**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №3**
**по дисциплине «ООП»**
**Тема: Добавление логирования**

Студент гр. 9383 _____ Моисейченко К.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

**Цель работы.**

Добавить в программу логирование. Создать классы, отслеживающие игрока и элементы на поле, которые будут выводить информацию в файл или консоль. Изучить паттерны Наблюдатель и Мост, по возможности реализовать их.

**Задание.**

Создан набор классов, которые отслеживают игрока и элементы на поле, и выводят/сохраняют информацию об их изменениях.

Обязательные требования:

- Реализована возможность одновременной записи логов в терминал и/или файл
- Взаимодействие с файлом реализовано по идиоме RAII
- Перегружен оператор вывода в поток для всех классов, которые должны быть логированы

Дополнительные требования:

- Классы, которые отслеживают элементы, реализованы через паттерн Наблюдатель
- Разделение интерфейса и реализации класса логирования через паттерн Мост

**Выполнение работы.**

Класс Logfile:

Класс отвечает за работу с файлом и размещение в нем логов. Работа с ним осуществлена по идиоме RAII.

Поля:

- ofstream* out - указатель на поток вывода в файл

- static int count - счётчик записанных логов

Методы:

Logfile() - открывает файл для записи логов

AddLog(string) — добавление лога в файл в формате LOG[номер лога] сообщение лога

~Logfile() - закрывает файл для записи логов, удаляет указатель на поток вывода

Класс LogPlayer:

Класс реализует запись в файл логов, связанных с игроком

Поля:

- Player* player - указатель на класс игрока

Методы:

- addLog(string) - создает экземпляр класса LogFile и вызывает в нем метод AddLog от переданной строки

+ LogPlayer(Player*) - конструктор, инициализирует указатель на класс игрока

+ void GameStarts(int) - запись лога о начале игры

+ void PlayerMoves() - запись лога об изменении позиции игрока

+ void PlayerCollects(Item*, int) - запись лога о подборе элемента игроком

+ void PlayerJumps() - запись лога о том, что игрок испльзовал прыжок

+ void PlayerTakesDamage() - запись лога о получении урона игроком

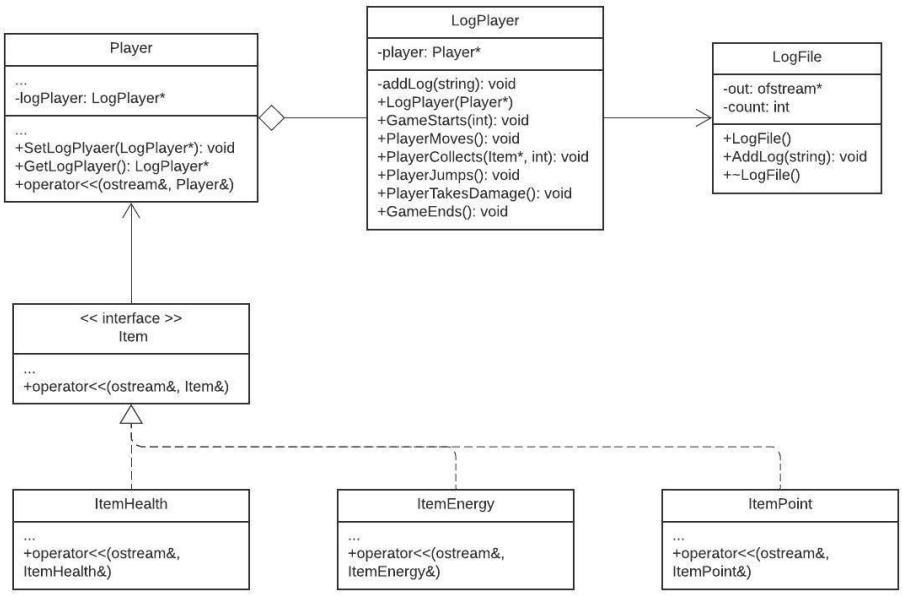
+ void GameEnds() - запись лога об окончании игры

Был перегружен оператор вывода в поток для классов Player, ItemHealth, ItemEnergy, ItemPoint

Классу игрока было добавлено поле LogPlayer* и методы SetLogPlayer(LogPlayer*) и GetLogPlayer()

Разработанный программный код см. в приложении А.

