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

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Объектно-ориентированное программирование»

Тема: Сохранение и загрузка / Написание исключений

Студент гр. 9383 Поплавский И

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Изучить метод сохранения программы и реализовать по паттерну Снимок, а также защитить файл сохранения от внешнего изменения.

Задача

Создать классы, которые позволяют сохранить игру, а потом загрузить ее. Также, написать набор исключений, которые как минимум позволяют контролировать процесс сохранения и загрузки

Обязательные требования:

- Игру можно сохранить в файл
- Игру можно загрузить из файла
- Взаимодействие с файлами по идиоме RAII
- Добавлена проверка файлов на корректность
- Написаны исключения, которые обеспечивают транзакционность

Дополнительные требования:

- Для получения состояния программы используется паттерн **Снимок**

Архитектурные Решения

Интерфейс Memento и класс GameMemento - класс состояния, который получает и хранит его.

Класс Caretaker - класс отвечающий за сохранение и восстановление данных.

Само сохранение и восстановление прописано в GameHandler.

Для защиты данных от внешнего воздействия были использованы хеш функции. Сначала берется хеш функция от состояния игры и добавляется в начало файла сохранений. При восстановлении хеши проверяются и если не совпадают, то вызывается исключение.

Выводы.

Был изучен метод сохранения программы и реализован по паттерну Снимок, а также реализована защита файла сохранения от внешнего изменения при помощи хеш функции.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: Memento.h

```
#pragma once
#include <string>
#include <ctime>

class Memento {
public:
    virtual std::string state() const = 0;
};

class GameMemento : public Memento {
private:
    std::string state_;

public:
    GameMemento(std::string state) : state_(state) {
        this->state_ = state;
    }

    std::string state() const override {
        return this->state_;
    }
};
```

Файл: snapshot.h

```
#pragma once
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include "Memento.h"
#include "GameHandler.h"

class Caretaker {
private:
    std::vector<Memento*> mementos_;
    GameHandler* game_;

public:
    Caretaker(GameHandler* game) : game_(game) {
        this->game_ = game;
    }

    void Backup() {
        this->mementos_.push_back(this->game_->Save());
    }
    bool Undo() {
        if (!this->mementos_.size()) {
            std::ifstream fin("save.txt");
            if (!fin.is_open())
            {
                std::cout << "Save in data and file save.txt not found";
                return false;
```

```
            }
            std::string total;
            std::string s;
            while (getline(fin, s)) {
               total.append(s);
            }
            fin.close();

            Memento* memento = new GameMemento(total);
            this->game_->Restore(memento);
            return true;
        }
        Memento* memento = this->mementos_.back();
        this->mementos_.pop_back();
        try {
            this->game_->Restore(memento);
        }
        catch (...) {
            std::cout << "Restoring is failed" << std::endl;
            return false;
        }
    }
};
```

Изменения в GameHandler

```
Memento* GameHandler::Save()
{
    using namespace std;
    ofstream fout;
    fout.open("save.txt");
```

```
state_ = "";
// Saving Area
int heightArea = AreaHandler::getHeight();
int widthArea = AreaHandler::getWidth();
state_.append(to_string(heightArea));
state_.append(" ");
state_.append(to_string(widthArea));
state_.append(" ");
for (int i = 0; i < heightArea; i++)
{
   for (int j = 0; j < widthArea; j++)
   {
      CellType typeCell = AreaHandler::getCellType(j, i);
      ItemType typeItem = AreaHandler::getCellItem(j, i);
      state_.append(to_string(int(typeCell)));
      state_.append(" ");
      state_.append(to_string(int(typeItem)));
      state_.append(" ");
   }
}

// Saving Person
state_.append(to_string(person->getX()));
state_.append(" ");
state_.append(to_string(person->getY()));
state_.append(" ");
state_.append(to_string(person->getCounterCoin()));
state_.append(" ");
```

```
    state_.append(to_string(person->getLives()));
    state_.append(" ");

    // Saving Enemy
    state_.append(to_string(enemy1->getX()));
    state_.append(" ");
    state_.append(to_string(enemy1->getY()));
    state_.append(" ");
    state_.append(to_string(enemy2->getX()));
    state_.append(" ");
    state_.append(to_string(enemy2->getY()));
    state_.append(" ");
    state_.append(to_string(enemy3->getX()));
    state_.append(" ");
    state_.append(to_string(enemy3->getY()));

    //hash string
    hash<std::string> str_hash;
    string hashed = std::to_string(str_hash(state_)) + " ";
    state_.insert(0, hashed);
    //cout << state_ << endl << endl;

    fout << state_;
    fout.close();
    return new GameMemento(this->state_);
}

void GameHandler::Restore(Memento* momento)
{
```

```
using namespace std;
state_ = momento->state();
// Hash check
hash<std::string> str_hash;
string hash1 = state_.substr(0, state_.find(' '));
   string hash2 = to_string(str_hash(state_.substr(hash1.length() + 1,
state_.size() - hash1.length())));
if (hash1.length() == hash2.length())
{
   for (auto i = 0; i < hash1.length(); i++)
   {
      if (hash1[i] != hash2[i])
      {
         cout << "The data is corrupted. Restore save failed" << endl;
         return;
      }
   }
}
else
{
   cout << "The data is corrupted. Restore save failed" << endl;
   return;
}

// Move data to vector
state_ = state_.substr(hash1.length() + 1, state_.size() - hash1.length());
vector<int> vect_data;
char* s = new char[state_.size() + 1];
strcpy(s, state_.c_str());
```

```
char* p = strtok(s, " ");
while (p!= NULL) {
    vect_data.push_back(atoi(p));
    p = strtok(NULL, " ");
}

// Recovery data
int ind = 0;
int heightArea = vect_data[ind++];
int widthArea = vect_data[ind++];
Area::getInstance();
for (int i = 0; i < heightArea; i++)
{
    for (int j = 0; j < widthArea; j++)
    {
        CellType type = (CellType)vect_data[ind++];
        ItemType item = (ItemType)vect_data[ind++];
        AreaHandler::setCell(j, i, type);
        if (type == CellType::OBJECT)
        {
            AreaHandler::setItem(j, i, item);
        }
    }
}

person->Instance();
person->setX(vect_data[ind++]);
person->setY(vect_data[ind++]);
person->setCoin(vect_data[ind++]);
```

```
    person->setLives(vect_data[ind++]);

    enemy1->setX(vect_data[ind++]);
    enemy1->setY(vect_data[ind++]);
    enemy2->setX(vect_data[ind++]);
    enemy2->setY(vect_data[ind++]);
    enemy3->setX(vect_data[ind++]);
    enemy3->setY(vect_data[ind++]);

}
```