МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Объектно-ориентированное программирование»

Тема: Добавление логгирования

Студент гр. 9381	___________________	Колованов Р.А.
Преподаватель	___________________	Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Изучить парадигму объектно-ориентрированного программирования; реализовать классы для логгирования; изучить и реализовать паттерны проектирования *Bridge* и *Observer*.

Задание.

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

Обязательные требования:

- Реализована возможность 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.

Для начала были реализованы абстрактный класс *Logger* и его наследники *FileLogger* и *ConsoleLogger*. Класс *Logger* представляет собой базовый класс для различных видов логгера. Класс *FileLogger* осуществляет вывод сообщений в файл (взаимодействие с файлом реализовано по идиоме RAII). Класс *ConsoleLogger* осуществляет вывод сообщений в консоль. Реализация данный классов отделена от интерфейса при помощи паттерна проектирования Bridge. В качетсве интерфейса реализации логгера был реализован интерфейс *LoggerImplementation*. Наследюясь от данного интерфейса была реализована конкретные реализации логгера *FileLoggerImplementation* и *ConsoleLoggerImplementation*. Далее было реализованы классы для паттерна проектирования *Observer*. Класс *EventManager* является классом-издателем. Его можно поместить в класс, который мы хотим отслеживать. Далее был реализован интерфейс EventListener — интерфейс для классов-подписчиков, которые могут связываться с классами-издателями. В качестве производного класса для *EventListener* был реализован класс *LoggingListener*, который отлавливает от издателей сообщения для логгирования.

В программе используются умные указатели, поэтому очистка памяти для них не требуется. Для реализации GUI-интерфейса программы был использован фреймворк *Qt*.

Подробное описание классов приведено ниже (см. Раздел *Описание классов и структур*).

Разработанный программный код см. в приложении А.

Описание классов и структур.

Класс Logger.

Абстрактный класс. Используется в качестве общего интерфейса для классов FileLogger и ConsoleLogger.

Поля класса *Logger*:

Модификатор доступа	Название и тип поля	Предназначение	Значение по умолчанию
protected	*pILoggerImplementation implementation_*	Хранит адрес конкретной реализации логгера.	-

Методы класса *Logger*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*Logger(const pILoggerImplementation& implementation)*
public	*void*	log(std::ostringstream& message) = 0
public	-	*~Logger() = default*

Класс FileLogger.

Используется в качестве интерфейса для вызова методов вывода логов в файл. Внутри себя хранит указатель на реализацию методов вывода сообщений *LoggerImplamantation*.

Поля класса *FileLogger*:

Модификатор доступа	Название и тип поля	Предназначение	Значение по умолчанию
private	*std::string*	Хранит путь к файлу для	-

	filepath_;	вывода сообщений.	
private	*std::ofstream file_*	Хранит файловый поток вывода.	-
private	*bool error_*	Хранит инфорацию о том, был ли объект успешно создан.	

Методы класса *FileLogger*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*FileLogger(const std::string& filepath)*
public	*bool*	*isValid()*
public	*void*	*log(std::ostringstream& message)*
public	-	*~FileLogger()*

Класс ConsoleLogger.

Используется в качестве интерфейса для вызова методов вывода логов на консоль. Внутри себя хранит указатель на реализацию методов вывода сообщений *ILoggerImplamantation*.

Поля класса *FileLogger*:

Модификатор доступа	Название и тип поля	Предназначение	Значение по умолчанию
private	*std::ostream& stream_*	Хранит ссылку на поток вывода.	-

Методы класса *FileLogger*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*ConsoleLogger(std::ostream& stream)*
public	*void*	*log(std::ostringstream& message)*

Класс LoggerImplementation.

Является интерфейсом для классов реализации методов логгирования.

Методы класса *LoggerImplementation*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
protected	*std::string*	*getCurrentDateTime()*
public	*void*	*log(std::ostream& stream, std::ostringstream& message) = 0*
public	-	*~LoggerImplementation() = default*

Класс LoggerImplementation.

Является интерфейсом для классов реализации методов логгирования.

Методы класса *LoggerImplementation*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
protected	*std::string*	*getCurrentDateTime()*
public	*void*	*log(std::ostream& stream, std::ostringstream& message) = 0*
public	-	*~LoggerImplementation() = default*

Класс FileLoggerImplementation.

Содержит конкретную реализацию методов логгирования в файл.

Методы класса *FileLoggerImplementation*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*void*	*log(std::ostream& stream,*

		std::ostringstream& message)

Класс ConsoleLoggerImplementation.

Содержит конкретную реализацию методов логгирования в консоль.

Методы класса *ConsoleLoggerImplementation*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*void*	*log(std::ostream& stream, std::ostringstream& message)*

Класс EventManager.

Является классом-издателем. Его можно поместить в класс, который требуется отслеживать. Содержит указатели на подписчиков, которых можно оповестить о изменениях отслеживаемого объекта при помощи метода *notify*.

Поля класса *EventManager*:

Модификатор доступа	Название и тип поля	Предназначение	Значение по умолчанию
protected	*std::set<pEventListener> listeners*	Хранит указатели на классы-слушатели (подписчики).	-

Методы класса *EventManager*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*void*	*subscribe(pEventListener& listener)*
public	*void*	*unsubscribe(pEventListener& listener)*
public	*void*	*notify(const std::string& message)*
public	*void*	*notify(std::ostringstream& message)*

Класс EventListener.

Является интерфейсом для классов-слушателей (подписчиков).

Методы класса *EventListener*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*void*	*update(std::ostringstream& message) = 0*

Класс LoggingListener.

Является классом-слушателем. Используется для прослушки сообщений от отслеживаемых объектов, содержищих объект класса *EventManager*.

Поля класса *LoggingListener*:

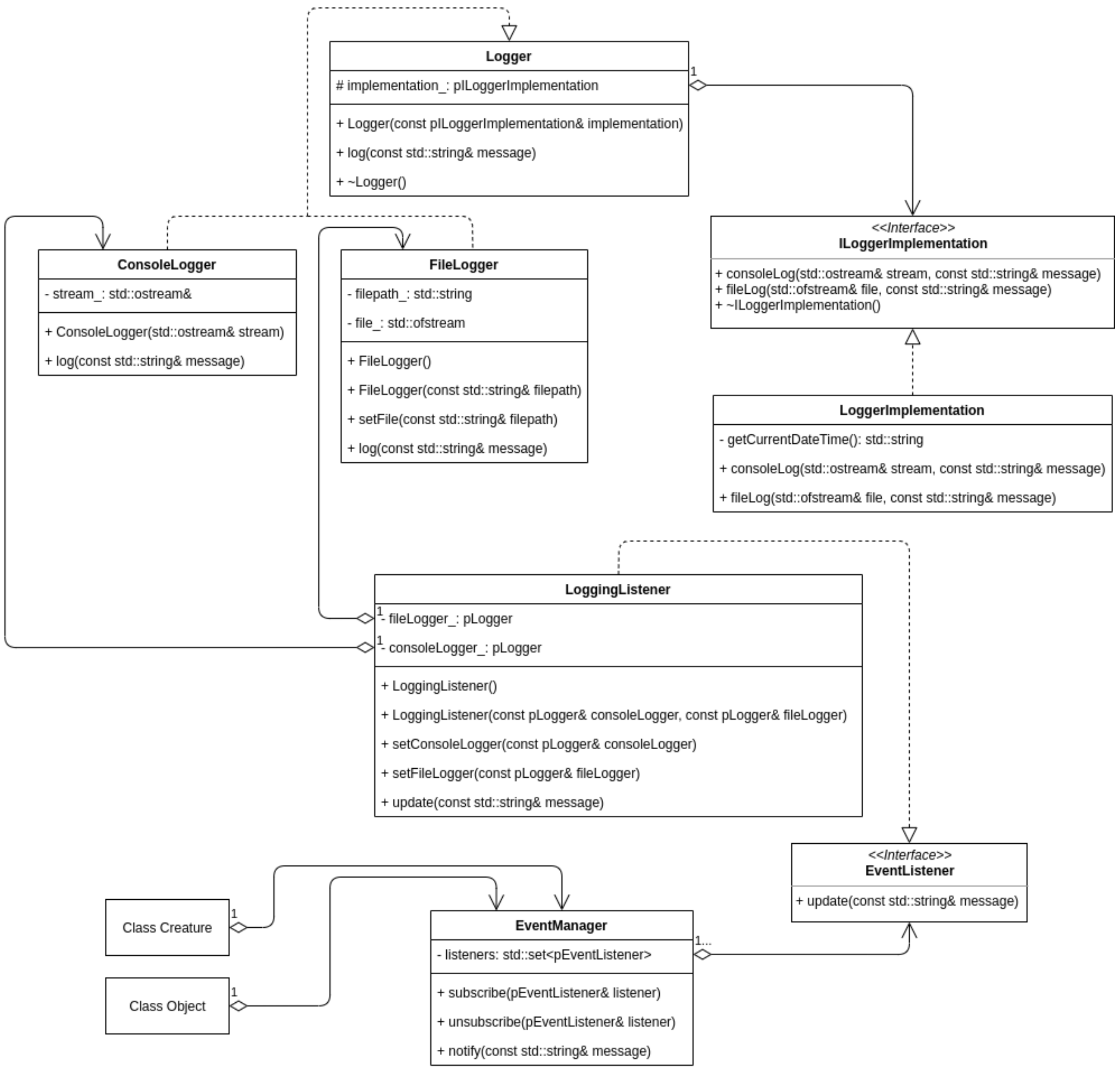
Модификатор доступа	Название и тип поля	Предназначение	Значение по умолчанию
private	*pLogger consoleLogger_*	Хранит логгер для вывода на консоль.	-
private	*pLogger fileLogger_*	Хранит логгер для вывода в файл.	-

Методы класса *LoggingListener*:

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*LoggingListener() = default*
public	-	*LoggingListener(const pLogger& consoleLogger, const pLogger&*

		fileLogger)
public	*void*	*setConsoleLogger(const pLogger& consoleLogger)*
public	*void*	*setFileLogger(const pLogger& fileLogger)*
public	*void*	*update(std::ostringstream& message)*
public	*void*	*update(const std::string& message)*

UML-диаграмма.