**UML-диаграмма.**




**Выводы.**

В ходе работы были добавлены классы логирования для отслеживания игрока и элементом на поле, и сохранения информации об их изменениях.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: logFile.h

```
#pragma once

#include <iostream>
#include <fstream>
#include <string>

class LogFile
{
private:
std::ofstream* out;
static int count;
public:
LogFile();
void AddLog(std::string s);
~LogFile();
};
```

Название файла: logFile.cpp

```
#include "logFile.h"

int LogFile::count = 0;

LogFile::LogFile()
{
out = new std::ofstream;
if (!count)
out->open("./logs.txt");
else
out->open("./logs.txt", std::ios::app);
}

void LogFile::AddLog(std::string s)
{
*out << "LOG[" << count++ << "]\t" << s << '\n';
}

LogFile::~LogFile()
{
out->close();
delete out;
}
```

Название файла: logPlayer.h

```
#pragma once

#include <iostream>
#include "logFile.h"
#include "../Characters/player.h"
#include "../Items/item.h"
```

```
class Player;
class Item;

class LogPlayer
{
private:
Player* player;
void addLog(std::string s);
public:
LogPlayer(Player* player);
void GameStarts(int pointsToWin);
void PlayerMoves();
void PlayerCollects(Item* item, int pointsToWin);
void PlayerJumps();
void PlayerTakesDamage();
void GameEnds();
};
```

Название файла: logPlayer.cpp

```
#include "logPlayer.h"

LogPlayer::LogPlayer(Player* player)
{
this->player = player;
this->player->SetLogPlayer(this);
}

void LogPlayer::addLog(std::string s)
{
LogFile logFile;
logFile.AddLog(s);
}

void LogPlayer::GameStarts(int pointsToWin)
{
std::string s = "Game starts\nHealth: " + std::to_string(player->GetHealth()) + "\nEnergy: " + std::to_string(player->GetEnergy()) + "\nPoints: " + std::to_string(player->GetPoints()) + "/" + std::to_string(pointsToWin) + "\n";
addLog(s);
}

void LogPlayer::PlayerMoves()
{
std::string s = "Player moves\nX position: " + std::to_string(player->GetX()) + "\tY position: " + std::to_string(player->GetY()) + "\n";
addLog(s);
}

void LogPlayer::PlayerCollects(Item* item, int pointsToWin) {
std::string s;
switch (item->GetIndex())
{
```

```
    case 0:
    s = "Player collects 1 health\nHealth: " +
std::to_string(player->GetHealth()) + "\n";
    break;
    case 1:
    s = "Player collects 1 energy\nEnergy: " +
std::to_string(player->GetEnergy()) + "\n";
    break;
    case 2:
    s = "Player collects 1 point\nPoints: " +
std::to_string(player->GetPoints()) + "/" + std::to_string(pointsToWin) +
"\n";
    break;
    default:
    break;
    }
    addLog(s);
    }

    void LogPlayer::PlayerJumps() {
    std::string s = "Player uses jump\nEnergy: " +
std::to_string(player->GetEnergy()) + "\n";
    addLog(s);
    }

    void LogPlayer::PlayerTakesDamage() {
    std::string s = "Player takes damage\nHealth: " +
std::to_string(player->GetHealth()) + "\n";
    addLog(s);
    }

    void LogPlayer::GameEnds()
    {
    std::string s = "Game ends\n";
    addLog(s);
    }
```

