state();

}

Session session() const

{

return curSession;

}

void session(const Session& newSession)

{

curSession = newSession;

}

void showState(std::ostream& log = std::clog) const

{

log << "-------------------------------------------" << std::endl;

log << std::setw(25) << std::left << "Player (" + curSession.player.name() + "): "

<< std::setw(15) << std::right << curSession.player.health() << "HP"

<< std::setw(15) << curSession.player.xp() << "XP" << std::endl;

if (!curSession.enemies.empty())

log << std::setw(25) << std::left

<< "Current Enemy (" + curSession.enemies.front().name() + "): "

<< std::setw(15) << std::right << curSession.enemies.front().health()

<< "HP" << std::endl;

log << "-------------------------------------------" << std::endl;

}

};

//--------------- **Functions** ---------------//

bool **attack**(const Unit& attacker, Unit& victim, std::ostream& log)

{

if (attacker.state() == DEAD || victim.state() == DEAD)

return false;

log << attacker.name() << " attack " << victim.name() << " (-";

if (victim.health() - attacker.damage() <= 0)

{

log << victim.health();

victim.health(0);

victim.state(DEAD);

}

else

{

log << attacker.damage();

victim.health(victim.health() - attacker.damage());

}

log << "HP)" << std::endl;

if (victim.state() == DEAD)

log << victim.name() << " has been killed by " << attacker.name() << std::endl;

return true;

}

void **saveSize**(std::ofstream& stream, size\_t \* i, std::ostream& log)

{

stream.write((char\*)i, sizeof(int));

}

void **saveUnit**(std::ofstream& stream, Unit \* i, std::ostream& log)

{

UnName enName\_str = i->name();

char enName[1024] = "";

for (size\_t i = 0; i < enName\_str.length(); i++)

enName[i] = enName\_str[i];

UnHealth enHealth = i->health();

UnDamage enDamage = i->damage();

UnState enState = i->state();

stream.write((char\*)&enName, sizeof(enName));

stream.write((char\*)&enHealth, sizeof(enHealth));

stream.write((char\*)&enDamage, sizeof(enDamage));

stream.write((char\*)&enState, sizeof(enState));

}

void **saveEnemy**(std::ofstream& stream, Enemy \* i, std::ostream& log)

{

saveUnit(stream, i, log);

XP enWorth = i->worth();

stream.write((char\*)&enWorth, sizeof(enWorth));

}

void **savePlayer**(std::ofstream& stream, Player \* i, std::ostream& log)

{

saveUnit(stream, i, log);

XP unExp = i->xp();

stream.write((char\*)&unExp, sizeof(unExp));

}

void **saveSession**(const std::string& filename, const Session& session, std::ostream& log)

{

std::ofstream fout(filename, std::ios::ate | std::ios::out | std::ios::binary);

fout.clear();

if (!fout)

{

LOG("Cannot open the file!", log);

return;

}

size\_t size = session.enemies.size();

saveSize(fout, &size, log);

Player player = session.player;

savePlayer(fout, &player, log);

for (Enemy en : session.enemies)

{

saveEnemy(fout, &en, log);

}

fout.close();

}

size\_t **retrieveSize**(std::ifstream& stream, std::ostream& log)

{

size\_t temp;

stream.read((char\*)&temp, sizeof(temp));

return temp;

}

Unit **retrieveUnit**(std::ifstream& stream, std::ostream& log)

{

UnName enName\_str;

char enName[1024] = "";

UnHealth enHealth;

UnDamage enDamage;

UnState enState;

stream.read((char\*)&enName, sizeof(enName));

stream.read((char\*)&enHealth, sizeof(enHealth));

stream.read((char\*)&enDamage, sizeof(enDamage));

stream.read((char\*)&enState, sizeof(enState));

enName\_str = enName;

Unit result(enName\_str, enHealth, enDamage);

result.state(enState);

return result;

}

Enemy **retrieveEnemy**(std::ifstream& stream, std::ostream& log)

{

Unit un = retrieveUnit(stream, log);

XP enWorth;

stream.read((char\*)&enWorth, sizeof(enWorth));

Enemy result(un);

result.worth(enWorth);

return result;

}

Player **retrievePlayer**(std::ifstream& stream, std::ostream& log)

{

Unit un = retrieveUnit(stream, log);