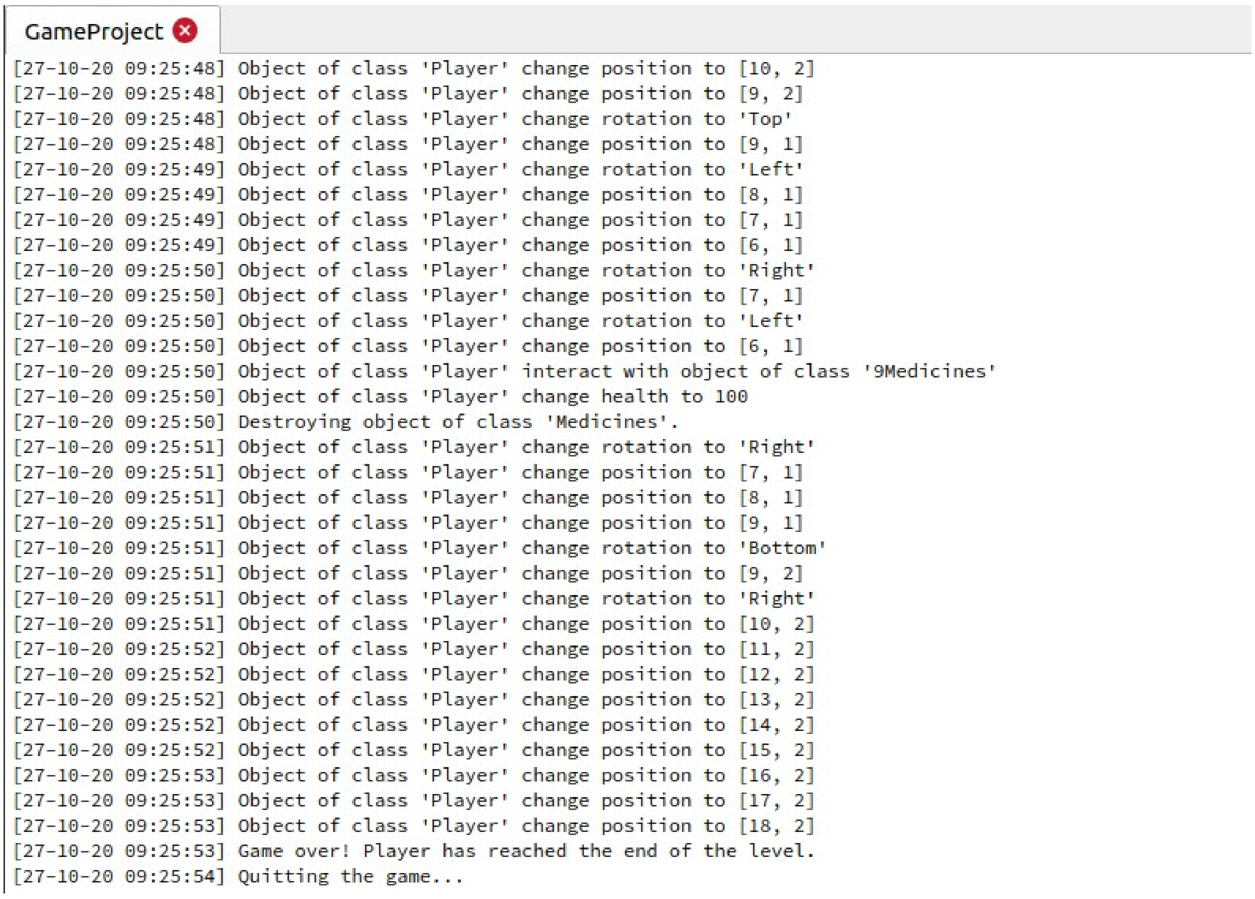


Logger
# implementation_: pILoggerImplementation
+ Logger(const pILoggerImplementation& implementation)
+ log(const std::string& message)
+ ~Logger()
1
<<Interface>>
ILoggerImplementation
+ consoleLog(std::ostream& stream, const std::string& message)
+ fileLog(std::ofstream& file, const std::string& message)
+ ~ILoggerImplementation()
LoggerImplementation
- getCurrentDateTime(): std::string
+ consoleLog(std::ostream& stream, const std::string& message)
+ fileLog(std::ofstream& file, const std::string& message)
ConsoleLogger
- stream_: std::ostream&
+ ConsoleLogger(std::ostream& stream)
+ log(const std::string& message)
FileLogger
- filepath_: std::string
- file_: std::ofstream
+ FileLogger()
+ FileLogger(const std::string& filepath)
+ setFile(const std::string& filepath)
+ log(const std::string& message)
LoggingListener
1
- fileLogger_: pLogger
1
- consoleLogger_: pLogger
+ LoggingListener()
+ LoggingListener(const pLogger& consoleLogger, const pLogger& fileLogger)
+ setConsoleLogger(const pLogger& consoleLogger)
+ setFileLogger(const pLogger& fileLogger)
+ update(const std::string& message)
<<Interface>>
EventListener
+ update(const std::string& message)
Class Creature
1
Class Object
1
EventManager
- listeners: std::set<pEventListener>
+ subscribe(pEventListener& listener)
+ unsubscribe(pEventListener& listener)
+ notify(const std::string& message)
1...

Тестирование.

Результаты тестирования представлены на рис. 1, 2, 3, 4.

Рисунок 1 — Логгирование в консоль



```
[27-10-20 09:25:48] Object of class 'Player' change position to [10, 2]
[27-10-20 09:25:48] Object of class 'Player' change position to [9, 2]
[27-10-20 09:25:48] Object of class 'Player' change rotation to 'Top'
[27-10-20 09:25:48] Object of class 'Player' change position to [9, 1]
[27-10-20 09:25:49] Object of class 'Player' change rotation to 'Left'
[27-10-20 09:25:49] Object of class 'Player' change position to [8, 1]
[27-10-20 09:25:49] Object of class 'Player' change position to [7, 1]
[27-10-20 09:25:49] Object of class 'Player' change position to [6, 1]
[27-10-20 09:25:50] Object of class 'Player' change rotation to 'Right'
[27-10-20 09:25:50] Object of class 'Player' change position to [7, 1]
[27-10-20 09:25:50] Object of class 'Player' change rotation to 'Left'
[27-10-20 09:25:50] Object of class 'Player' change position to [6, 1]
[27-10-20 09:25:50] Object of class 'Player' interact with object of class '9Medicines'
[27-10-20 09:25:50] Object of class 'Player' change health to 100
[27-10-20 09:25:50] Destroying object of class 'Medicines'.
[27-10-20 09:25:51] Object of class 'Player' change rotation to 'Right'
[27-10-20 09:25:51] Object of class 'Player' change position to [7, 1]
[27-10-20 09:25:51] Object of class 'Player' change position to [8, 1]
[27-10-20 09:25:51] Object of class 'Player' change position to [9, 1]
[27-10-20 09:25:51] Object of class 'Player' change rotation to 'Bottom'
[27-10-20 09:25:51] Object of class 'Player' change position to [9, 2]
[27-10-20 09:25:51] Object of class 'Player' change rotation to 'Right'
[27-10-20 09:25:51] Object of class 'Player' change position to [10, 2]
[27-10-20 09:25:52] Object of class 'Player' change position to [11, 2]
[27-10-20 09:25:52] Object of class 'Player' change position to [12, 2]
[27-10-20 09:25:52] Object of class 'Player' change position to [13, 2]
[27-10-20 09:25:52] Object of class 'Player' change position to [14, 2]
[27-10-20 09:25:52] Object of class 'Player' change position to [15, 2]
[27-10-20 09:25:53] Object of class 'Player' change position to [16, 2]
[27-10-20 09:25:53] Object of class 'Player' change position to [17, 2]
[27-10-20 09:25:53] Object of class 'Player' change position to [18, 2]
[27-10-20 09:25:53] Game over! Player has reached the end of the level.
[27-10-20 09:25:54] Quitting the game...
```

