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. 0304 Крицын Д.Р.
Преподаватель Жангиров Т.Р.

Санкт-Петербург
2021

Цель работы.

Имплементировать механизм сохранения и загрузки состояния игры в поток ввода/вывода (файл).

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.

Выполнение работы.

1. Создание класса *savestream* (поток сохранений) — наследника класса стандартной библиотеки *std::fstream*, который предоставляет некоторые вспомогательные функции для сериализации в двоичном

формате: *writeBytes(void*, size_t) / readBytes(void*, size_t)* и *writeString(std::string) / std::string readString()*. Первая пара функций считывает или записывать произвольное количество байт в поток, вторая пара функций — строки, терминированные символом переноса строки.

2. Создание класса *serializable*, который представляет собой некий интерфейс для сериализуемых объектов (однако в отличие от чистого интерфейса, методы в нём изначально определены): метод *std::string getClassName()* возвращает имя класса (что используется в классе *class_creator*), метод *void saveToFile(savestream& svs)*, который сохраняет объект в поток сохранений, и метод *void loadFromFile(savestream& svs)*, который конструирует объект из потока сохранений. При этом очевидно, что у каждого сериализуемого объекта должен быть определён конструктор по умолчанию, либо все аргументы по умолчанию хотя бы для одного из конструкторов, т.к. загрузка информации о сериализуемом объекте из файла осуществляется посредством вызова *void loadFromFile(savestream& svs)* и ничем более. Кроме того, при наследовании сериализуемых классов в методах *saveToFile() / loadFromFile()* в первую очередь должен вызываться этот же метод, но у родительского класса, для того, чтобы обеспечить запись в первую очередь состояния родителя, а потом уже новые, присущие дочернему классу, части состояния. Пример:

```
void actor::saveToFile(savestream& svs)
{
    cell_object::saveToFile(svs);
    svs.writeBytes(&health, sizeof(health));
```

```
    svs.writeBytes(&max_health, sizeof(max_health));
    svs.writeBytes(&dmg_thres, sizeof(dmg_thres));
    svs.writeBytes(&dmg_res, sizeof(dmg_res));
    svs.writeBytes(&team_id, sizeof(team_id));
}
```

3. Создание класса *class_creator*, который отвечает за:

а) Управление ассоциативным массивом, который связывает имя класса и функцию, возвращающую прототип объекта этого класса (созданный через конструктор по умолчанию) (*typedef std::function<void*()> default_constructor_func*). Так как это добавление происходит статически (до вызова *main()*), то добавление элементов в ассоциативный массив на этом этапе затруднительно; Поэтому для начала используются статические поля *static std::string* preinit_str* и *static default_constructor_func** *preinit_func*, которые представляют собой параллельные массивы для имён классов и функций, возвращающих их прототипы, которые затем при вызове в *main()* метода *static void init()* этого класса записываются наконец в ассоциативный массив *static std::map<std::string, default_constructor_func*> def_constructors*. Статическое добавление происходит посредством вызова метода *static void addDefConstructor(std::string class_name, default_constructor_func* cn)* через класс *class_creator_inserter*, описанный ниже.

б) Считывание из потока объектов полиморфного типа. Класс *serializable* в своих методах загрузки и сохранения в поток записывает и считывает имя класса; Этот факт в совокупности с ассоциативным массивом имён классов и функций-«конструкторов по умолчанию» используется для реализации метода *static void**

loadObject(savestream& svs), который считывает из потока сохранений и создаёт объект какого-то полиморфного класса.

4. Создание класса *class_creator_inserter*, который в своём конструкторе осуществляет вставку класса и его конструктора по умолчанию в *class_creator* — посредством вызова метода *addDefConstructor*. Это позволяет использовать его статически. Пример:

```
static default_constructor_func dcf = [](){ return new actor(); };
static class_creator_inserter cci("actor", &dcf);
```

5. Создание двух классов исключений для обработки исключительных ситуаций при считывании сохранения:

а) *save_error_unexpected_eof* представляет собой исключение неожиданного конца файла (EOF), когда ожидались какие-то данные. Данное исключение хранит в себе (и выводит при вызове метода *what()* размер операции ввода/вывода *size_t op_sz*.

б) *save_error_unknown_class_name* представляет собой исключение неизвестного имени класса, в случае чтения такого имени из сохранения и попытки создания его объекта, но неспособности класса *class_creator* найти имя такого класса в ассоциативном массиве имён классов и конструкторов по умолчанию.

6. Модифицирование классов, подлежащих сериализации, что в последствии создаёт некоторую иерархию, требущую для полной сериализации игры сериализации только основного поля *field*: *field, cell, cell_exit, cell_entrance, cell_object, actor, enemy, item, player, acolyte, sentinel, stalkerbot, armor_item, health_item, weapon_item,*

action, action_combat, action_move, action_shoot_ballistic, armor_item_wear, health_item_heal, weapon_item_fire.

