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

ОТЧЕТ

по курсовой работе (Лаб. 1 – 4)
по дисциплине «Программирование»
Тема: Основы ООП. Введение в паттерны проектирования

Студентка гр. 0324 _____ Жигалова Д. А.

Преподаватель _____ Глущенко А. Г.

Санкт-Петербург
2021

Цель работы.

Пройти от написания простых конкретных классов к профессиональному конструированию приложения в объектно-ориентированной парадигме. Научиться писать красиво и элегантно, использовать проверенные временем концепции и создавать масштабируемые программы, так как использование паттернов проектирования является признаком профессионализма программиста.

Основные теоретические положения.

В окончательном виде любая программа представляет собой набор инструкций процессора. Все, что написано на любом языке программирования, — более удобная, упрощенная запись этого набора инструкций, облегчающая написание, отладку и последующую модификацию программы. Чем выше уровень языка, тем в более простой форме записываются одни и те же действия.

С ростом объема программы становится невозможным удерживать в памяти все детали, и становится необходимым структурировать информацию, выделять главное и отбрасывать несущественное. Этот процесс называется повышением степени абстракции программы.

Первым шагом к повышению абстракции является использование функций, позволяющее после написания и отладки функции отвлечься от деталей ее реализации, поскольку для вызова функции требуется знать только ее интерфейс. Если глобальные переменные не используются, интерфейс полностью определяется заголовком функции.

Следующий шаг — описание собственных типов данных, позволяющих структурировать и группировать информацию, представляя ее в более естественном виде. Например, можно представить с помощью одной структуры все разнородные сведения, относящиеся к одному виду товара на складе.

Для работы с собственными типами данных требуются специальные функции. Естественно, сгруппировать их с описанием этих типов данных в одном месте программы, а также по возможности отделить от ее остальных

частей. При этом для использования этих типов и функций не требуется полного знания того, как именно они написаны — необходимы только описания интерфейсов. Объединение в модули описаний типов данных и функций, предназначенных для работы с ними, со скрытием от пользователя модуля несущественных деталей, является дальнейшим развитием структуризации программы.

Все три описанных выше метода повышения абстракции преследуют цель упростить структуру программы, то есть представить ее в виде меньшего количества более крупных блоков и минимизировать связи между ними. Это позволяет управлять большим объемом информации и, следовательно, успешно отлаживать более сложные программы.

Введение понятия класса является естественным развитием идей модульности. В классе структуры данных и функции их обработки объединяются. Класс используется только через его интерфейс — детали реализации для пользователя класса несущественны. Идея классов отражает строение объектов реального мира — ведь каждый предмет или процесс обладает набором характеристик или отличительных черт, иными словами, свойствами и поведением. Программы часто предназначены для моделирования предметов, процессов и явлений реального мира поэтому в языке программирования удобно иметь адекватный инструмент для представления моделей.

Существенным свойством класса является то, что детали его реализации скрыты от пользователей класса за интерфейсом (ведь и в реальном мире можно, например, управлять автомобилем, не имея представления о принципе внутреннего сгорания и устройстве двигателя, а пользоваться телефоном — не зная, «как идет сигнал, принципов связи и кто клал кабель). Интерфейсом класса являются заголовки его методов. Таким образом, класс как модель объекта реального мира является черным ящиком, замкнутым по отношению к внешнему миру.

Идея классов является основой объектно-ориентированного программирования (ООП). Основные принципы ООП были разработаны еще в языках Simula-67 и Smalltalk, но в то время не получили широкого применения из-за трудностей освоения и низкой эффективности реализации. В C++ эти концепции реализованы эффективно, красиво и непротиворечиво, что и явилось основой успешного распространения этого языка и внедрения подобных средств в другие языки программирования.

ООП — это не просто набор новых средств, добавленных в язык (на C++ можно успешно писать и без использования ООП, и наоборот, возможно написать объектную, по сути, программу на языке, не содержащим специальных средств поддержки объектов), ООП часто называют новой парадигмой программирования.

При создании программных систем перед разработчиками часто встает проблема выбора тех или иных проектных решений. В этих случаях на помощь приходят паттерны. Дело в том, что почти наверняка подобные задачи уже решались ранее и уже существуют хорошо продуманные элегантные решения, составленные экспертами. Если эти решения описать и систематизировать в каталоги, то они станут доступными менее опытным разработчикам, которые после изучения смогут использовать их как шаблоны или образцы для решения задач подобного класса. Паттерны как раз описывают решения таких повторяющихся задач.

Постановка задачи.

Задание на разработку №1

Разработать и реализовать набор классов:

- Класс игрового поля
- Набор классов юнитов

Игровое поле является контейнером для объектов, представляющим прямоугольную сетку. Основные требования к классу игрового поля:

- Создание поля произвольного размера

- Контроль максимального количества объектов на поле
- Возможность добавления и удаления объектов на поле
- Возможность копирования поля (включая объекты на нем)

Юнит является объектов, размещаемым на поля боя. Один юнит представляет собой отряд. Основные требования к классам юнитов:

- Все юниты должны иметь как минимум один общий интерфейс
- Реализованы 3 типа юнитов (например, пехота, лучники, конница)
- Реализованы 2 вида юнитов для каждого типа (например, для пехоты могут быть созданы мечники и копейщики)
- Юниты имеют характеристики, отражающие их основные атрибуты, такие как здоровье, броня, атака.
- Юнит имеет возможность перемещаться по карте

Задание на разработку №2

Разработать и реализовать набор классов:

- Класс базы
- Набор классов ландшафта карты
- Набор классов нейтральных объектов поля

Класс базы должен отвечать за создание юнитов, а также учитывать юнитов, относящихся к текущей базе. Основные требования к классу база:

- База должна размещаться на поле
- Методы для создания юнитов
- Учет юнитов, и реакция на их уничтожение и создание
- База должна обладать характеристиками такими, как здоровье, максимальное количество юнитов, которые могут быть одновременно созданы на базе, и.т.д.