Файл log.txt

```
[27-10-20 09:25:27] Starting the game...
[27-10-20 09:25:27] Creating the game field...
[27-10-20 09:25:27] Creating the game field... Done.
[27-10-20 09:25:27] Object of class 'Player' change position to [2, 2]
[27-10-20 09:25:28] Object of class 'Player' change position to [2, 3]
[27-10-20 09:25:28] Object of class 'Player' change position to [2, 4]
[27-10-20 09:25:28] Object of class 'Player' change position to [2, 5]
[27-10-20 09:25:28] Object of class 'Player' change position to [2, 6]
[27-10-20 09:25:28] Object of class 'Player' change position to [2, 7]
[27-10-20 09:25:29] Object of class 'Player' change position to [2, 8]
[27-10-20 09:25:29] Object of class 'Player' change position to
```

```
[2, 9]
     [27-10-20 09:25:29] Object of class 'Player' change position to
[2, 10]
     [27-10-20 09:25:29] Object of class 'Player' change position to
[2, 11]
     [27-10-20 09:25:29] Object of class 'Player' change position to
[2, 12]
     [27-10-20 09:25:30] Object of class 'Player' change position to
[2, 13]
     [27-10-20 09:25:30] Object of class 'Player' change position to
[2, 14]
     [27-10-20 09:25:30] Object of class 'Player' change position to
[2, 15]
     [27-10-20 09:25:30] Object of class 'Player' change position to
[2, 16]
     [27-10-20 09:25:30] Object of class 'Player' interact with
object of class '9Medicines'
     [27-10-20 09:25:30] Object of class 'Player' change health to 98
     [27-10-20 09:25:30] Destroying object of class 'Medicines'.
     [27-10-20 09:25:31] Object of class 'Player' change position to
[2, 17]
     [27-10-20 09:25:31] Object of class 'Player' change position to
[2, 18]
     [27-10-20 09:25:31] Object of class 'Player' change rotation to
'Right'
     [27-10-20 09:25:31] Object of class 'Player' change position to
[3, 18]
     [27-10-20 09:25:31] Object of class 'Player' change position to
[4, 18]
     [27-10-20 09:25:31] Object of class 'Player' change position to
[5, 18]
     [27-10-20 09:25:31] Object of class 'Player' change position to
[6, 18]
     [27-10-20 09:25:32] Object of class 'Player' change position to
[7, 18]
     [27-10-20 09:25:32] Object of class 'Player' change position to
[8, 18]
     [27-10-20 09:25:32] Object of class 'Player' change position to
[9, 18]
     [27-10-20 09:25:32] Object of class 'Player' change position to
[10, 18]
     [27-10-20 09:25:32] Object of class 'Player' change position to
[11, 18]
     [27-10-20 09:25:33] Object of class 'Player' change position to
[12, 18]
     [27-10-20 09:25:33] Object of class 'Player' change position to
[13, 18]
     [27-10-20 09:25:33] Object of class 'Player' change position to
[14, 18]
     [27-10-20 09:25:33] Object of class 'Player' change position to
[15, 18]
     [27-10-20 09:25:33] Object of class 'Player' change position to
[16, 18]
     [27-10-20 09:25:34] Object of class 'Player' change position to
[17, 18]
```

```
     [27-10-20 09:25:34] Object of class 'Player' interact with
object of class '6Weapon'
     [27-10-20 09:25:34] Object of class 'Player' change attack
damage to 8
     [27-10-20 09:25:34] Destroying object of class 'Weapon'.
     [27-10-20 09:25:34] Object of class 'Player' change rotation to
'Top'
     [27-10-20 09:25:34] Object of class 'Player' change position to
[17, 17]
     [27-10-20 09:25:35] Object of class 'Player' change position to
[17, 16]
     [27-10-20 09:25:35] Object of class 'Player' change rotation to
'Right'
     [27-10-20 09:25:35] Object of class 'Player' interact with
object of class '5Armor'
     [27-10-20 09:25:35] Object of class 'Player' change protection
to 7
     [27-10-20 09:25:35] Destroying object of class 'Armor'.
     [27-10-20 09:25:35] Object of class 'Player' change rotation to
'Left'
     [27-10-20 09:25:35] Object of class 'Player' change position to
[16, 16]
     [27-10-20 09:25:35] Object of class 'Player' change position to
[15, 16]
     [27-10-20 09:25:36] Object of class 'Player' change position to
[14, 16]
     [27-10-20 09:25:36] Object of class 'Player' change position to
[13, 16]
     [27-10-20 09:25:36] Object of class 'Player' change position to
[12, 16]
     [27-10-20 09:25:36] Object of class 'Player' change rotation to
'Top'
     [27-10-20 09:25:36] Object of class 'Player' change position to
[12, 15]
     [27-10-20 09:25:37] Object of class 'Player' change position to
[12, 14]
     [27-10-20 09:25:37] Object of class 'Player' change rotation to
'Left'
     [27-10-20 09:25:37] Object of class 'Player' change position to
[11, 14]
     [27-10-20 09:25:38] Object of class 'Player' change position to
[10, 14]
     [27-10-20 09:25:38] Object of class 'Player' change position to
[9, 14]
     [27-10-20 09:25:38] Object of class 'Player' change rotation to
'Top'
     [27-10-20 09:25:38] Object of class 'Player' change position to
[9, 13]
     [27-10-20 09:25:38] Object of class 'Player' change rotation to
'Left'
     [27-10-20 09:25:38] Object of class 'Player' change position to
[8, 13]
     [27-10-20 09:25:38] Object of class 'Player' change position to
[7, 13]
     [27-10-20 09:25:39] Object of class 'Player' change position to
```

```
[6, 13]
      [27-10-20 09:25:39] Object of class 'Player' interact with
object of class '6Weapon'
      [27-10-20 09:25:39] Object of class 'Player' change attack
damage to 10
      [27-10-20 09:25:39] Destroying object of class 'Weapon'.
      [27-10-20 09:25:39] Object of class 'Player' change rotation to
'Right'
      [27-10-20 09:25:39] Object of class 'Player' change position to
[7, 13]
      [27-10-20 09:25:39] Object of class 'Player' change position to
[8, 13]
      [27-10-20 09:25:39] Object of class 'Player' change position to
[9, 13]
      [27-10-20 09:25:40] Object of class 'Player' change position to
[10, 13]
      [27-10-20 09:25:40] Object of class 'Player' change rotation to
'Top'
      [27-10-20 09:25:40] Object of class 'Player' change position to
[10, 12]
      [27-10-20 09:25:40] Object of class 'Player' change position to
[10, 11]
      [27-10-20 09:25:40] Object of class 'Player' change position to
[10, 10]
      [27-10-20 09:25:40] Object of class 'Player' change position to
[10, 9]
      [27-10-20 09:25:41] Object of class 'Player' change rotation to
'Left'
      [27-10-20 09:25:41] Object of class 'Player' change position to
[9, 9]
      [27-10-20 09:25:41] Object of class 'Player' change position to
[8, 9]
      [27-10-20 09:25:41] Object of class 'Player' change rotation to
'Top'
      [27-10-20 09:25:41] Object of class 'Player' change position to
[8, 8]
      [27-10-20 09:25:41] Object of class 'Player' change rotation to
'Left'
      [27-10-20 09:25:41] Object of class 'Player' change position to
[7, 8]
      [27-10-20 09:25:42] Object of class 'Player' change position to
[6, 8]
      [27-10-20 09:25:42] Object of class 'Player' interact with
object of class '5Armor'
      [27-10-20 09:25:42] Object of class 'Player' change protection
to 10
      [27-10-20 09:25:42] Destroying object of class 'Armor'.
      [27-10-20 09:25:42] Object of class 'Player' change rotation to
'Right'
      [27-10-20 09:25:42] Object of class 'Player' change position to
[7, 8]
      [27-10-20 09:25:42] Object of class 'Player' change position to
[8, 8]
      [27-10-20 09:25:42] Object of class 'Player' change position to
[9, 8]
```

```
      [27-10-20 09:25:43] Object of class 'Player' change position to
[10, 8]
      [27-10-20 09:25:43] Object of class 'Player' change position to
[11, 8]
      [27-10-20 09:25:43] Object of class 'Player' change rotation to
'Top'
      [27-10-20 09:25:43] Object of class 'Player' change position to
[11, 7]
      [27-10-20 09:25:43] Object of class 'Player' change position to
[11, 6]
      [27-10-20 09:25:44] Object of class 'Player' change rotation to
'Bottom'
      [27-10-20 09:25:44] Object of class 'Player' change position to
[11, 7]
      [27-10-20 09:25:44] Object of class 'Player' change rotation to
'Right'
      [27-10-20 09:25:44] Object of class 'Player' change position to
[12, 7]
      [27-10-20 09:25:44] Object of class 'Player' change position to
[13, 7]
      [27-10-20 09:25:44] Object of class 'Player' change rotation to
'Bottom'
      [27-10-20 09:25:44] Object of class 'Player' change position to
[13, 8]
      [27-10-20 09:25:44] Object of class 'Player' change rotation to
'Right'
      [27-10-20 09:25:44] Object of class 'Player' change position to
[14, 8]
      [27-10-20 09:25:45] Object of class 'Player' change position to
[15, 8]
      [27-10-20 09:25:45] Object of class 'Player' change position to
[16, 8]
      [27-10-20 09:25:45] Object of class 'Player' change position to
[17, 8]
      [27-10-20 09:25:45] Object of class 'Player' interact with
object of class '15LevelPassObject'
      [27-10-20 09:25:45] Destroying object of class
'LevelPassObject'.
      [27-10-20 09:25:45] Object of class 'Player' change rotation to
'Left'
      [27-10-20 09:25:45] Object of class 'Player' change position to
[16, 8]
      [27-10-20 09:25:46] Object of class 'Player' change position to
[15, 8]
      [27-10-20 09:25:46] Object of class 'Player' change position to
[14, 8]
      [27-10-20 09:25:46] Object of class 'Player' change position to
[13, 8]
      [27-10-20 09:25:46] Object of class 'Player' change position to
[12, 8]
      [27-10-20 09:25:46] Object of class 'Player' change rotation to
'Top'
      [27-10-20 09:25:46] Object of class 'Player' change position to
[12, 7]
      [27-10-20 09:25:47] Object of class 'Player' change position to
```

```
[12, 6]
     [27-10-20 09:25:47] Object of class 'Player' change position to
[12, 5]
     [27-10-20 09:25:47] Object of class 'Player' change position to
[12, 4]
     [27-10-20 09:25:47] Object of class 'Player' change position to
[12, 3]
     [27-10-20 09:25:47] Object of class 'Player' change position to
[12, 2]
     [27-10-20 09:25:48] Object of class 'Player' change rotation to
'Left'
     [27-10-20 09:25:48] Object of class 'Player' change position to
[11, 2]
     [27-10-20 09:25:48] Object of class 'Player' change position to
[10, 2]
     [27-10-20 09:25:48] Object of class 'Player' change position to
[9, 2]
     [27-10-20 09:25:48] Object of class 'Player' change rotation to
'Top'
     [27-10-20 09:25:48] Object of class 'Player' change position to
[9, 1]
     [27-10-20 09:25:49] Object of class 'Player' change rotation to
'Left'
     [27-10-20 09:25:49] Object of class 'Player' change position to
[8, 1]
     [27-10-20 09:25:49] Object of class 'Player' change position to
[7, 1]
     [27-10-20 09:25:49] Object of class 'Player' change position to
[6, 1]
     [27-10-20 09:25:50] Object of class 'Player' change rotation to
'Right'
     [27-10-20 09:25:50] Object of class 'Player' change position to
[7, 1]
     [27-10-20 09:25:50] Object of class 'Player' change rotation to
'Left'
     [27-10-20 09:25:50] Object of class 'Player' change position to
[6, 1]
     [27-10-20 09:25:50] Object of class 'Player' interact with
object of class '9Medicines'
     [27-10-20 09:25:50] Object of class 'Player' change health to
100
     [27-10-20 09:25:50] Destroying object of class 'Medicines'.
     [27-10-20 09:25:51] Object of class 'Player' change rotation to
'Right'
     [27-10-20 09:25:51] Object of class 'Player' change position to
[7, 1]
     [27-10-20 09:25:51] Object of class 'Player' change position to
[8, 1]
     [27-10-20 09:25:51] Object of class 'Player' change position to
[9, 1]
     [27-10-20 09:25:51] Object of class 'Player' change rotation to
'Bottom'
     [27-10-20 09:25:51] Object of class 'Player' change position to
[9, 2]
     [27-10-20 09:25:51] Object of class 'Player' change rotation to
```

```
'Right'
     [27-10-20 09:25:51] Object of class 'Player' change position to
[10, 2]
     [27-10-20 09:25:52] Object of class 'Player' change position to
[11, 2]
     [27-10-20 09:25:52] Object of class 'Player' change position to
[12, 2]
     [27-10-20 09:25:52] Object of class 'Player' change position to
[13, 2]
     [27-10-20 09:25:52] Object of class 'Player' change position to
[14, 2]
     [27-10-20 09:25:52] Object of class 'Player' change position to
[15, 2]
     [27-10-20 09:25:53] Object of class 'Player' change position to
[16, 2]
     [27-10-20 09:25:53] Object of class 'Player' change position to
[17, 2]
     [27-10-20 09:25:53] Object of class 'Player' change position to
[18, 2]
     [27-10-20 09:25:53] Game over! Player has reached the end of the
level.
     [27-10-20 09:25:54] Quitting the game...
```

Выводы.