Название файла: player.h

```
    #pragma once
    #include "character.h"
    #include "../Logs/logPlayer.h"

    class LogPlayer;

    class Player : public Character
    {
    private:
        int health;
        int energy;
        int points;
        LogPlayer* logPlayer = nullptr;
    public:
        Player(int x, int y);
        int GetHealth();
        int GetEnergy();
        int GetPoints();
```

```
        void TakeItemHealth();
        void TakeItemEnergy();
        void TakeItemPoint();
        void TakeDamage(int x);
        void UseJump();
        void SetLogPlayer(LogPlayer* logPlayer);
        LogPlayer* GetLogPlayer();
        friend  std::ostream&  operator<<(std::ostream&  out,  Player&
player);
    };
```

Название файла: player.cpp

```
#include "player.h"

Player::Player(int x, int y)
{
this->x = x;
this->y = y;
health = 6;
energy = 3;
points = 0;
}

int Player::GetHealth()
{
return health;
}

int Player::GetEnergy()
{
return energy;
}

int Player::GetPoints()
{
return points;
}

void Player::TakeItemHealth()
{
this->health++;
}

void Player::TakeItemEnergy()
{
this->energy++;
}

void Player::TakeItemPoint()
{
this->points++;
}

void Player::TakeDamage(int x)
{
this->health -= x;
```

```
}

void Player::UseJump() {
this->energy--;
}

void Player::SetLogPlayer(LogPlayer* logPlayer)
{
this->logPlayer = logPlayer;
}

LogPlayer* Player::GetLogPlayer()
{
return logPlayer;
}

std::ostream& operator<<(std::ostream& out, Player& player)
{
out << "X position: " << player.GetX() << "\tY position: " <<
player.GetY() << "\nHealth: " << player.GetHealth() << "\nItemPoints: " <<
player.GetPoints() << "\nEnergy: " << player.GetEnergy() << "\n";
return out;
}
```

Название файла: itemHealth.h

```
#pragma once

#include "../item.h"

class ItemHealth : public Item {
public:
    void operator+(Player* player);
    int GetIndex();
    friend std::ostream& operator<<(std::ostream& out, ItemHealth&
itemHealth);
};
```

Название файла: itemHealth.cpp

```
#include "itemHealth.h"

void ItemHealth::operator+(Player* player)
{
player->TakeItemHealth();
}

int ItemHealth::GetIndex()
{
return 0;
}

std::ostream& operator<<(std::ostream& out, ItemHealth& itemHealth)
{
out << "Player collects 1 health\n";
return out;
```

```
}
```

Название файла: itemEnergy.h

```
#pragma once

#include "../item.h"

class ItemEnergy : public Item {
public:
    void operator+(Player* player);
    int GetIndex();
    friend std::ostream& operator<<(std::ostream& out, ItemEnergy&
itemEnergy);
};
```

Название файла: itemEnergy.cpp

```
#include "itemEnergy.h"

void ItemEnergy::operator+(Player* player)
{
player->TakeItemEnergy();
}

int ItemEnergy::GetIndex()
{
return 1;
}

std::ostream& operator<<(std::ostream& out, ItemEnergy& itemEnergy)
{
out << "Player collects 1 energy\n";
return out;
}
```

Название файла: itemPoint.h

```
#pragma once

#include "../item.h"

class ItemPoint : public Item {
public:
    void operator+(Player* player);
    int GetIndex();
    friend std::ostream& operator<<(std::ostream& out, ItemPoint&
itemPoint);
};
```

Название файла: itemPoint.cpp

```
#include "itemPoint.h"
```

```
void ItemPoint::operator+(Player* player)
{
player->TakeItemPoint();
}

int ItemPoint::GetIndex()
{
return 2;
}

std::ostream& operator<<(std::ostream& out, ItemPoint& itemPoint) {
out << "Player collects 1 point\n";
return out;
}
```

