

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

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: Шаблонные классы, управление.**

Студентка гр. 0382

\_\_\_\_\_

Кривенцова Л.С.

Преподаватель

\_\_\_\_\_

Жангиров Т.Р.

Санкт-Петербург

2021

## **Цель работы.**

Изучить понятие шаблон (рассмотреть шаблонные классы и функций), сделать с их помощью программу более гибкой, реализуя в ней обобщённые алгоритмы.

## **Задание.**

Необходимо определить набор правил для игры в виде классов (например, какие задачи необходимо выполнить, чтобы он мог выйти с поля; какое кол-во врагов и вещей должно быть на поле, и.т.д.). Затем определить класс игры, которое параметризуется правилами. Класс игры должен быть прослойкой между бизнес-логикой и командами управления, то есть непосредственное изменение состояния игрой должно проходить через этот класс.

Требование:

- Созданы шаблонные классы правил игры. В данном случае параметр шаблона должен определить конкретные значения в правилах.
- Создан шаблонный класс игры, который параметризуется конкретными правилами. Класс игры должен проводить управление врагами, передачей хода, передавать информацию куда переместить игрока, и.т.д.

## **Основные теоретические положения.**

Шаблоны.

- Шаблоны— средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).
- Шаблоны приводят к парадигме обобщенного программирования (метапрограммирования).

Особенности шаблонов.

- Обеспечивают статический полиморфизм.
- Параметры шаблонов – типы и константы времени компиляции.

- Не замедляют скорость работы программы.
- Не требует связи типов между собой (иерархии).

Инстанцирование.

- Шаблоны не компилируются до инстанцирования.
- Сгенерированный шаблон – полноценный класс или функция.
- Шаблоны необходимо определять в header файлах.

Несколько параметров шаблона.

- Количество задаваемых параметров не ограничено.
- Недостаток, возвращаемый тип зависит от аргумента на определенной позиции.

Аргументы шаблонов по умолчанию.

- Можно определять значения параметров шаблонов по умолчанию, которые будут использовать, если явно не указаны. Выполнение работы.

**Ход решения:**

Для создания шаблонов не требуется подключение никаких новых заголовочных файлов стандартной библиотеки.

1. Были созданы интерфейсы для классов правил игры:

a) Класс *NumberOfEnemiesInterface* – класс правил, задающий количество на игровом поле врагов определенной категории.

Объявлены виртуальные методы с модификатором доступа public:

*virtual int GetScout() = 0;* - Геттер для получения значения количества мелких врагов.

*virtual int GetMcount() = 0;* - Геттер для получения значения количества средних врагов.

*virtual int GetLcount() = 0;* - Геттер для получения значения количества сложных врагов.

b) Класс *NumberOfItemsInterface* – класс правил, задающий количество на

игровом поле вещей определенной категории.

Объявлены виртуальные методы с модификатором доступа public:

*virtual int GetBoxCount() = 0;* - Геттер для получения значения количества сундуков на поле.

*virtual int GetHealCount() = 0;* - Геттер для получения значения количества аптек на поле.

*virtual int GetPortalCount() = 0;* - Геттер для получения значения количества порталов на поле.

с) Класс *RuleOfFieldInterface* – класс правил, задающий внешний вид поля (его размер и шанс появления стены на пустой клетке).

Объявлены виртуальные методы с модификатором доступа public:

*virtual int GetHeight() = 0;* - Геттер для получения значения высоты поля.

*virtual int GetWidth() = 0;* - Геттер для получения значения ширины поля.

*virtual int GetSize() = 0;* - Геттер для получения значения площади игрового поля.

*virtual int GetWallChance() = 0;* - Геттер для получения значения шанса появления стены на пустой клетке порталов.

d) Класс *RuleOfWinInterface* – класс правил, задающий условия выхода с поля (нужен ли ключ, и минимальное количество здоровья и денег, которыми должен обладать персонаж).

Объявлены виртуальные методы с модификатором доступа public:

*virtual bool KeyNeeded() = 0;* - Геттер для получения логического значения, обозначающего требуется ли наличие ключа от выхода для победы в игре.

*virtual int MinimumHealthNeeded() = 0;* - Геттер для получения минимального значения здоровья, которым должен обладать игрок для победы.

*virtual int MinimumMoneyNeeded() = 0;* - Геттер для получения значения минимального количества монет, которым должен обладать игрок для победы.