Была изучена парадигма объектно-ориентрированного программирования. Были реализованны классы логгирования. При работе с файлами используется идиома RAII. Для отслеживаемых классов был перегружен оператор вывода в поток <<. Были изучены и реализованы паттерны проектирования *Bridge* и *Observer*. По мимо этого, был реализован GUI-интерфейс игры при помощи фреймворка Qt.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <QApplication>
#include "classes/mainwindow.h"

int main(int argc, char* argv[]) {
    QApplication app(argc, argv);
    MainWindow window;
    window.show();
    return app.exec();
}
```

Название файла: armor.h

```
#ifndef ARMOR_H
#define ARMOR_H

#include "memory"
#include "object.h"

typedef std::shared_ptr<class Armor> pArmor;


class Armor: public Object {
private:
    int protectionValue_;

public:
    explicit Armor(int protectionValue);
    pObject getCopy() const;
    void executeInteraction(Creature& creature);
    const std::type_info& getClass() const;
    Texture getTexture() const;
    bool getReusable() const;
};


#endif // ARMOR_H
```

Название файла: armor.cpp

```
#include "armor.h"
#include "armorfactory.h"

Armor::Armor(int protectionValue): protectionValue_(protectionValue) {}

pObject Armor::getCopy() const {
    pArmorFactory factory(new ArmorFactory);
    return pObject(factory->createArmor(protectionValue_));
}

void Armor::executeInteraction(Creature& creature) {
    if (creature.getProtection() < protectionValue_) {
        creature.setProtection(protectionValue_);
    }
}

const std::type_info &Armor::getClass() const {
    return typeid(Armor);
}

Texture Armor::getTexture() const {
    return kTextureObjectArmor;
}

bool Armor::getReusable() const {
    return false;
}
```

Название файла: armorfactory.h

```
#ifndef ARMOR_FACTORY_H
#define ARMOR_FACTORY_H

#include "objectfactory.h"
#include "armor.h"

typedef std::shared_ptr<class ArmorFactory> pArmorFactory;


class ArmorFactory: public ObjectFactory {
public:
    virtual pObject createObject();
    virtual pObject createArmor(int protectionValue);
};


#endif // ARMOR_FACTORY_H
```

Название файла: armorfactory.cpp

```
#include "armorfactory.h"

pObject ArmorFactory::createObject() {
    return pObject(new Armor(5));
}

pObject ArmorFactory::createArmor(int protectionValue) {
    return pObject(new Armor(protectionValue));
}
```

Название файла: cell.h

```
#ifndef CELL_H
#define CELL_H

#include <memory>
#include "point2d.h"
#include "celltype.h"
#include "texture.h"
#include "object.h"
```

```
typedef std::shared_ptr<class Cell> pCell;
typedef std::shared_ptr<std::shared_ptr<class Cell>> ppCell;


class Cell {
private:
    bool passable_ = false;
    CellType type_ = kCellTypeNone;
    Texture texture_ = kTextureVoid;
    Position2D position_;
    pObject object_;

public:
    Cell() = default;
    explicit Cell(Position2D position, Texture texture = kTextureVoid,
CellType type = kCellTypeNone, pObject object = nullptr);
    Cell(const Cell& other);
    Cell(Cell&& other);
    ~Cell() = default;

    Cell& operator=(const Cell& other);
    Cell& operator=(Cell&& other);

    bool isPassable() const;
    bool getPassible() const;
    pConstObject getObject() const;
    Texture getTexture() const;
    CellType getType() const;
    Position2D getPosition() const;
    pObject& getObject();
    void setObject(const pObject& object);
    void setTexture(Texture texture);
    void changeType(CellType type);
};


#endif // CELL_H
```

Название файла: cell.cpp

```
#include "cell.h"

Cell::Cell(Position2D coords, Texture texture, CellType type, pObject
object) {
    position_ = coords;
    texture_ = texture;
    object_ = object;
    changeType(type);
}

Cell::Cell(const Cell& other) {
    operator=(other);
}

Cell::Cell(Cell&& other) {
    position_ = other.position_;
    texture_ = other.texture_;
    type_ = other.type_;
    passable_ = other.passable_;
    object_ = other.object_;
}

Cell& Cell::operator=(const Cell& other) {
    if (this != &other) {
        position_ = other.position_;
        texture_ = other.texture_;
        type_ = other.type_;
        passable_ = other.passable_;

        if (other.object_ != nullptr) {
            object_ = other.object_->getCopy();
        }
    }

    return *this;
}

Cell& Cell::operator=(Cell&& other) {
    if (this != &other) {
        std::swap(position_, other.position_);
        std::swap(texture_, other.texture_);
        std::swap(type_, other.type_);
        std::swap(passable_, other.passable_);
        std::swap(object_, other.object_);
    }

    return *this;
}

bool Cell::isPassable() const {
    return passable_ && object_ == nullptr;
}

bool Cell::getPassible() const {
    return passable_;
}

pConstObject Cell::getObject() const {
    return object_;
}

Texture Cell::getTexture() const {
    return texture_;
}

CellType Cell::getType() const {
    return type_;
}

Position2D Cell::getPosition() const {
    return position_;
}

pObject& Cell::getObject() {
    return object_;
}

void Cell::setObject(const pObject& object) {
    object_ = object;
}
```

```
void Cell::setTexture(Texture texture) {
    texture_ = texture;
}

void Cell::changeType(CellType type) {
    type_ = type;

    switch (type) {
    case kCellTypeEmpty:
    case kCellTypeEntry:
    case kCellTypeExit:
        passable_ = true;
        return;

    case kCellTypeNone:
    case kCellTypeWall:
    default:
        passable_ = false;
        return;
    }
}
```

Название файла: celltype.h

```
#ifndef CELL_TYPE_H
#define CELL_TYPE_H

enum CellType {
    kCellTypeNone,
    kCellTypeEmpty,
    kCellTypeWall,
    kCellTypeEntry,
    kCellTypeExit
};

#endif // CELL_TYPE_H
```

Название файла: creature.h

```
#ifndef CREATURE_H
#define CREATURE_H

#include <memory>
#include "point2d.h"
#include "direction.h"
#include "texture.h"
#include "interactionstrategy.h"

typedef std::shared_ptr<class Creature> pCreature;
typedef std::shared_ptr<class Object> pObject;


class Creature {
private:
    int health_;
    int maxHealth_;
    int attackDamage_;
    int protection_;
    Position2D position_;
    Rotation rotation_ = kDirectionBottom;

public:
    virtual void interact(pObject& object) = 0;
    virtual Texture getTexture() const = 0;
    virtual ~Creature() =default;

    Rotation getRotation() const;
    Position2D getPosition() const;
    int getHealth() const;
    int getMaxHealth() const;
    int getAttackDamage() const;
    int getProtection() const;
    void setRotation(Rotation rotation);
    void setPosition(Position2D position);
    void setHealth(int health);
    void setMaxHealth(int maxHealth);
    void setAttackDamage(int damage);
    void setProtection(int protection);
};


#endif // CREATURE_H
```

Название файла: creature.cpp

```
#include "creature.h"

Rotation Creature::getRotation() const {
    return rotation_;
}

Position2D Creature::getPosition() const {
    return position_;
}

int Creature::getHealth() const {
    return health_;
}

int Creature::getMaxHealth() const {
    return maxHealth_;
}

int Creature::getAttackDamage() const {
    return attackDamage_;
}

int Creature::getProtection() const {
    return protection_;
}

void Creature::setRotation(Rotation rotation) {
    rotation_ = rotation;
}

void Creature::setPosition(Position2D position) {
    position_ = position;
}
```

```
void Creature::setHealth(int health) {
    if (health > maxHealth_) {
        health_ = maxHealth_;
    } else {
        health_ = health;
    }
}

void Creature::setMaxHealth(int maxHealth) {
    maxHealth_ = maxHealth;
}

void Creature::setAttackDamage(int damage) {
    attackDamage_ = damage;
}

void Creature::setProtection(int protection) {
    protection_ = protection;
}
```

Название файла: direction.h

```
#ifndef DIRECTION_H
#define DIRECTION_H

enum Direction {
    kDirectionTop,
    kDirectionLeft,
    kDirectionRight,
    kDirectionBottom
};

typedef Direction Rotation;

#endif // DIRECTION_H
```

Название файла: exception.h

```
#ifndef EXCEPTION_H
#define EXCEPTION_H

#include <string>


class Exception {
private:
    std::string error_;

public:
    Exception(const std::string& error);
    const std::string& getError() const;
};


#endif // EXCEPTION_H
```

Название файла: exception.cpp

```
#include "exception.h"

Exception::Exception(const std::string& error): error_(error) {}

const std::string& Exception::getError() const {
    return error_;
}
```

Название файла: field.h

```
#ifndef FIELD_H
#define FIELD_H

#include <memory>
#include "cell.h"
#include "point2d.h"

typedef std::unique_ptr<class Field> pField;


class Field {
private:
    static pField instance_;

    Size2D size_ = Size2D(0, 0);
    ppCell cells_ = nullptr;

    Field(const Size2D& size);
    Field(const Field& other);
    Field(Field&& other);

    Field& operator=(const Field& other);
    Field& operator=(Field&& other);

    class FieldIterator;
    class ConstFieldIterator;

public:
    static Field& initInstance(const Size2D& size);
    static Field& getInstance();
    static void deleteInstance();
    static bool isInstanceCreated();
    Cell& getCell(const Position2D& position);
    const Cell& getCell(const Position2D& position) const;
    Size2D getSize() const;
    FieldIterator begin();
    FieldIterator end();
    const ConstFieldIterator begin() const;
    const ConstFieldIterator end() const;
};


class Field::FieldIterator {
    Position2D position_;

public:
    explicit FieldIterator(const Position2D& position);

    bool operator==(const FieldIterator& other) const;
    bool operator!=(const FieldIterator& other) const;
    FieldIterator& operator++();
    FieldIterator operator++(int);
    Cell& operator*();
};

class Field::ConstFieldIterator {
    Position2D position_;

public:
    explicit ConstFieldIterator(const Position2D& position);

    bool operator==(const ConstFieldIterator& other) const;
    bool operator!=(const ConstFieldIterator& other) const;
    ConstFieldIterator& operator++();
    ConstFieldIterator operator++(int);
    const Cell& operator*() const;
};


