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: сериализация, исключения

Студент гр. 0383: Бояркин Н.А.

Преподаватель: Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Изучить механизм работы сериализации и исключений.

Задание.

Сериализация - это сохранение в определенном виде состоянии программы с возможностью последующего его восстановления даже после закрытия программы. В рамках игры, это сохранения и загрузка игры.

Требования:

- Реализовать сохранения всех необходимых состояний игры в файл
- Реализовать загрузку файла сохранения и восстановления состояния игры
- Должны быть возможность сохранить и загрузить игру в любой момент
- При запуске игры должна быть возможность загрузить нужный файл
- Написать набор исключений, который срабатывают если файл с сохранением некорректный
- Исключения должны сохранять транзакционность. Если не удалось сделать загрузку, то программа должна находится в том состоянии, которое было до загрузки. То есть, состояние игры не должно загружаться частично

Потенциальные паттерны проектирования, которые можно использовать:

- Снимок (Memento) - получение и восстановления состояния объектов при сохранении и загрузке

Выполнение работы:

1) Создан класс создателя (*class Originator*), объекты которого должны создавать снимки своего состояния. Класс создателя имеет единственное поле *Field* state_*, которое мы получаем с помощью функции *Field* getField()* и передаем в класс игры. Также создан метод получения снимков (*Memento* save(std::string filename, bool flag = true)*). Создатель создаёт новые объекты снимков, передавая значения своих полей через конструктор. Параметр *flag* используется для того, чтобы определить нужно ли сохранять файл или нет. Соответственно, с помощью этого параметра мы можем при запуске игры запустить нужный файл. Т.е. мы его кладем в стек истории снимков, а далее восстанавливаем. Метод восстановления из снимка был реализован в функции *void restore(Memento* memento)*.

2) Создали класс снимка (*class Memento*) с тем же полем, что и в создателе, также добавили поле с названием файла. Функция сохранения файл реализована в *void save_file()*, а восстановления в *void load_file()*.

3) Создан класс опекун (*class Caretaker*), с помощью которого мы сохраняем (функция *void Backup(std::string filename, bool flag = true)*) и восстанавливаем (функция *void Undo()*) поле. Также создан вектор *histrory_,* который хранит историю всех снимков в виде стека, т.е. у нас появляется возможность откатиться назад.

4) Также в функции Undo() прописан набор исключений, которые срабатывают если файл с сохранением некорректный. Помимо этого,

если не удалось сделать загрузку, то программа должна находится в том состоянии, которое было до загрузки, т.е. мы просто ничего не возвращаем.

Разработанный программный код см. в приложении А.

Диаграмму с зависимостями классов см. в приложении Б.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Тест	Результат	Комментарии
1.	При запуске игры загружается нужный файл		Верно
2.	Если мы пытаемся загрузить некорректный файл, то игра находится в том же состоянии, что была до этого.		Верно
3.	Есть возможность в любой момент времени сохранить		Верно

	игру и восстановится обратно.		

Выводы.

В ходе работы были изучен механизм работы сериализации и исключений.

Приложение А.

Исходный код программы.

Название файла: Caretaker.h

```
#pragma once
#include <utility>

#include "Memento.h"
#include "Originator.h"

class Caretaker{
private:
    Originator *originator_;
    std::vector<Memento*> history_;
public:
    Caretaker(Originator *originator);
    void Backup(std::string filename, bool flag = true);
    void Undo();
};
```

Название файла: Caretaker.cpp

```
#include "Caretaker.h"

Caretaker::Caretaker(Originator *originator) {
    this->originator_ = originator;
}

void Caretaker::Backup(std::string filename, bool flag) {
//    if (flag){
//
this->history_.push_back(this->originator_->save(std::move(filenam
e)));
//    } else{
//
this->history_.push_back(this->originator_->save(std::move(filenam
e), flag));
//    }
    if (!flag){
        this->history_.clear();
    }

this->history_.push_back(this->originator_->save(std::move(filenam
e), flag));
}

void Caretaker::Undo() {
    if (this->history_.empty()) {
```

```
            return;
        }
        Memento *memento = this->history_.back();
        this->history_.pop_back();
        try {
            this->originator_->restore(memento);
        } catch (...) {
            //this->Undo();
            return;
        }
    }
```

Название файла: Memento.h

```
    #pragma once
    #include <utility>
    #include "../Field.h"
    #include <fstream>
    //#include "../Game.h"

    class Memento{
    private:
        Field* state_;
        std::string filename_;
    public:
         Memento(Field* state, std::string filename, bool flag =
true);
        Field* getState();
        void save_file();
        void load_file();
    };
```