XP enExp;

stream.read((char\*)&enExp, sizeof(enExp));

Player result(un);

result.xp(enExp);

return result;

}

bool **retrieveSession**(const std::string& filename, Session& session, std::ostream& log)

{

std::ifstream fin(filename, std::ios::in | std::ios::binary);

if (!fin)

{

LOG("Cannot open the file!", log);

return false;

}

size\_t size = retrieveSize(fin, log);

Player player = retrievePlayer(fin, log);

session.player = player;

Enemy enemy;

for (size\_t i = 0; i < size; i++)

{

enemy = retrieveEnemy(fin, log);

session.enemies.push\_back(enemy);

}

fin.close();

return true;

}

void **clearFile**(const std::string& filename, std::ostream& log)

{

std::ofstream temp(filename, std::ios::out);

if (!temp)

{

LOG("Cannot open the file!", log);

return;

}

temp.close();

}

void showLog(std::istream& log)

{

log.seekg(0);

std::string str;

while (!log.eof())

{

getline(log, str);

std::cout << str << std::endl;

}

}

//----------------------------------------------

int main()

{

clearFile("res/result.txt");

clearFile("res/saved.txt");

std::fstream result("res/result.txt", std::ios::ate | std::ios::out | std::ios::in);

if (!result)

{

std::cerr << "Error" << std::endl;

return -1;

}

std::ofstream log("res/log.txt", std::ios::ate | std::ios::out);

if (!log)

{

std::cerr << "Error" << std::endl;

return -1;

}

std::streambuf \*temp = std::clog.rdbuf(log.rdbuf());

Enemy orc("Orc", 100, 20); orc.worth(10);

Enemy troll("Troll", 500, 60); troll.worth(100);

Enemy dragon("Dragon", 1000, 200); dragon.worth(10000);

Session session(Player("Player", 2000, 200), orc, troll, dragon);

saveSession("res/saved.txt", session);

Session re\_session;

retrieveSession("res/saved.txt", re\_session);

Game game(re\_session);

game.showState(result);

while (game.isEnemies() && game.isPlayerAlive())

{

game.run(result);

game.showState(result);

}

std::clog.rdbuf(temp);

showLog(result);

log.close();

result.close();

system("pause");

return 0;

}

Результат выполнения:

log.txt:

Game have been created! - Sat Jan 5 13:31:05 2019

result.txt и консоль:

-------------------------------------------

Player (Player): 2000HP 0XP

Current Enemy (Orc): 100HP

-------------------------------------------

Player attack Orc (-100HP)

Orc has been killed by Player

-------------------------------------------

Player (Player): 2000HP 10XP

Current Enemy (Troll): 500HP

-------------------------------------------

Player attack Troll (-200HP)

Troll attack Player (-60HP)

-------------------------------------------

Player (Player): 1000HP 10XP

Current Enemy (Troll): 300HP

-------------------------------------------

Player attack Troll (-200HP)

Troll attack Player (-60HP)

-------------------------------------------

Player (Player): 940HP 10XP

Current Enemy (Troll): 100HP

-------------------------------------------

Player attack Troll (-100HP)

Troll has been killed by Player

-------------------------------------------

Player (Player): 940HP 110XP

Current Enemy (Dragon): 1000HP

-------------------------------------------

Player attack Dragon (-200HP)

Dragon attack Player (-200HP)

-------------------------------------------

Player (Player): 740HP 110XP

Current Enemy (Dragon): 800HP

-------------------------------------------

Player attack Dragon (-200HP)

Dragon attack Player (-200HP)

-------------------------------------------

Player (Player): 540HP 110XP

Current Enemy (Dragon): 600HP

-------------------------------------------

Player attack Dragon (-200HP)

Dragon attack Player (-200HP)

-------------------------------------------

Player (Player): 340HP 110XP

Current Enemy (Dragon): 400HP

-------------------------------------------

Player attack Dragon (-200HP)

Dragon attack Player (-200HP)

-------------------------------------------

Player (Player): 140HP 110XP

Current Enemy (Dragon): 200HP

-------------------------------------------

Player attack Dragon (-200HP)

Dragon has been killed by Player

-------------------------------------------

Player (Player): 140HP 10110XP

-------------------------------------------

Вывод: познакомился с объектно-ориентированной библиотекой ввода-вывода С++ (iostream).