#endif // FIELD_H
```

Название файла: field.cpp

```
#include "field.h"
#include "exception.h"
#include <iostream>

pField Field::instance_ = nullptr;

Field::Field(const Size2D& size): size_(size) {
    cells_ = ppCell(new pCell[size.y], std::default_delete<pCell[]>());

    for (size_t y = 0; y < size.y; y++) {
        cells_.get()[y] = pCell(new Cell[size.x],
std::default_delete<Cell[]>());

        for (size_t x = 0; x < size.x; x++) {
            cells_.get()[y].get()[x] = Cell(Position2D(x, y));
        }
    }
}

Field::Field(const Field& other) {
    size_ = other.size_;

    if (other.cells_ != nullptr) {
        cells_ = ppCell(new pCell[size_.y],
std::default_delete<pCell[]>());

        for (size_t y = 0; y < size_.y; y++) {
            cells_.get()[y] = pCell(new Cell[size_.x],
std::default_delete<Cell[]>());

            for (size_t x = 0; x < size_.x; x++) {
                cells_.get()[y].get()[x] = other.cells_.get()[y].get()
[x];
            }
        }
    }
}
```

```
      Field::Field(Field&& other) {
          size_ = other.size_;
          cells_ = other.cells_;
      }

      Field& Field::operator=(const Field& other) {
          if (this != &other) {
              size_ = other.size_;

              if (other.cells_ != nullptr) {
                  cells_ = ppCell(new pCell[size_.y],
std::default_delete<pCell[]>());

                  for (size_t y = 0; y < size_.y; y++) {
                      cells_.get()[y] = pCell(new Cell[size_.x],
std::default_delete<Cell[]>());

                      for (size_t x = 0; x < size_.x; x++) {
                          cells_.get()[y].get()[x] = other.cells_.get()
[y].get()[x];
                      }
                  }
              }
          }

          return *this;
      }

      Field& Field::operator=(Field&& other) {
          if (this != &other) {
              std::swap(size_, other.size_);
              std::swap(cells_, other.cells_);
          }

          return *this;
      }

      Field& Field::initInstance(const Size2D& size) {
          if (!isInstanceCreated()) {
              instance_ = pField(new Field(size));
          }
          return *instance_;
      }

      Field& Field::getInstance() {
          if (!isInstanceCreated()) {
              instance_ = pField(new Field(Size2D(10, 10)));
          }
          return *instance_;
      }

      void Field::deleteInstance() {
          Field::instance_ = nullptr;
      }

      bool Field::isInstanceCreated() {
          return Field::instance_ != nullptr;
      }

      Cell& Field::getCell(const Position2D& position) {
          if (position.x >= size_.x || position.y >= size_.y) {
              throw Exception("Method Field::getCell. Out of range.");
          }
          return cells_.get()[position.y].get()[position.x];
      }

      const Cell& Field::getCell(const Position2D& position) const {
          if (position.x >= size_.x || position.y >= size_.y) {
              throw Exception("Method Field::getCell. Out of range.");
          }
          return cells_.get()[position.y].get()[position.x];
      }

      Size2D Field::getSize() const {
          return size_;
      }

      Field::FieldIterator Field::begin() {
          return FieldIterator(Position2D(0, 0));
      }

      Field::FieldIterator Field::end() {
          return FieldIterator(Position2D(0, getSize().y));
      }

      const Field::ConstFieldIterator Field::begin() const {
          return ConstFieldIterator(Position2D(0, 0));
      }

      const Field::ConstFieldIterator Field::end() const {
          return ConstFieldIterator(Position2D(0, getSize().y));
      }

      Field::FieldIterator::FieldIterator(const Position2D& position):
position_(position) {}

      bool Field::FieldIterator::operator==(const FieldIterator& other) const
{
          return position_ == other.position_;
      }

      bool Field::FieldIterator::operator!=(const FieldIterator& other) const
{
          return !operator==(other);
      }

      Field::FieldIterator& Field::FieldIterator::operator++() {
          Field& field = Field::getInstance();

          if (position_.x + 1 >= field.getSize().x) {
              position_.y++;
              position_.x = 0;
          } else {
              position_.x++;
          }

          return *this;
      }

      Field::FieldIterator Field::FieldIterator::operator++(int) {
          FieldIterator iterator(*this);
```

```
    operator++();
    return iterator;
}

Cell& Field::FieldIterator::operator*() {
    Field& field = Field::getInstance();
    return field.getCell(position_);
}

Field::ConstFieldIterator::ConstFieldIterator(const Position2D& position): position_(position) {}

bool Field::ConstFieldIterator::operator==(const Field::ConstFieldIterator& other) const {
    return position_ == other.position_;
}

bool Field::ConstFieldIterator::operator!=(const Field::ConstFieldIterator& other) const {
    return !operator==(other);
}

Field::ConstFieldIterator& Field::ConstFieldIterator::operator++() {
    const Field& field = Field::getInstance();

    if (position_.x + 1 >= field.getSize().x) {
        position_.y++;
        position_.x = 0;
    } else {
        position_.x++;
    }

    return *this;
}

Field::ConstFieldIterator Field::ConstFieldIterator::operator++(int) {
    ConstFieldIterator iterator(*this);
    operator++();
    return iterator;
}

const Cell& Field::ConstFieldIterator::operator*() const {
    const Field& field = Field::getInstance();
    return field.getCell(position_);
}
```

Название файла: gamecontroller.h

```
#ifndef GAMECONTROLLER_H
#define GAMECONTROLLER_H

#include "player.h"
#include "field.h"

typedef std::shared_ptr<const Player> pConstPlayer;


class GameController {
private:
    pPlayer player_;
    bool gameOver_ = false;

public:
    GameController();
    void createFieldMap();
    const Field& getField() const;
    pConstPlayer getPlayer() const;
    bool isPlayerReachedExit() const;
    void movePlayer(Direction direction);
    void executePlayerInteraction();
    bool isGameOver();
};


#endif // GAMECONTROLLER_H
```

Название файла: gamecontroller.cpp

```
#include "gamecontroller.h"
#include "medicinesfactory.h"
#include "weaponfactory.h"
#include "armorfactory.h"
#include "levelpassobjectfactory.h"
#include <iostream>

GameController::GameController() {
    createFieldMap();
}

void GameController::createFieldMap() {
    Field& field = Field::initInstance(Size2D(20, 20));
    pMedicinesFactory medicinesFactory(new MedicinesFactory);
    pArmorFactory armorFactory(new ArmorFactory);
    pWeaponFactory weaponFactory(new WeaponFactory);
    pLevelPassObjectFactory levelPassFactory(new
LevelPassObjectFactory);

    int textureMap[20][20] = {
        {6, 2, 2, 2, 8, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 7},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 12, 2, 2, 2, 2, 2, 9, 1, 1, 1, 13, 2, 2, 2, 2, 14},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 12, 2, 2, 2, 9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 4, 2, 2, 2, 2, 2, 9, 1, 1, 1, 13, 2, 2, 2, 2, 14},
        {3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3},
        {4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5}
    };
```

```
          int cellTypeMap[20][20] = {
              {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2},
              {2, 1, 3, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 2},
              {2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2},
              {2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2},
              {2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2},
              {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}
          };

          Position2D entryPoint;

          for (Cell& cell : field) {
              Position2D coords = cell.getPosition();
              cell.changeType(static_cast<CellType>(cellTypeMap[coords.y]
[coords.x]));
              cell.setTexture(static_cast<Texture>(textureMap[coords.y]
[coords.x]));

              if (cell.getType() == kCellTypeEntry) {
                  entryPoint = coords;
              }
          }

          player_ = pPlayer(new Player(entryPoint));
          player_->setMaxHealth(100);
          player_->setHealth(100);
          player_->setAttackDamage(3);
          player_->setProtection(0);

          field.getCell(Position2D(2, 17)).setObject(medicinesFactory-
>createMedicines(25));
          field.getCell(Position2D(5, 1)).setObject(medicinesFactory-
>createMedicines(25));
          field.getCell(Position2D(18, 16)).setObject(armorFactory-
>createArmor(7));
          field.getCell(Position2D(18, 18)).setObject(weaponFactory-
>createWeapon(8));
          field.getCell(Position2D(5, 8)).setObject(armorFactory-
>createArmor(10));
          field.getCell(Position2D(5, 13)).setObject(weaponFactory-
>createWeapon(10));
          field.getCell(Position2D(18, 8)).setObject(levelPassFactory-
>createObject());
      }

      const Field& GameController::getField() const {
          return Field::getInstance();
```

```
}

pConstPlayer GameController::getPlayer() const {
    return pConstPlayer(player_);
}

bool GameController::isPlayerReachedExit() const {
    const Field& field = Field::getInstance();
    return field.getCell(player_->getPosition()).getType() ==
kCellTypeExit;
}

void GameController::movePlayer(Direction direction) {
    Field& field = Field::getInstance();
    Position2D newPosition = Position2D(player_->getPosition().x,
player_->getPosition().y);

    newPosition.shift(direction);
    player_->setRotation(direction);

    if (field.getCell(newPosition).isPassable()) {
        player_->setPosition(newPosition);
    }

    if (isPlayerReachedExit() && player_->getPassFounded()) {
        gameOver_ = true;
    }
}

void GameController::executePlayerInteraction() {
    Field& field = Field::getInstance();
    Position2D interactionPosition = player_->getPosition();

    interactionPosition.shift(player_->getRotation());

    Cell& cell = field.getCell(interactionPosition);
    pObject& object = cell.getObject();

    *player_ <= object; // Взаимодействие через оператор <=
}

bool GameController::isGameOver() {
    return gameOver_;
}
```

Название файла: interactionnone.h

```
#ifndef INTERACTION_NONE_H
#define INTERACTION_NONE_H

#include "interactionstrategy.h"

typedef std::shared_ptr<class InteractionNone> pInteractionNone;


class InteractionNone: public InteractionStrategy {
public:
    void interact(Creature& creature, pObject& object);
};
```

```
#endif // INTERACTION_NONE_H
```

Название файла: interactionnone.cpp

```
#include "interactionnone.h"

void InteractionNone::interact(Creature&, pObject&) {}
```

Название файла: interactionstrategy.h