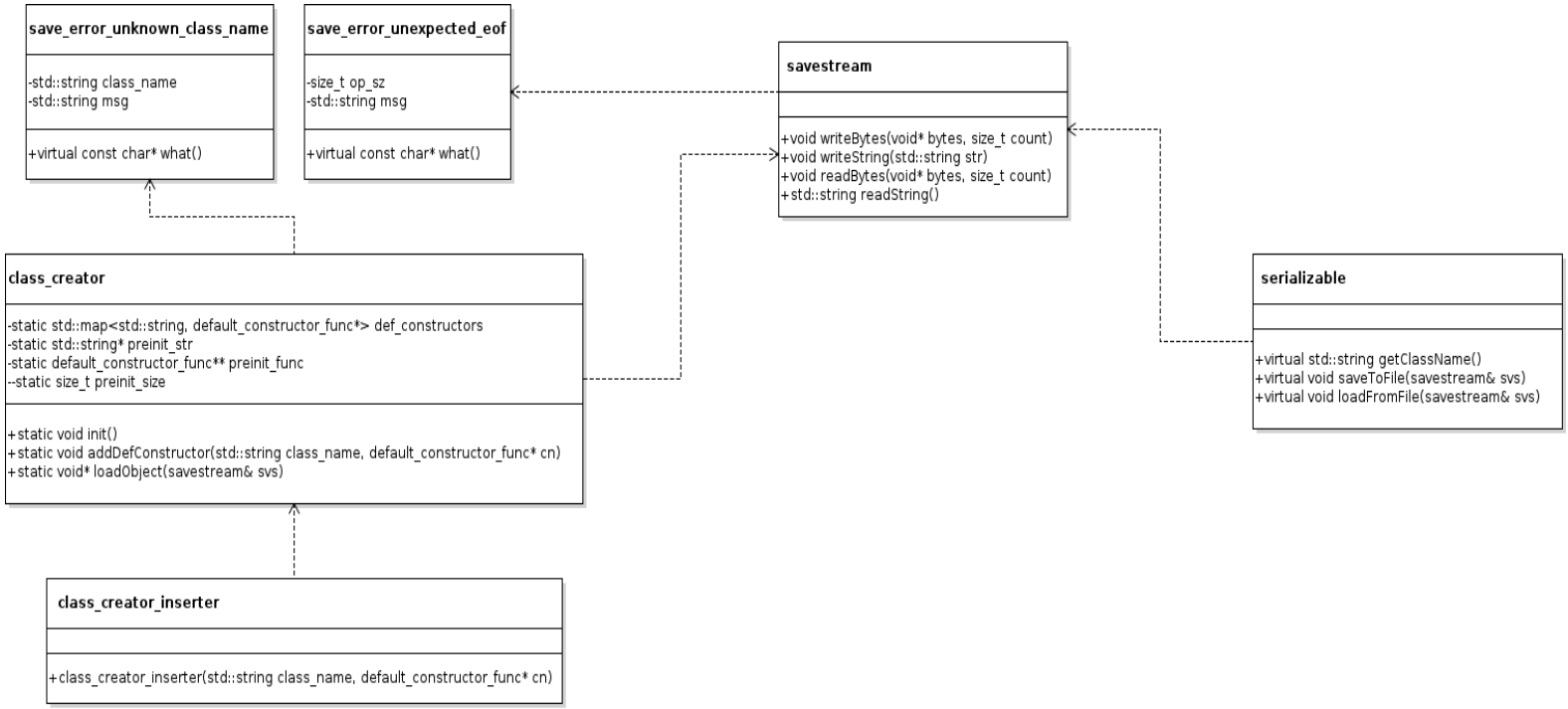
7. Создание методов сохранения и загрузки игры *bool load(std::string)* и *bool save(std::string)* в классе *game*, возвращающих булевое значение, сигнализирующее об успешности загрузки или сохранения игры (для соответствующей перерисовки интерфейса). При этом именно в этих методах происходит отлавливание исключений работы с сохранениями.

8. Создание необходимого интерфейса для сохранения-загрузки через горячие клавиши в классе-меню *ui_menu_player_actions*.

Выводы.

В ходе выполнения данной лабораторной работы было изучено понятие сериализации, и также понятие состояния объекта, и различные способы сохранения и восстановления такого состояния. Это было реализовано на практике при помощи интерфейса сериализуемого объекта и множества вспомогательных классов.

ПРИЛОЖЕНИЕ А
UML-ДИАГРАММА КЛАССОВ



ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

class_creator.cpp

```
#include "class_creator.h"

#include "serializable.h"

#include <cstring>
#include "save_exceptions.h"

std::map<std::string, default_constructor_func*>
   class_creator::def_constructors;
std::string* class_creator::preinit_str = nullptr;
default_constructor_func** class_creator::preinit_func =
   nullptr;
size_t class_creator::preinit_size = 0;


void class_creator::init()
{
 for(size_t i = 0; i < preinit_size; ++i)
 {
    def_constructors[preinit_str[i]] = preinit_func[i];
 }
 std::free(preinit_str);
 std::free(preinit_func);
}

void class_creator::addDefConstructor(std::string
   class_name, default_constructor_func* cn)
{
 preinit_size++;
 preinit_str = (std::string*)std::realloc(preinit_str,
   preinit_size * sizeof(std::string));
 preinit_func =
    (default_constructor_func**)std::realloc(preinit_func,
   preinit_size * sizeof(void*));

 preinit_str[preinit_size-1] = class_name;
 preinit_func[preinit_size-1] = cn;
}
void* class_creator::loadObject(savestream& svs)
{
 std::string class_name = svs.readString();
 int backw = std::strlen(class_name.c_str()) + 1;
```

```
 svs.seekg(-backw, std::ios_base::cur);

 auto defcn = def_constructors.find(class_name);
 if(defcn == def_constructors.end())
    throw save_error_unknown_class_name(class_name);
 serializable* obj = (serializable*)(*defcn->second)();
 obj->loadFromFile(svs);
 return obj;
}
```