Название файла: Memento.cpp

```
    #include "Memento.h"

    // РЕАЛИЗОВАНО СОХРАНЕНИЕ В ФАЙЛ
    Memento::Memento(Field*  state,  std::string  filename,  bool
flag) {
        this->state_ = state;
        this->filename_ = std::move(filename);
        if (flag){
            save_file();
        }
    //    save_file();
    }
    // ВОССТАНОВЛЕНИЕ ИЗ ФАЙЛА
```

```
Field* Memento::getState() {
    // std::cout << "filename: " << filename_ << std::endl;
    load_file();
    return this->state_;
}

void Memento::save_file() {
    std::ofstream file;
    file.open(filename_, std::ofstream::trunc);
    file << "height " << state_->getHeight() << std::endl;
    file << "width " << state_->getHeight() << std::endl;
      file << "CountEnemy " << state_->getCountEnemies() <<
std::endl;
       //file << "CountEnemy " << state_->getCountEnemies() <<
std::endl;
    file << std::endl;
    for (int i = 0; i < state_->getHeight(); ++i) {
        for (int j = 0; j < state_->getWidth(); ++j) {
             // file << "Cage " << state_->board[i][j].getX()
<< " " << state_->board[i][j].getY() << std::endl;
                                        file << "TypeObj " <<
state_->board[i][j].getTypeObj() << std::endl;
                                       file << "TypeCage " <<
state_->board[i][j].getTypeCage() << std::endl;
                                                            if
((dynamic_cast<MainCharacter*>(state_->board[i][j].getEntity()))){
                    file << "MainHero" << std::endl;
                }
                                                        else if
((dynamic_cast<Zombie*>(state_->board[i][j].getEntity()))){
                    file << "Zombie" << std::endl;
                }
                                                        else if
((dynamic_cast<Ghost*>(state_->board[i][j].getEntity()))){
                    file << "Ghost" << std::endl;
                }
                                                        else if
((dynamic_cast<Monster*>(state_->board[i][j].getEntity()))){
                    file << "Monster" << std::endl;
                }
                                                        else if
((dynamic_cast<Armor*>(state_->board[i][j].getEntity()))){
                    file << "Armor" << std::endl;
                }
                                                        else if
((dynamic_cast<Health*>(state_->board[i][j].getEntity()))){
                    file << "Health" << std::endl;
                }
                                                        else if
((dynamic_cast<Weapon*>(state_->board[i][j].getEntity()))){
                    file << "Weapon" << std::endl;
```

```
                } else{
                    file << "nothing" << std::endl;
                };
            }
        }
        file << "MainHeroStats" << std::endl;
                                                    file           <<
(dynamic_cast<MainCharacter*>(state_->hero->getEntity()))->getHeal
th() << std::endl;
                                                    file           <<
(dynamic_cast<MainCharacter*>(state_->hero->getEntity()))->getPowe
r() << std::endl;
                                                    file           <<
(dynamic_cast<MainCharacter*>(state_->hero->getEntity()))->getArmo
r() << std::endl;
        for (int i = 0; i < state_->getCountEnemies(); ++i) {
                                                                   if
((dynamic_cast<Enemies*>(state_->arr_enemies[i]->getEntity()))) {
                file << "Enemy" << '\n';
            }
    //                                                             if
((dynamic_cast<Zombie*>(state_->arr_enemies[i]->getEntity()))) {
    //             file << "Zombie" << '\n';
    //         }
    //                                                             if
((dynamic_cast<Ghost*>(state_->arr_enemies[i]->getEntity()))) {
    //             file << "Ghost" << '\n';
    //         }
    //                                                             if
((dynamic_cast<Monster*>(state_->arr_enemies[i]->getEntity()))) {
    //             file << "Monster" << '\n';
    //         }
             file << "X " << state_->arr_enemies[i]->getX() <<
'\n';
             file << "Y " << state_->arr_enemies[i]->getY() <<
'\n';
                                file   <<   "EnemyHealth   "   <<
(dynamic_cast<Enemies&>(*state_->arr_enemies[i]->getEntity())).get
Health() << std::endl;
                                 file   <<   "EnemyPower   "   <<
(dynamic_cast<Enemies&>(*state_->arr_enemies[i]->getEntity())).get
Damage() << std::endl;
        }
        file.close();
    }

    void Memento::load_file() {
        std::ifstream file;
        file.open(filename_);
        std::string nothing;
        std::string name_enemy;
```

```
            int h, w, typeObj, typeCage, HealthHero, ArmorHero,
PowerHero, HealthEnemy, PowerEnemy, x, y, countEnemy;
          int x_new_hero , y_new_hero;
          file >> nothing;
          file >> h;
          file >> nothing;
          file >> w;
          file >> nothing;
          file >> countEnemy;
          state_ = new Field(w, h);
          for (int i = 0; i < state_->getCountEnemies(); ++i) {
              state_->arr_enemies[i] = nullptr;
          }

          MainCharacter* newMainCharacter = new MainCharacter();
          Zombie* newZombie = new Zombie();
          Ghost* newGhost = new Ghost();
          Monster* newMonster = new Monster();
          Health* newHealth = new Health();
          Armor* newArmor = new Armor();
          Weapon* newWeapon = new Weapon();
          for (int i = 0; i < h; ++i) {
              for (int j = 0; j < w; ++j) {
                  state_->board[i][j].setEntity(nullptr);
                  file >> nothing;
                  file >> typeObj;
                  switch (typeObj) {
                      case 0:

state_->board[i][j].setTypeObj(TypeObj::NOTHING);
                          break;
                      case 1:

state_->board[i][j].setTypeObj(TypeObj::PLAYER);
                          break;
                      case 2:

state_->board[i][j].setTypeObj(TypeObj::ENEMY);
                          break;
                      case 3:

state_->board[i][j].setTypeObj(TypeObj::INTERACTION_ELEMENTS);
                          break;
                  }
                  file >> nothing;
                  file >> typeCage;
                  switch (typeCage) {
                      case 0:

state_->board[i][j].setTypeCage(TypeCage::START);
                          break;
```

```
                    case 1:
state_->board[i][j].setTypeCage(TypeCage::END);
                        break;
                    case 2:
state_->board[i][j].setTypeCage(TypeCage::PASSABLE);
                        break;
                    case 3:
state_->board[i][j].setTypeCage(TypeCage::IMPASSABLE);
                        break;
                    case 4:
state_->board[i][j].setTypeCage(TypeCage::EMPTY);
                        break;
                }
                file >> nothing;
                if (nothing == "MainHero"){
                    x_new_hero = i;
                    y_new_hero = j;
state_->board[i][j].setEntity(newMainCharacter);
                }
                else if (nothing == "Zombie"){
                    state_->board[i][j].setEntity(newZombie);
                                    //  state_->arr_enemies[i]  =
&state_->board[i][j];
                }
                else if (nothing == "Ghost"){
                    state_->board[i][j].setEntity(newGhost);
                                    //  state_->arr_enemies[i]  =
&state_->board[i][j];
                }
                else if (nothing == "Monster"){
                    state_->board[i][j].setEntity(newMonster);
                                    //  state_->arr_enemies[i]  =
&state_->board[i][j];
                }
                if (nothing == "Health"){
                    state_->board[i][j].setEntity(newHealth);
                                    //  state_->arr_enemies[i]  =
&state_->board[i][j];
                }
                else if (nothing == "Armor"){
                    state_->board[i][j].setEntity(newArmor);
                                    //  state_->arr_enemies[i]  =
&state_->board[i][j];
                }
                else if (nothing == "Weapon"){
                    state_->board[i][j].setEntity(newWeapon);
```

```
                                        //  state_->arr_enemies[i]  =
&state_->board[i][j];
                }
            }
        }
        file >> nothing;
        file >> HealthHero;
        file >> PowerHero;
        file >> ArmorHero;
        newMainCharacter->setHealth(HealthHero);
        newMainCharacter->setPower(PowerHero);
        newMainCharacter->setArmor(ArmorHero);
        state_->hero = &state_->board[x_new_hero][y_new_hero];
        for (int i = 0; i < countEnemy; ++i) {
            file >> nothing;
            if (nothing == "Enemy"){
                file >> nothing;
                file >> x;
                file >> nothing;
                file >> y;
                state_->arr_enemies[i] = &state_->board[x][y];
            }
            file >> nothing;
            file >> HealthEnemy;
            file >> nothing;
            file >> PowerEnemy;

dynamic_cast<Enemies*>(state_->board[x][y].getEntity())->setHealth
(HealthEnemy);

dynamic_cast<Enemies*>(state_->board[x][y].getEntity())->setDamage
(PowerEnemy);
        }
    //    while (!file.eof()){
    //
    //    }
        file.close();
    }
```