Смысл интерфейсов в том, чтобы от них могли наследоваться не только шаблонные классы (описанные ниже), которые принимают значения в качестве параметров шаблона, но и другие классы правил количества предметов или

условий игры. Например, правила, принимающие нужные значения из потока ввода (файла или консоли). При этом методы получения этих значений классами извне (геттеры) останутся теми же (описанными в интерфейсе) и будут обязательны к реализации, что избавит от обращения к несуществующим полям в классе *Game*.

2. Были созданы следующие шаблонные классы правил игры – классы-наследники интерфейсов, описанных выше:

а) Шаблонный класс канонного правила игры, задающего количество врагов на поле.

```
template <int s, int m, int l>
```

```
class CanonNumberOfEnemies: NumberOfEnemiesInterface.
```

Класс – реализация интерфейса *NumberOfEnemiesInterface*.

Имеет три поля с модификатором доступа *private*:

```
int number_of_s; - количество мелких врагов;
```

```
int number_of_m; - количество средних врагов;
```

```
int number_of_l; - количество сложных врагов;
```

Кроме геттеров, описанных в интерфейсе, служащих для доступа к этим полям, переопределяется конструктор:

```
CanonNumberOfEnemies(int size_of_field = 150); - получает на вход  
площадь игрового поля (или использует значение по умолчанию). В конструкторе  
параметры шаблона ( s – количество мелких врагов в процентах, m– количество  
средних врагов в процентах, l– количество сложных врагов) служат для  
вычислений реального количества врагов на поле (рассчитываются относительно  
полученных размеров поля), которые присваиваются полям класса (таким образом  
в конструкторе происходит инициализация полей).
```

б) Шаблонный класс канонного правила игры, задающего вещей на поле.

```
template <int boxes, int heals, int teleports>
```

```
class CanonNumberOfItems : NumberOfItemsInterface.
```

Класс – реализация интерфейса *NumberOfItemsInterface*.

Имеет три поля с модификатором доступа *private*:

*int number\_of\_boxes;* - количество сундуков;  
*int number\_of\_heals;* - количество аптек;  
*int number\_of\_teleports;* - количество телепортов;

Кроме геттеров, описанных в интерфейсе, служащих для доступа к этим полям, переопределяется конструктор:

*CanonNumberOfItems(int size\_of\_field = 150);* - получает на вход площадь игрового поля (или использует значение по умолчанию). В конструкторе параметры шаблона (*boxes* – количество сундуков в процентах, *heals* – количество аптек в процентах, *teleports* – количество телепортов) служат для вычислений реального количества вещей на поле (рассчитываются относительно полученных размеров поля), которые присваиваются полям класса (таким образом в конструкторе происходит инициализация полей).

с) Шаблонный класс канонного правила игры, задающего размер поля и шанс появления непроходимой клетки.

*template <int h, int w, int chance>*  
*class CanonRuleOfField: RuleOfFieldInterface.*

Класс – реализация интерфейса *RuleOfFieldInterface*.

Имеет три поля с модификатором доступа *private*:

*int height;* - высота поля;  
*int width;* - ширина поля;  
*int advent\_of\_wall\_chance;* - шанс возникновения стены на ранее проходимой клетке.

Кроме геттеров, описанных в интерфейсе, служащих для доступа к этим полям, переопределяется конструктор:

*CanonRuleOfField();* - не принимает аргументов. В конструкторе параметры шаблона (*h*– высота поля, *w* – ширина поля, *chance* – шанс появления непроходимой клетки) служат для инициализации полей класса.

d) Шаблонный класс канонного правила игры, задающего условия выхода игрока с поля.

*template <bool ky, int hlth, int mns>*

*class CanonRuleOfWin: RuleOfWinInterface.*

Класс – реализация интерфейса *RuleOfWinInterface*.

Имеет три поля с модификатором доступа *private*:

*bool key;* - наличие ключа;

*int health;* - минимальное количество здоровья;

*int moneys;* - минимальное количество монет.

Кроме геттеров, описанных в интерфейсе, служащих для доступа к этим полям, переопределяется конструктор:

*CanonRuleOfWin ()*; - не принимает аргументов. В конструкторе параметры шаблона (*ky*– наличие ключа, *hlth* – количество здоровья, *mns* – количество монет) служат для инициализации полей класса.