```
#ifndef INTERACTION_STRATEGY_H
#define INTERACTION_STRATEGY_H

#include <memory>
#include "object.h"
#include "creature.h"

typedef std::shared_ptr<class InteractionStrategy> pInteractionStrategy;
typedef std::shared_ptr<class Creature> pCreature;
typedef std::shared_ptr<class Object> pObject;


class InteractionStrategy {
public:
    virtual void interact(Creature& creature, pObject& object) = 0;
    virtual ~InteractionStrategy() = default;
};


#endif // INTERACTION_STRATEGY_H
```

Название файла: interactionuse.h

```
#ifndef INTERACTION_USE_H
#define INTERACTION_USE_H

#include "interactionstrategy.h"

typedef std::shared_ptr<class InteractionUse> pInteractionUse;


class InteractionUse: public InteractionStrategy {
public:
    void interact(Creature& creature, pObject& object);
};


#endif // INTERACTION_USE_H
```

Название файла: interactionuse.cpp

```
#include "field.h"
#include "interactionuse.h"

void InteractionUse::interact(Creature& creature, pObject& object) {
    if (object != nullptr) {
        object->executeInteraction(creature);

        if (!object->getReusable()) {
            object = nullptr;
        }
    }
}
```

Название файла: levelpassobject.h

```
#ifndef LEVEL_PASS_OBJECT_H
#define LEVEL_PASS_OBJECT_H

#include "object.h"


class LevelPassObject: public Object {
public:
    pObject getCopy() const;
    const std::type_info& getClass() const;
    Texture getTexture() const;
    void executeInteraction(Creature& creature);
    bool getReusable() const;
};


#endif // LEVEL_PASS_OBJECT_H
```

Название файла: levelpassobject.cpp

```
#include "levelpassobject.h"
#include "levelpassobjectfactory.h"
#include "player.h"

pObject LevelPassObject::getCopy() const {
    pLevelPassObjectFactory factory(new LevelPassObjectFactory);
    return pObject(factory->createObject());
}

const std::type_info &LevelPassObject::getClass() const {
    return typeid(LevelPassObject);
}

Texture LevelPassObject::getTexture() const {
    return kTextureObjectLevelPass;
}

void LevelPassObject::executeInteraction(Creature& creature) {
    try {
        Player& player = dynamic_cast<Player&>(creature);
        player.setPassFounded(true);
    } catch (std::bad_cast) {
        return;
    }
}

bool LevelPassObject::getReusable() const {
    return false;
}
```

Название файла: levelpassobjectfactory.h

```
#ifndef LEVEL_PASS_OBJECT_FACTORY_H
#define LEVEL_PASS_OBJECT_FACTORY_H

#include "objectfactory.h"
#include "levelpassobject.h"

typedef std::shared_ptr<class LevelPassObjectFactory>
pLevelPassObjectFactory;


class LevelPassObjectFactory: public ObjectFactory {
public:
    virtual pObject createObject();
};


#endif // LEVEL_PASS_OBJECT_FACTORY_H
```

Название файла: levelpassobjectfactory.cpp

```
#include "levelpassobjectfactory.h"

pObject LevelPassObjectFactory::createObject() {
    return pObject(new LevelPassObject);
}
```

Название файла: mainwindow.h

```
#ifndef MAIN_WINDOW_H
#define MAIN_WINDOW_H

#include <QMainWindow>
#include <QGraphicsView>
#include <QGraphicsScene>
#include <QImage>
#include <QLabel>
#include <QMap>
#include "gamecontroller.h"
#include "texture.h"

QT_BEGIN_NAMESPACE
namespace Ui {
    class MainWindow;
}
QT_END_NAMESPACE

typedef std::shared_ptr<Ui::MainWindow> pMainWindowUi;
typedef std::shared_ptr<QGraphicsView> pQGraphicsView;
typedef std::shared_ptr<QGraphicsScene> pQGraphicsScene;
typedef std::shared_ptr<QPixmap> pQPixmap;
typedef std::shared_ptr<QLabel> pQLabel;


class MainWindow: public QMainWindow {
    Q_OBJECT

private:
    pMainWindowUi ui;
    pQGraphicsView view;
    pQGraphicsScene scene;
    pQPixmap fieldPixelMap;
    pQLabel healthLabel;
    pQLabel attackLabel;
    pQLabel armorLabel;
    GameController controller;
    QMap<Texture, QImage> textures;
    bool screenPinning = false;
    bool isPressed = false;

public:
    MainWindow(QWidget* parent = nullptr);
    void updateScene();
    void keyPressEvent(QKeyEvent* event);
    void keyReleaseEvent(QKeyEvent* event);
};


#endif // MAIN_WINDOW_H
```

Название файла: mainwindow.cpp

```
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QMap>
#include <QKeyEvent>
#include <QMessageBox>
#include <iostream>
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "field.h"

MainWindow::MainWindow(QWidget *parent): QMainWindow(parent), ui(new
Ui::MainWindow) {
    ui->setupUi(this);

    view = pQGraphicsView(new QGraphicsView(this));
    scene = pQGraphicsScene(new QGraphicsScene(this));
    healthLabel = pQLabel(new QLabel(this));
    attackLabel = pQLabel(new QLabel(this));
    armorLabel = pQLabel(new QLabel(this));

    textures[kTextureVoid] = QImage(":/textures/tiles/tile_00.png");
    textures[kTextureWoodFloor1] =
QImage(":/textures/tiles/tile_100.png");
    textures[kTextureWoodWall1] =
QImage(":/textures/tiles/tile_120.png");
    textures[kTextureWoodWall2] =
QImage(":/textures/tiles/tile_147.png");
    textures[kTextureWoodWall3] =
QImage(":/textures/tiles/tile_145.png");
    textures[kTextureWoodWall4] =
QImage(":/textures/tiles/tile_146.png");
    textures[kTextureWoodWall5] =
QImage(":/textures/tiles/tile_118.png");
    textures[kTextureWoodWall6] =
QImage(":/textures/tiles/tile_119.png");
```

```
          textures[kTextureWoodWall7] =
QImage(":/textures/tiles/tile_121.png");
          textures[kTextureWoodWall8] =
QImage(":/textures/tiles/tile_123.png");
          textures[kTextureWoodWall9] =
QImage(":/textures/tiles/tile_124.png");
          textures[kTextureWoodWall10] =
QImage(":/textures/tiles/tile_122.png");
          textures[kTextureWoodWall11] =
QImage(":/textures/tiles/tile_148.png");
          textures[kTextureWoodWall12] =
QImage(":/textures/tiles/tile_151.png");
          textures[kTextureWoodWall13] =
QImage(":/textures/tiles/tile_149.png");
          textures[kTextureEntry] = QImage(":/textures/tiles/tile_132.png");
          textures[kTextureExit] = QImage(":/textures/tiles/tile_133.png");
          textures[kTextureShadow1] =
QImage(":/textures/tiles/shadow_01.png");
          textures[kTextureShadow2] =
QImage(":/textures/tiles/shadow_02.png");
          textures[kTextureShadow3] =
QImage(":/textures/tiles/shadow_03.png");
          textures[kTextureShadow4] =
QImage(":/textures/tiles/shadow_04.png");
          textures[kTextureCell] = QImage(":/textures/tiles/cell.png");
          textures[kTexturePlayerStandT] =
QImage(":/textures/player/player_stand_t.png");
          textures[kTexturePlayerStandB] =
QImage(":/textures/player/player_stand_d.png");
          textures[kTexturePlayerStandR] =
QImage(":/textures/player/player_stand_r.png");
          textures[kTexturePlayerStandL] =
QImage(":/textures/player/player_stand_l.png");
          textures[kTextureObjectMedicines] =
QImage(":/textures/tiles/tile_290.png");
          textures[kTextureObjectArmor] =
QImage(":/textures/tiles/tile_129.png");
          textures[kTextureObjectWeapon] =
QImage(":/textures/tiles/tile_129_2.png");
          textures[kTextureObjectLevelPass] =
QImage(":/textures/tiles/tile_key.png");

          view-
>setHorizontalScrollBarPolicy(Qt::ScrollBarPolicy::ScrollBarAlwaysOff);
          view-
>setVerticalScrollBarPolicy(Qt::ScrollBarPolicy::ScrollBarAlwaysOff);
          view->setStyleSheet("background-color: black;");
          view->setScene(scene.get());

          healthLabel->move(25, 25);
          attackLabel->move(25, 45);
          armorLabel->move(25, 65);
          healthLabel->setStyleSheet("QLabel { font-weight: bold; font-size:
16px; color: white; }");
          attackLabel->setStyleSheet("QLabel { font-weight: bold; font-size:
16px; color: white; }");
          armorLabel->setStyleSheet("QLabel { font-weight: bold; font-size:
16px; color: white; }");

          if (!screenPinning) {
              view->setDragMode(QGraphicsView::ScrollHandDrag);
          }

          updateScene();
          setCentralWidget(view.get());
      }

      void MainWindow::updateScene() {
          const Field& field = controller.getField();
          pConstPlayer player = controller.getPlayer();
          Size2D fieldSize = field.getSize();

          if (fieldPixelMap == nullptr ||
              static_cast<size_t>(fieldPixelMap->width()) != fieldSize.x * 64
||
              static_cast<size_t>(fieldPixelMap->height()) != fieldSize.y *
64)
          {
              fieldPixelMap = pQPixmap(new QPixmap(fieldSize.x * 64,
fieldSize.y * 64));
          }