Название файла: Originator.h

```
#pragma once
#include <utility>
#include "Memento.h"


class Originator{
private:
    Field* state_;
```

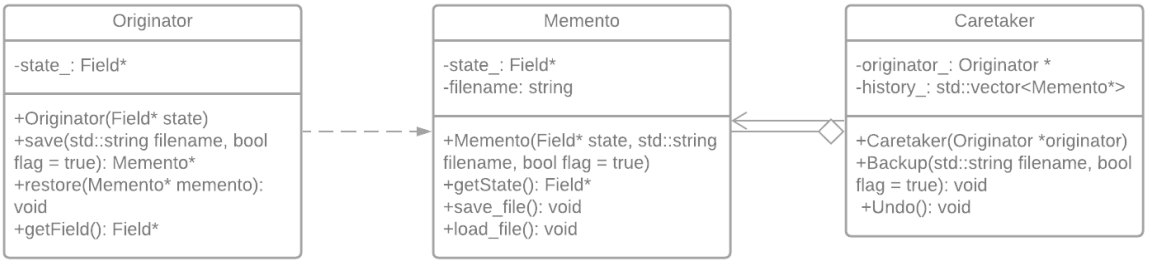
```
    public:
        Originator(Field* state);
        Memento* save(std::string filename, bool flag = true);
        void restore(Memento* memento);
        Field* getField();
    };
```

Название файла: Originator.cpp

```
    #include "Originator.h"

    Originator::Originator(Field* state) {
        this->state_ = state;
    }

    Memento *Originator::save(std::string filename, bool flag) {
           return  new  Memento(this->state_,  std::move(filename),
flag);
    }

    void Originator::restore(Memento *memento) {
        this->state_ = memento->getState();
    }

    Field *Originator::getField() {
        return this->state_;
    }
```

Приложение Б

UML-диаграмма классов



Link