3. Были изменены некоторые классы. Например, Класс *Game* стал шаблонным:

*template <typename T1 = CanonRuleOfField<10,15,10>, typename T2 = CanonNumberOfEnemies<2, 1, 1>, typename T3 = CanonNumberOfItems<2, 2, 1>, typename T4 = CanonRuleOfWin<true,0,0>>*

*class Game: public IObservable*

В отличие от шаблонных классов правил, здесь параметр шаблона это не значение, а тип переменной.

Здесь используются параметры по умолчанию. Так что если не передать в качестве параметров классы правил игры (или передать частично), то игра запустится по классическим правилам.

Следовательно, были добавлены поля правил игры (с модификатором доступа *private*):

*T1 field\_rules;*

*T2 number\_of\_enemies;*

*T3 number\_of\_items;*

*T4 win\_rules;*

Которые инициализируются в конструкторе (создаётся объект класса каждого правила и присваивается полю класса).

При создании объекта поля теперь в конструктор передаются правила из класса *field\_rules*:

```
Field game1( field_rules.GetWidth(), field_rules.GetHeight());
```

Следующие изменения в классе *Game* коснулись вызова методов класса *Field* (методов создания вещей, врагов, стен поля). Теперь в них также передаются правила игры (значение полей классов правил игры, полученных с помощью геттера):

- *game.MakeObjects(field\_rules.GetWallChance());*
- *game.MakeItems(number\_of\_items.GetPortalCount(),  
number\_of\_items.GetBoxCount(), number\_of\_items.GetHealCount());*
- *game.MakeEnemies(number\_of\_enemies.GetScount(),  
number\_of\_enemies.GetMcount(), number\_of\_enemies.GetLcount());*

Соответственно, эти методы класса *Field* также были изменены: теперь значения, которые раньше генерировались (или хранились) в теле методов - принимаются их конструкторами в качестве аргументов.

Также в классе *Game* был частично изменён метод *GameMove()*:

Если игрок оказывается на клетке выхода, то с помощью *if* проверяется, соответствуют ли поля игрока правилам выхода с поля. Если нет, то выводится предупреждение о несоблюдение определённого условия. Если правила соблюдены, то выводится сообщение об окончании игры.

```
ExitDoorPrint door;
```

```
if (person.GetKey() < win_rules.KeyNeeded()) {  
    door.KeyPrint();  
} else if (person.GetCoins() < win_rules.MinimumMoneyNeeded()) {  
    door.CoinsPrint();  
} else if (person.GetHealth() < win_rules.MinimumHealthNeeded()) {  
    door.HealthPrint();  
} else {  
    door.OpenPrint();  
}
```