          QPainter painter(fieldPixelMap.get());

          for (const Cell& cell : field) {
              Position2D coords = cell.getPosition();

              painter.drawImage(coords.x * 64, coords.y * 64,
textures[cell.getTexture()]);

              if (cell.getType() == kCellTypeEntry) {
                  painter.drawImage(coords.x * 64, coords.y * 64,
textures[kTextureEntry]);
              } else if (cell.getType() == kCellTypeExit) {
                  painter.drawImage(coords.x * 64, coords.y * 64,
textures[kTextureExit]);
              }

              if (cell.getPassible()) {
                  painter.drawImage(coords.x * 64, coords.y * 64,
textures[kTextureCell]);
              }

              if (cell.getObject() != nullptr) {
                  painter.drawImage(coords.x * 64, coords.y * 64,
textures[cell.getObject()->getTexture()]);
              }

              if (coords.y == 0) {
                  painter.drawImage(coords.x * 64, coords.y * 64,
textures[kTextureShadow2]);
              }

              if (coords.x == 0) {
                  painter.drawImage(coords.x * 64, coords.y * 64,
textures[kTextureShadow1]);
              }

              if (coords.y == fieldSize.y - 1) {
                  painter.drawImage(coords.x * 64, coords.y * 64,
textures[kTextureShadow4]);
              }
              if (coords.x == fieldSize.x - 1) {
                  painter.drawImage(coords.x * 64, coords.y * 64,
textures[kTextureShadow3]);
              }
          }

          painter.drawImage(player->getPosition().x * 64, player-
>getPosition().y * 64, textures[player->getTexture()]);

          healthLabel->setText("Health: " + QString::number(player-
>getHealth()));
          attackLabel->setText("Attack: " + QString::number(player-
>getAttackDamage()));
          armorLabel->setText("Armor: " + QString::number(player-
>getProtection()));

          scene->clear();
          scene->addPixmap(*fieldPixelMap);

          if (screenPinning) {
              view->centerOn(player->getPosition().x * 64 + 32, player-
>getPosition().y * 64 + 32);
          }
      }

      void MainWindow::keyPressEvent(QKeyEvent* event) {
          if (!isPressed) {
              isPressed = true;

              if (event->key() == Qt::Key_W) {
                  controller.movePlayer(kDirectionTop);
              } else if (event->key() == Qt::Key_S) {
                  controller.movePlayer(kDirectionBottom);
              } else if (event->key() == Qt::Key_A) {
                  controller.movePlayer(kDirectionLeft);
              } else if (event->key() == Qt::Key_D) {
                  controller.movePlayer(kDirectionRight);
              } else if (event->key() == Qt::Key_E) {
                  controller.executePlayerInteraction();
              }

              updateScene();

              if (controller.isGameOver()) {
                  QMessageBox::information(this, "Game over", "Great job,
level passed!");
                  QApplication::quit();
              }
          }
      }

      void MainWindow::keyReleaseEvent(QKeyEvent* event) {
          if (!event->isAutoRepeat()) {
              isPressed = false;
          }
      }
```

Название файла: medicines.h

```
#ifndef MEDICINES_H
#define MEDICINES_H

#include <memory>
#include "object.h"
#include "creature.h"

typedef std::shared_ptr<class Medicines> pMedicines;


class Medicines: public Object {
private:
    int healthRecovery_;

public:
    explicit Medicines(int healthRecovery);
    pObject getCopy() const;
    void executeInteraction(Creature& creature);
    const std::type_info& getClass() const;
    Texture getTexture() const;
    bool getReusable() const;
};


#endif // MEDICINES_H
```

Название файла: medicines.cpp

```
#include "medicines.h"
#include "medicinesfactory.h"

Medicines::Medicines(int healthRecovery):
healthRecovery_(healthRecovery) {}

pObject Medicines::getCopy() const {
    pMedicinesFactory factory(new MedicinesFactory);
    return pObject(factory->createMedicines(healthRecovery_));
}

void Medicines::executeInteraction(Creature& creature) {
    creature.setHealth(creature.getHealth() + healthRecovery_);
}

const std::type_info& Medicines::getClass() const {
    return typeid(Medicines);
}

Texture Medicines::getTexture() const {
    return kTextureObjectMedicines;
}

bool Medicines::getReusable() const {
    return false;
}
```

Название файла: medicinesfactory.h

```
#ifndef MEDICINES_FACTORY_H
#define MEDICINES_FACTORY_H

#include "objectfactory.h"
#include "medicines.h"

typedef std::shared_ptr<class MedicinesFactory> pMedicinesFactory;


class MedicinesFactory: public ObjectFactory {
public:
    virtual pObject createObject();
    virtual pObject createMedicines(int healthRecovery);
};


#endif // MEDICINES_FACTORY_H
```

Название файла: medicinesfactory.cpp

```
#include "medicinesfactory.h"

pObject MedicinesFactory::createObject() {
    return pObject(new Medicines(20));
}

pObject MedicinesFactory::createMedicines(int healthRecovery) {
    return pObject(new Medicines(healthRecovery));
}
```

Название файла: object.h

```
#ifndef OBJECT_H
#define OBJECT_H

#include <typeinfo>
#include <memory>
#include "texture.h"
#include "creature.h"

typedef std::shared_ptr<class Object> pObject;
typedef std::shared_ptr<const class Object> pConstObject;
typedef std::shared_ptr<class Creature> pCreature;


class Object {
public:
    virtual pObject getCopy() const = 0;
    virtual const std::type_info& getClass() const = 0;
    virtual Texture getTexture() const = 0;
    virtual void executeInteraction(Creature& creature) = 0;
    virtual bool getReusable() const = 0;
    virtual ~Object() = default;
};


#endif // OBJECT_H
```

Название файла: objectfactory.h

```
#ifndef OBJECT_FACTORY_H
#define OBJECT_FACTORY_H

#include "object.h"

typedef std::shared_ptr<class ObjectFactory> pObjectFactory;


class ObjectFactory {
public:
    virtual pObject createObject() = 0;
    virtual ~ObjectFactory() = default;
};


#endif // OBJECT_FACTORY_H
```

Название файла: player.h

```
#ifndef PLAYER_H
#define PLAYER_H

#include <memory>
#include "creature.h"

typedef std::shared_ptr<class Player> pPlayer;


class Player: public Creature {
private:
    bool passFounded_ = false;
    pInteractionStrategy objectInteractionStrategy_;

public:
    Player(Position2D position);
    void interact(pObject& object);
    Texture getTexture() const;
    void operator<=(pObject& object);
    bool getPassFounded() const;
    void setPassFounded(bool value);
};


#endif // PLAYER_H
```

Название файла: player.cpp

```
#include "player.h"
#include "interactionuse.h"
#include "interactionnone.h"
#include <iostream>

Player::Player(Position2D position) {
    objectInteractionStrategy_ = pInteractionStrategy(new
InteractionUse);
    setPosition(position);
}
```

```
      const std::type_info &Player::getClass() const {
          return typeid(Player);
      }

      void Player::operator<=(pObject& object) {
          if (objectInteractionStrategy_ != nullptr) {
              if (object != nullptr) {
                  eventManager.notify("Object of class 'Player' interact with
object of class '" + std::string(object->getClass().name()) + "'\n");
              }
              objectInteractionStrategy_->interact(*this, object);
          }
      }

      bool Player::getPassFounded() const {
          return passFounded_;
      }

      void Player::setPassFounded(bool value) {
          passFounded_ = value;
      }

      void Player::changeInteraction(pInteractionStrategy
objectInteractionStrategy) {
          objectInteractionStrategy_ = objectInteractionStrategy;
      }

      void Player::setRotation(Rotation rotation) {
          if (rotation != getRotation()) {
              Creature::setRotation(rotation);
              std::string directionName;

              switch (rotation) {
              case kDirectionTop:
                  directionName = "Top";
                  break;
              case kDirectionBottom:
                  directionName = "Bottom";
                  break;
              case kDirectionLeft:
                  directionName = "Left";
                  break;
              case kDirectionRight:
                  directionName = "Right";
                  break;
              };

              eventManager.notify("Object of class 'Player' change rotation to
'" + directionName + "'\n");
          }
      }

      void Player::setPosition(Position2D position) {
          if (position != getPosition()) {
              Creature::setPosition(position);
              eventManager.notify("Object of class 'Player' change position to
[" + std::to_string(getPosition().x) + ", " + std::to_string(getPosition().y)
+ "]\n");
          }
      }
```

```
void Player::setHealth(int health) {
    if (health != getHealth()) {
        Creature::setHealth(health);
        eventManager.notify("Object of class 'Player' change health to " + std::to_string(getHealth()) + "\n");
    }
}

void Player::setMaxHealth(int maxHealth) {
    if (maxHealth != getMaxHealth()) {
        Creature::setMaxHealth(maxHealth);
        eventManager.notify("Object of class 'Player' change maximum health to " + std::to_string(getMaxHealth()) + "\n");
    }
}

void Player::setAttackDamage(int damage) {
    if (damage != getAttackDamage()) {
        Creature::setAttackDamage(damage);
        eventManager.notify("Object of class 'Player' change attack damage to " + std::to_string(getAttackDamage()) + "\n");
    }
}

void Player::setProtection(int protection) {
    if (protection != getProtection()) {
        Creature::setProtection(protection);
        eventManager.notify("Object of class 'Player' change protection to " + std::to_string(getProtection()) + "\n");
    }
}

std::ostream& operator<<(std::ostream& stream, const Player& player) {
    stream << "Object of class 'Player': Position(" << player.getPosition() << "); Health(" << player.getHealth() << "); MaxHealth("
           << player.getMaxHealth() << "); AttackDamage(" << player.getAttackDamage() << "); Protection(" << player.getProtection()
           << "); Rotation(" << player.getRotation() << "); PassFounded(" << player.getPassFounded() << ")\n";
    return stream;
}
```

Название файла: point2d.h

```
#ifndef POINT_2D_H
#define POINT_2D_H

#include <cstddef>
#include "direction.h"

typedef struct Point2D Size2D;
typedef struct Point2D Position2D;


struct Point2D {
public:
    size_t x = 0;
    size_t y = 0;
    Point2D() = default;
    Point2D(size_t x, size_t y);
    bool operator==(const Point2D& other) const;
    bool operator!=(const Point2D& other) const;
    void shift(Direction direction);
};


#endif // POINT_2D_H
```

Название файла: point2d.cpp

```
#include "point2d.h"

Point2D::Point2D(size_t x, size_t y): x(x), y(y) {}

bool Point2D::operator==(const Point2D& other) const {
    return x == other.x && y ==other.y;
}

bool Point2D::operator!=(const Point2D& other) const {
    return !operator==(other);
}

void Point2D::shift(Direction direction) {
    switch (direction) {
    case kDirectionTop:
        y--;
        return;
    case kDirectionBottom:
        y++;
        return;
    case kDirectionLeft:
        x--;
        return;
    case kDirectionRight:
        x++;
        return;
    }
}
```

Название файла: texture.h