Название файла: gameManager.cpp

```
#include "gameManager.h"

GameManager::GameManager()
{
field = nullptr;
player = nullptr;
userInterface = new UserInterface;
}

std::pair<int, int> GameManager::randomCell()
{
int x = rand() % field->GetWidth();
int y = rand() % field->GetHeight();

while (field->GetField()[y][x].GetCellType() != CellType::PATH ||
field->IsItemSet(x, y))
{
x = rand() % field->GetWidth();
y = rand() % field->GetHeight();
}
return std::make_pair(x, y);
}

void GameManager::setGameObjects()
{
ItemFactory* itemFactory;
std::pair<int, int> randPos = randomCell();
itemFactory = new ItemPointFactory;
for (int i = 0; i < numOfItemPoint; i++)
{
randPos = randomCell();
field->GetField()[randPos.second][randPos.first].SetItem(itemFactor
y->CreateItem());
}
delete itemFactory;
itemFactory = new ItemHealthFactory;
for (int i = 0; i < numOfItemHealth; i++)
{
randPos = randomCell();
field->GetField()[randPos.second][randPos.first].SetItem(itemFactor
y->CreateItem());
```

```
	}
	delete itemFactory;
	itemFactory = new ItemEnergyFactory;
	for (int i = 0; i < numOfItemEnergy; i++)
	{
	randPos = randomCell();
	field->GetField()[randPos.second][randPos.first].SetItem(itemFactor
y->CreateItem());
	}
	delete itemFactory;
	randPos = randomCell();
	}

	void GameManager::OpenMenu()
	{
	userInterface->PrintMenu();
	std::string userCommand = userInterface->ScanCommand();
	if (userCommand == "n" || userCommand == "new")
	openNew();
	else if (userCommand == "h" || userCommand == "help")
	openHelp(0);
	else if (userCommand != "q" && userCommand != "quit")
	{
	std::cout << "Invalid command!\n";
	OpenMenu();
	}
	}

	void GameManager::openNew()
	{
	if (player)
	delete player;
	if (field)
	field->DeleteField();
	field = Field::GetInstance();
	player         =         new         Player(field->GetStart().first,
field->GetStart().second);
	player->SetLogPlayer(new LogPlayer(player));
	srand(time(0));
	numOfItemPoint = rand() % 4 + 4;
	pointsToWin = numOfItemPoint;
	numOfItemHealth = rand() % 7 + 6;
	numOfItemEnergy = rand() % 6 + 10;
	field->SetPlayer(player);
	field->GetField()[field->GetStart().second][field->GetStart().first
].PlacePlayer(player);
	setGameObjects();
	player->GetLogPlayer()->GameStarts(pointsToWin);
	parseMove();
	}

	void GameManager::parseMove()
	{
	userInterface->PrintGame(player, numOfItemPoint);
	std::string userCommand = userInterface->ScanCommand();
	if (userCommand == "p" || userCommand == "pause")
	openPause();
	else if (userCommand == "a" || userCommand == "left")
```

```
      nextMove(-1, 0);
      else if (userCommand == "d" || userCommand == "right")
      nextMove(1, 0);
      else if (userCommand == "w" || userCommand == "up")
      nextMove(0, -1);
      else if (userCommand == "s" || userCommand == "down")
      nextMove(0, 1);
      else if (userCommand == "a2" || userCommand == "left2" || userCommand
== "2a" || userCommand == "2left")
      nextMove(-2, 0);
      else if (userCommand == "d2" || userCommand == "right2" || userCommand
== "2d" || userCommand == "2right")
      nextMove(2, 0);
      else if (userCommand == "w2" || userCommand == "up2" || userCommand
== "2w" || userCommand == "2up")
      nextMove(0, -2);
      else if (userCommand == "s2" || userCommand == "down2" || userCommand
== "2s" || userCommand == "2down")
      nextMove(0, 2);
      else if (userCommand != "q" && userCommand != "quit")
      {
      std::cout << "Invalid command!\n";
      parseMove();
      }
      }

      void GameManager::openHelp(int back)
      {
      userInterface->PrintHelp();
      std::string userCommand = userInterface->ScanCommand();
      if (userCommand == "r" || userCommand == "rules")
      openRules(back);
      else if (userCommand == "c" || userCommand == "controls")
      openControls(back);
      else if (userCommand == "b" || userCommand == "back") {
      if (back == 0)