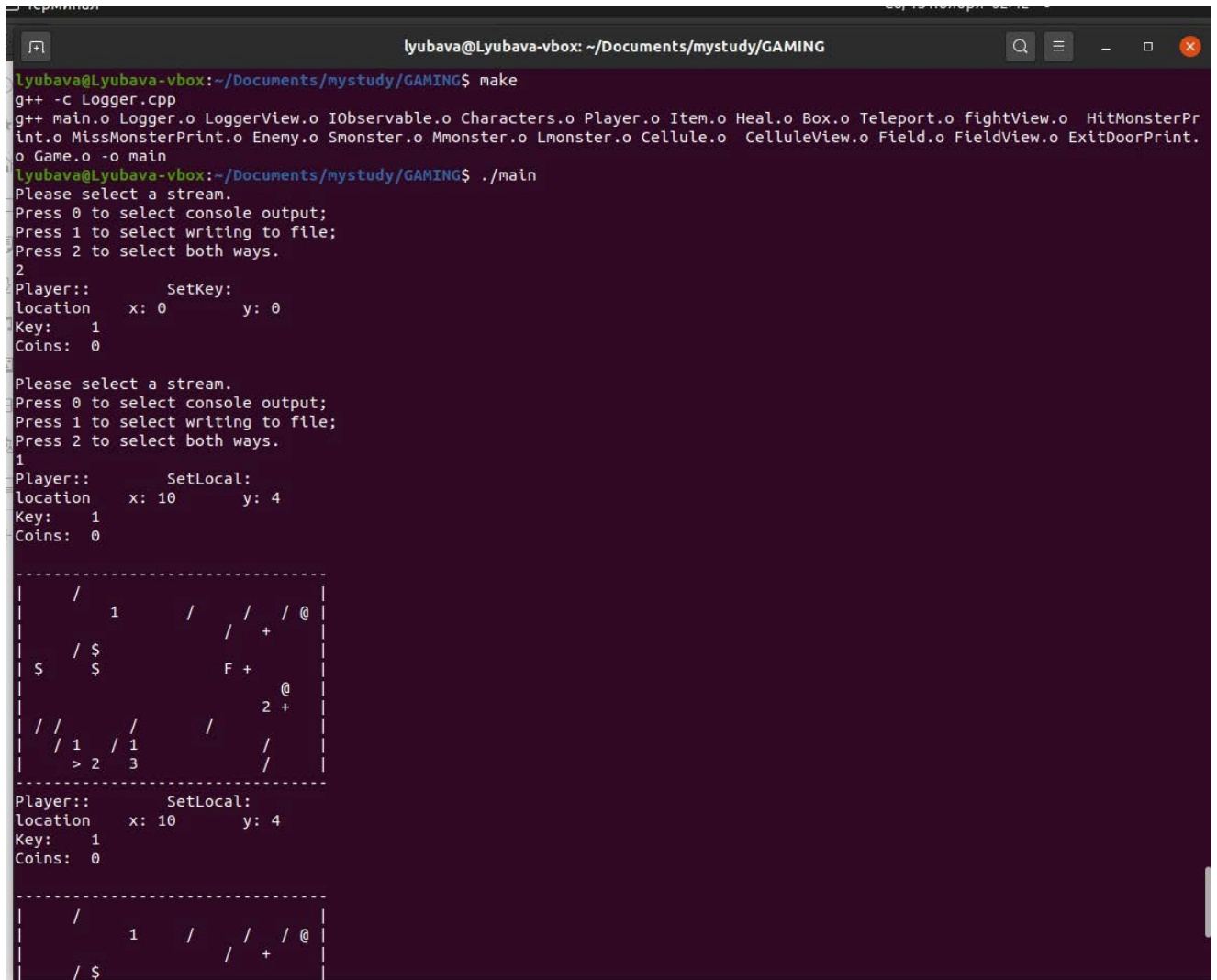


Для этого был немного модифицирован класс сообщений при выходе *ExitDoorPrint*. Теперь у него есть 4 метода с различными сообщениями, указывающими на состояние игрока и игры.

### Результат работы программы.

Никак не изменился.

Рис 1. – демонстрация работы программы с логированием в терминал Ubuntu.



```
lyubava@Lyubava-vbox: ~/Documents/mystudy/GAMING
lyubava@Lyubava-vbox:~/Documents/mystudy/GAMING$ make
g++ -c Logger.cpp
g++ main.o Logger.o LoggerView.o IObservable.o Characters.o Player.o Item.o Heal.o Box.o Teleport.o fightView.o HitMonsterPrint.o MissMonsterPrint.o Enemy.o Smonster.o Mmonster.o Lmonster.o Cellule.o CelluleView.o Field.o FieldView.o ExitDoorPrint.o Game.o -o main
lyubava@Lyubava-vbox:~/Documents/mystudy/GAMING$ ./main
Please select a stream.
Press 0 to select console output;
Press 1 to select writing to file;
Press 2 to select both ways.
2
Player:: SetKey:
location x: 0 y: 0
Key: 1
Coins: 0

Please select a stream.
Press 0 to select console output;
Press 1 to select writing to file;
Press 2 to select both ways.
1
Player:: SetLocal:
location x: 10 y: 4
Key: 1
Coins: 0

-----
|      /      1      /      /      /      @      |
|      / $      /      /      +      |
| $      $      F +      @      |
|      /      /      /      /      2 +      |
| / /      /      /      /      /      |
| / 1      / 1      /      /      |
| > 2      3      /      /      |
|-----|
Player:: SetLocal:
location x: 10 y: 4
Key: 1
Coins: 0

-----
|      /      1      /      /      /      @      |
|      / $      /      /      +      |
```

### **UML-диаграмма межклассовых отношений.**

Находится в PDF-файле. Формат документов Word не позволяет вставить сюда копию, сохраняя читаемость и разборчивость диаграммы.

### **Выводы.**

Было изучено понятие шаблона, рассмотрены (и реализованы) шаблонные классы и функции, получены навыки реализовывать в программе обобщённые алгоритмы.