```
#ifndef TEXTURE_H
#define TEXTURE_H

enum Texture {
    kTextureVoid,
    kTextureWoodFloor1,
    kTextureWoodWall1,
    kTextureWoodWall2,
    kTextureWoodWall3,
    kTextureWoodWall4,
    kTextureWoodWall5,
    kTextureWoodWall6,
    kTextureWoodWall7,
    kTextureWoodWall8,
    kTextureWoodWall9,
    kTextureWoodWall10,
    kTextureWoodWall11,
    kTextureWoodWall12,
    kTextureWoodWall13,
    kTextureEntry,
    kTextureExit,
    kTextureShadow1,
    kTextureShadow2,
    kTextureShadow3,
    kTextureShadow4,
    kTextureCell,
    kTexturePlayerStandT,
    kTexturePlayerStandB,
    kTexturePlayerStandR,
    kTexturePlayerStandL,
    kTextureObjectMedicines,
    kTextureObjectArmor,
    kTextureObjectWeapon,
    kTextureObjectLevelPass
};

#endif // TEXTURE_H
```

Название файла: weapon.h

```
#ifndef WEAPON_H
#define WEAPON_H

#include "memory"
#include "object.h"

typedef std::shared_ptr<class Armor> pArmor;


class Weapon: public Object {
private:
    int damage_;

public:
    explicit Weapon(int damage);
    pObject getCopy() const;
    void executeInteraction(Creature& creature);
    const std::type_info& getClass() const;
    bool getReusable() const;
    ~Weapon();

    friend std::ostream& operator<<(std::ostream& stream, const Weapon&
weapon);
};


#endif // WEAPON_H
```

Название файла: weapon.cpp

```
#include "weapon.h"
#include "weaponfactory.h"

Weapon::Weapon(int damage): damage_(damage) {}
```

```
pObject Weapon::getCopy() const {
    pWeaponFactory factory(new WeaponFactory);
    return pObject(factory->createWeapon(damage_));
}

void Weapon::executeInteraction(Creature& creature) {
    if (creature.getAttackDamage() < damage_) {
        creature.setAttackDamage(damage_);
    }
}

const std::type_info &Weapon::getClass() const {
    return typeid(Weapon);
}

bool Weapon::getReusable() const {
    return false;
}

Weapon::~Weapon() {
    eventManager.notify("Destroying object of class 'Weapon'.\n");
}

std::ostream& operator<<(std::ostream& stream, const Weapon& weapon) {
    stream << "Object of class 'Weapon': Damage(" << weapon.damage_ <<
")\n";
    return stream;
}
```

Название файла: weaponfactory.h

```
#ifndef WEAPON_FACTORY_H
#define WEAPON_FACTORY_H

#include "objectfactory.h"
#include "weapon.h"

typedef std::shared_ptr<class WeaponFactory> pWeaponFactory;


class WeaponFactory: public ObjectFactory {
public:
    virtual pObject createObject();
    virtual pObject createWeapon(int damage);
};


#endif // WEAPON_FACTORY_H
```

Название файла: weaponfactory.cpp

```
#include "weaponfactory.h"

pObject WeaponFactory::createObject() {
    return pObject(new Weapon(5));
}

pObject WeaponFactory::createWeapon(int damage) {
    return pObject(new Weapon(damage));
}
```

Название файла: logger.h

```
#ifndef LOGGER_H
#define LOGGER_H

#include <memory>
#include <string>

typedef std::shared_ptr<class ILoggerImplementation>
pILoggerImplementation;
typedef std::shared_ptr<class Logger> pLogger;


class Logger {
protected:
    pILoggerImplementation implementation_;

public:
    Logger(const pILoggerImplementation& implementation);
    virtual void log(const std::string& message) = 0;
    virtual ~Logger() = default;
};


#endif // LOGGER_H
```

Название файла: logger.cpp

```
#include "filelogger.h"
#include "loggerimplementation.h"

FileLogger::FileLogger(): Logger(pILoggerImplementation(new
LoggerImplementation)) {}

FileLogger::FileLogger(const std::string& filepath):
Logger(pILoggerImplementation(new LoggerImplementation)) {
    filepath_ = filepath;
    file_.open(filepath);
}

void FileLogger::setFile(const std::string &filepath) {
    file_.close();
    filepath_ = filepath;
    file_.open(filepath);
}

void FileLogger::log(const std::string& message) {
    implementation_->fileLog(file_, message);
}
```

Название файла: filelogger.h

```
#ifndef FILE_LOGGER_H
#define FILE_LOGGER_H

#include "logger.h"
```

```
#include "fstream"


class FileLogger: public Logger {
private:
    std::string filepath_;
    std::ofstream file_;

public:
    FileLogger();
    explicit FileLogger(const std::string& filepath);
    void setFile(const std::string& filepath);
    void log(const std::string& message);
};


#endif // FILE_LOGGER_H
```

Название файла: filelogger.cpp

```
#include "filelogger.h"
#include "loggerimplementation.h"

FileLogger::FileLogger(): Logger(pILoggerImplementation(new
LoggerImplementation)) {}

FileLogger::FileLogger(const std::string& filepath):
Logger(pILoggerImplementation(new LoggerImplementation)) {
    filepath_ = filepath;
    file_.open(filepath);
}

void FileLogger::setFile(const std::string &filepath) {
    file_.close();
    filepath_ = filepath;
    file_.open(filepath);
}

void FileLogger::log(const std::string& message) {
    implementation_->fileLog(file_, message);
}
```

Название файла: consolelogger.h

```
#ifndef CONSOLE_LOGGER_H
#define CONSOLE_LOGGER_H

#include "logger.h"


class ConsoleLogger: public Logger {
private:
    std::ostream& stream_;

public:
    ConsoleLogger(std::ostream& stream);
    void log(const std::string& message);
};
```

```
#endif // CONSOLE_LOGGER_H
```

Название файла: consolelogger.cpp

```
#include "consolelogger.h"
#include "loggerimplementation.h"

ConsoleLogger::ConsoleLogger(std::ostream& stream):
Logger(pILoggerImplementation(new LoggerImplementation)), stream_(stream) {}

void ConsoleLogger::log(const std::string& message) {
    implementation_->consoleLog(stream_, message);
}
```

Название файла: eventmanager.h

```
#ifndef EVENT_MANAGER_H
#define EVENT_MANAGER_H

#include <set>
#include "eventlistener.h"


class EventManager {
private:
    std::set<pEventListener> listeners;

public:
    void subscribe(pEventListener& listener);
    void unsubscribe(pEventListener& listener);
    void notify(const std::string& message);
};


#endif // EVENT_MANAGER_H
```

Название файла: eventmanager.cpp

```
#include "eventmanager.h"

void EventManager::subscribe(pEventListener& listener) {
    listeners.insert(listener);
}

void EventManager::unsubscribe(pEventListener& listener) {
    listeners.erase(listener);
}

void EventManager::notify(const std::string& message) {
    for (auto listener : listeners) {
        listener->update(message);
    }
}
```

Название файла: eventlistener.h

```
#ifndef EVENT_LISTENER_H
#define EVENT_LISTENER_H

#include <memory>

typedef std::shared_ptr<class EventListener> pEventListener;


class EventListener {
public:
    virtual void update(const std::string& message) = 0;
};


#endif // EVENT_LISTENER_H
```

Название файла: logginglistener.h

```
#ifndef LOGGINGLISTENER_H
#define LOGGINGLISTENER_H

#include <iostream>
#include <vector>
#include "eventlistener.h"
#include "consolelogger.h"
#include "filelogger.h"

typedef std::shared_ptr<class LoggingListener> pLoggingListener;


class LoggingListener: public EventListener {
private:
    pLogger consoleLogger_;
    pLogger fileLogger_;

public:
    LoggingListener() = default;
    LoggingListener(const pLogger& consoleLogger, const pLogger&
fileLogger);
    void setConsoleLogger(const pLogger& consoleLogger);
    void setFileLogger(const pLogger& fileLogger);
    void update(const std::string& message);
};

#endif // LOGGINGLISTENER_H
```

Название файла: logginglistener.cpp

```
#include "logginglistener.h"

LoggingListener::LoggingListener(const pLogger& consoleLogger, const
pLogger& fileLogger) {
    consoleLogger_ = consoleLogger;
    fileLogger_ = fileLogger;
}

void LoggingListener::setConsoleLogger(const pLogger& consoleLogger) {
    consoleLogger_ = consoleLogger;
}
```

```
void LoggingListener::setFileLogger(const pLogger& fileLogger) {
    fileLogger_ = fileLogger;
}

void LoggingListener::update(const std::string& message) {
    if (consoleLogger_ != nullptr) {
        consoleLogger_->log(message);
    }

    if (fileLogger_ != nullptr) {
        fileLogger_->log(message);
    }
}
```

Название файла: iloggerimplementation.h

```
#ifndef I_LOGGER_IMPLEMENTATION_H
#define I_LOGGER_IMPLEMENTATION_H

#include <string>
#include <fstream>


class ILoggerImplementation {
public:
    virtual void consoleLog(std::ostream& stream, const std::string&
message) = 0;
    virtual void fileLog(std::ofstream& file, const std::string&
message) = 0;
    virtual ~ILoggerImplementation() = default;
};


#endif // I_LOGGER_IMPLEMENTATION_H
```

Название файла: loggerimplementation.h

```
#ifndef LOGGER_IMPLEMENTATION_H
#define LOGGER_IMPLEMENTATION_H

#include <ctime>
#include "iloggerimplementation.h"


class LoggerImplementation: public ILoggerImplementation {
private:
    std::string getCurrentDateTime();

public:
    void consoleLog(std::ostream& stream, const std::string& message);
    void fileLog(std::ofstream& file, const std::string& message);
};


#endif // LOGGER_IMPLEMENTATION_H
```

Название файла: loggerimplementation.cpp

```
#include "loggerimplementation.h"
#include <iostream>
#include <fstream>
#include <ctime>

void LoggerImplementation::consoleLog(std::ostream& stream, const std::string& message) {
    stream << getCurrentDateTime() << message;
}

std::string LoggerImplementation::getCurrentDateTime() {
    char buffer[25] = {'\0'};
    time_t timestamp = time(nullptr);
    tm* timeinfo = localtime(&timestamp);

    if (timeinfo == nullptr) {
        sprintf(buffer, "[00-00-00 00:00:00] ");
    } else {
        strftime(buffer, 25, "[%d-%m-%y %H:%M:%S] ", timeinfo);
    }

    return std::string(buffer);
}

void LoggerImplementation::fileLog(std::ofstream& file, const std::string& message) {
    if (file.is_open()) {
        file << getCurrentDateTime() << message;
    }
}
```