class_creator.h

```
#ifndef CLASS_CREATOR_H
#define CLASS_CREATOR_H

#include <functional>
#include <map>

#include "savestream.h"

typedef std::function<void*()> default_constructor_func;

class class_creator
{
 private:

 static std::map<std::string, default_constructor_func*>
    def_constructors;
 static std::string* preinit_str;
 static default_constructor_func** preinit_func;
 static size_t preinit_size;


 public:

 static void init();

 static void addDefConstructor(std::string class_name,
    default_constructor_func* cn);
 static void* loadObject(savestream& svs);
};

#endif
```

class_creator_inserter.cpp

```
#include "class_creator_inserter.h"
```

```
class_creator_inserter::class_creator_inserter(std::string
   class_name, default_constructor_func* cn)
{
 class_creator::addDefConstructor(class_name, cn);
}
```

class_creator_inserter.h

```
#ifndef CLASS_CREATOR_INSERTER_H
#define CLASS_CREATOR_INSERTER_H

#include "class_creator.h"

class class_creator_inserter
{
 public:

 class_creator_inserter(std::string class_name,
   default_constructor_func* cn);
};

#endif
```

save_exceptions.cpp

```
#include "save_exceptions.h"

#include <sstream>

save_error_unexpected_eof::save_error_unexpected_eof(size_t
   op_sz) : op_sz(op_sz)
{
 std::stringstream ss;
 ss << "Unexpected EOF on operation of size " << op_sz;
 msg = ss.str();
}

const char* save_error_unexpected_eof::what()
{
 return msg.c_str();
}


save_error_unknown_class_name::save_error_unknown_class_name
   (std::string class_name) : class_name(class_name)
{
```

```
 std::stringstream ss;
 ss << "Unknown class name " << class_name;
 msg = ss.str();
}

const char* save_error_unknown_class_name::what()
{
 return msg.c_str();
}
```

save_exceptions.h

```
#ifndef SAVE_EXCEPTIONS_H
#define SAVE_EXCEPTIONS_H

#include <exception>
#include <string>

class save_error_unexpected_eof : public std::exception
{
 private:

 size_t op_sz;
 std::string msg;


 public:

 save_error_unexpected_eof(size_t op_sz);

 virtual const char* what();
};

class save_error_unknown_class_name : public std::exception
{
 private:

 std::string class_name;
 std::string msg;


 public:

 save_error_unknown_class_name(std::string class_name);

 virtual const char* what();
```

```
};

#endif
```

savestream.cpp

```
#include "savestream.h"

#include "save_exceptions.h"

#include <cstring>
#include <sstream>

void savestream::writeBytes(void* bytes, size_t count)
{
 write((const char*)bytes, count);
 if(!good())
    throw save_error_unexpected_eof(count);
}
void savestream::writeString(std::string str)
{
 write(str.c_str(), std::strlen(str.c_str()));
 put('\n');
 if(!good())
    throw save_error_unexpected_eof(1);
}

void savestream::readBytes(void* bytes, size_t count)
{
 read((char*)bytes, count);
 if(!good())
    throw save_error_unexpected_eof(count);
}
std::string savestream::readString()
{
 std::stringstream _out;
 while(1) {
    int in = get();
    if(!good())
        throw save_error_unexpected_eof(1);
    if(in == '\n')
        break;
    _out << (char)in;
 }
 return _out.str();
}
```

savestream.h

```
#ifndef SAVESTREAM_H
#define SAVESTREAM_H

#include <fstream>
#include <cstdlib>

class savestream : public std::fstream
{
 public:

 void writeBytes(void* bytes, size_t count);
 void writeString(std::string str);

 void readBytes(void* bytes, size_t count);
 std::string readString();
};

#endif
```

serializable.cpp

```
#include "serializable.h"

std::string serializable::getClassName() { return
   "serializable"; }
void serializable::saveToFile(savestream& svs)
{
 svs.writeString(getClassName());
}
void serializable::loadFromFile(savestream& svs)
{
 std::string name = svs.readString();
}
```

serializable.h

```
#ifndef SERIALIZABLE_H
#define SERIALIZABLE_H

#include "savestream.h"

class serializable
{
 public:
```

```
 virtual std::string getClassName();
 virtual void saveToFile(savestream& svs);
 virtual void loadFromFile(savestream& svs);
};

#endif
```