      OpenMenu();
      else
      openPause();
      }
      else if (userCommand != "q" && userCommand != "quit")
      {
      std::cout << "Invalid command!\n";
      openHelp(back);
      }
      }

      void GameManager::openRules(int back)
      {
      userInterface->PrintRules();
      std::string userCommand = userInterface->ScanCommand();
      if (userCommand == "b" || userCommand == "back")
      openHelp(back);
      else if (userCommand == "i" || userCommand == "items")
      openItems(back);
      else if (userCommand != "q" && userCommand != "quit")
      {
      std::cout << "Invalid command!\n";
```

```
openRules(back);
}
}

void GameManager::openItems(int back)
{
userInterface->PrintItems();
std::string userCommand = userInterface->ScanCommand();
if (userCommand == "b" || userCommand == "back")
openRules(back);
else if (userCommand != "q" && userCommand != "quit")
{
std::cout << "Invalid command!\n";
openItems(back);
}
}

void GameManager::openControls(int back)
{
userInterface->PrintControls();
std::string userCommand = userInterface->ScanCommand();
if (userCommand == "b" || userCommand == "back")
openHelp(back);
else if (userCommand != "q" && userCommand != "quit")
{
std::cout << "Invalid command!\n";
openControls(back);
}
}

void GameManager::openWin()
{
userInterface->PrintWin();
std::string userCommand = userInterface->ScanCommand();
if (userCommand == "m" || userCommand == "menu" || userCommand ==
"<<")
OpenMenu();
else if (userCommand == "n" || userCommand == "new" || userCommand
== ">>")
openNew();
else if (userCommand != "q" && userCommand != "quit")
{
std::cout << "Invalid command!\n";
openWin();
}
}

void GameManager::openPause()
{
userInterface->PrintPause();
std::string userCommand = userInterface->ScanCommand();
if (userCommand == "b" || userCommand == "back")
parseMove();
else if (userCommand == "h" || userCommand == "help")
openHelp(1);
else if (userCommand == "m" || userCommand == "menu")
OpenMenu();
else if (userCommand != "q" && userCommand != "quit")
```

```
      {
      std::cout << "Invalid command!\n";
      openPause();
      }
      }

      void GameManager::nextMove(int dx, int dy)
      {
      if (field->IsWall(player->GetX() + dx, player->GetY() + dy)) {
      std::cout << "That's not a valid move!\n";
      parseMove();
      return;
      }
      if (abs(dx + dy) == 2)
      {
      if (player->GetEnergy() > 0)
      {
      player->UseJump();
      player->GetLogPlayer()->PlayerJumps();
      }
      else
      {
      std::cout << "You have no energy!\n";
      parseMove();
      return;
      }
      }
      field->GetField()[player->GetY()][player->GetX()].RemovePlayer();
      player->Move(dx, dy);
      field->GetField()[player->GetY()][player->GetX()].PlacePlayer(playe
r);
      player->GetLogPlayer()->PlayerMoves();
      if (field->IsItemSet(player->GetX(), player->GetY()))
      {
      *(field->GetItem(player->GetX(), player->GetY())) + player;
      player->GetLogPlayer()->PlayerCollects(field->GetItem(player->GetX(
), player->GetY()), pointsToWin);
      switch (field->GetItem(player->GetX(), player->GetY())->GetIndex())
      {
      case 0:
      numOfItemHealth--;
      break;
      case 1:
      numOfItemEnergy--;
      break;
      case 2:
      numOfItemPoint--;
      break;
      default:
      break;
      }
      field->GetField()[player->GetY()][player->GetX()].DeleteItem();
      }
      if  (field->GetField()[player->GetY()][player->GetX()].GetCellType()
== CellType::END)
      {
      if (player->GetPoints() == pointsToWin)
      {
```

```
player->GetLogPlayer()->GameEnds();
openWin();
return;
}
else
{
std::cout << "You haven't collected all the points!\n";
parseMove();
return;
}
}
parseMove();
}

GameManager::~GameManager()
{
if (player)
delete player;
if (field)
field->DeleteField();
if (userInterface)
delete userInterface;
}
```