Набор классов ландшафта определяют вид поля. Основные требования к классам ландшафта:

Должно быть создано минимум 3 типа ландшафта

- Все классы ландшафта должны иметь как минимум один интерфейс

- Ландшафт должен влиять на юниты (например, возможно пройти по клетке с определенным ландшафтом или запрет для атаки определенного типа юнитов)
- На каждой клетке поля должен быть определенный тип ландшафта

Набор классов нейтральных объектов представляют объекты, располагаемые на поле и с которыми могут взаимодействие юнитов. Основные требования к классам нейтральных объектов поля:

- Создано не менее 4 типов нейтральных объектов
- Взаимодействие юнитов с нейтральными объектами, должно быть реализовано в виде перегрузки операций
- Классы нейтральных объектов должны иметь как минимум один общий интерфейс

Задание на разработку №3

Разработать и реализовать набора классов для взаимодействия пользователя с юнитами и базой. Основные требования:

- Должен быть реализован функционал управления юнитами
- Должен быть реализован функционал управления базой

Задание на разработку №4

Реализовать набор классов, для ведения логирования действий и состояний программы. Основные требования:

- Логирование действий пользователя
- Логирование действий юнитов и базы

Выполнение работы.

Для решения поставленных задач была создана программа на языке программирования C#.

Для создания поля произвольного размера Grid был создан массив Map.

Класс Grid имеет поля map, height и width со значениями которых происходит отрисовка Поля.

Проходя по массиву Map происходит отрисовка. Класс одиночка gameManager позволяет хранить данные поля и фабрики юнитов на нем в единственном экземпляре. GameManager содержит свойства: grid, player, BaseUnitFactory и MortalUnitFactory.

Для создания юнитов был использован Фабричный метод. Был создан абстрактный класс фабрики UnitFactory содержащий метод получения и создания Юнита. Классы наследники BaseUnitFactory и MortalUnitFactory переписывают метод, создающий IUnit. Интерфейс IUnit – интерфейс всех юнитов, объявляющий функцию getID.

Созданы 2 типа Юнитов: Base и Mortal. Base наследуют интерфейс IUnit, Mortal наследуют интерфейс IUnit и абстрактный класс Mortal со свойством здоровья. Base юниты — это стены и пустой пол. Mortal юниты — это игрок и враги.

Юнит Игрок имеет возможность передвижения по полю методом Move, учитывающий столкновение со стенами и врагами.

Класс базы в данной игре не предусмотрен. Однако у игрок обладает такими характеристиками как, здоровье и время. Также идет отсчет уровней.

Ландшафт создан с помощью классов Wall, Floor и Door. По клеткам Floor игрок проходит свободно, по клеткам Wall игрок не может ходить, а клетки Door открыты для игрока после взаимодействия им с определенным типом объекта.

К нейтральным объектам в созданной программе относятся Warrior, Trap, Heal и DoorSwitch. Все взаимодействия в игре реализованы с помощью паттерна Команда.

С объектами типа Warrior игрок может сражаться и теряет при этом свое здоровье. Объекты Trap – это ловушки, которые могут тратить ходы игрока с определённой вероятностью. Heal – объекты хиллок, которые возвращают первоначальное здоровье игроку.

DoorSwitch позволяет открыть двери для прохода на новый уровень. Открытие дверей реализуется через паттерн «Наблюдатель».

Все взаимодействия в игре реализованы с помощью паттерна Команда. Имеется абстрактный класс Action содержащий определение функции исполнения команды и вспомогательные функции для упрощения вывода результата команды в консоль и проверки типа результата команды. Далее от этого класса наследуются команды Move, Attack и Heal, реализующие перемещение юнитов по полю с взаимодействием их с разными типами юнитов, атаку одного юнита смертного типа другого юнита смертного типа, а также восполнение здоровья смертного юнита.

Логирование действий пользователя осуществляется с помощью паттерна Адаптер. Существует класс Recorder, который записывает логи в массив строк. Далее класс Adapter превращает массив строк в строку одним и тем же способом, передавая его классу Writer, который производит запись в файл или терминал.

Выводы.

Паттерны — это устойчивые конструкции в программировании. Все программисты в мире решают более-менее похожие задачи и для решения этих задач уже выработаны популярные решения.

Паттерны помогают сэкономить время на организации кода и сосредоточиться на решении задачи. Есть паттерны которые говорят вам как правильно написать какие-то отдельные решения в коде, например паттерн «перечисление»; есть паттерны, которые описывают как лучше разделить код приложения по папкам и что писать в конкретных файлах, например паттерн MVC и его родственники MVP и MVVM; а есть паттерны, которые рассказывают как между собой должны общаться разные модули, например паттерн «инверсия контроля».

Может показаться, что для того, чтобы применять паттерны, нужно очень глубоко понимать программирование и работать над сложными задачами, но на деле это не всегда так: часто разработчики используют паттерны, даже не

подозревая об этом, потому что некоторые паттерны встроены прямо в языки программирования.

Исходный код программы.

Исходный код программы доступен по адресу https://github.com/pooreshqa (PracticalWork3.1 - PracticalWork3.3) и на платформе Stepik.

ДОКУМЕНТАЦИЯ ПРОГРАММЫ

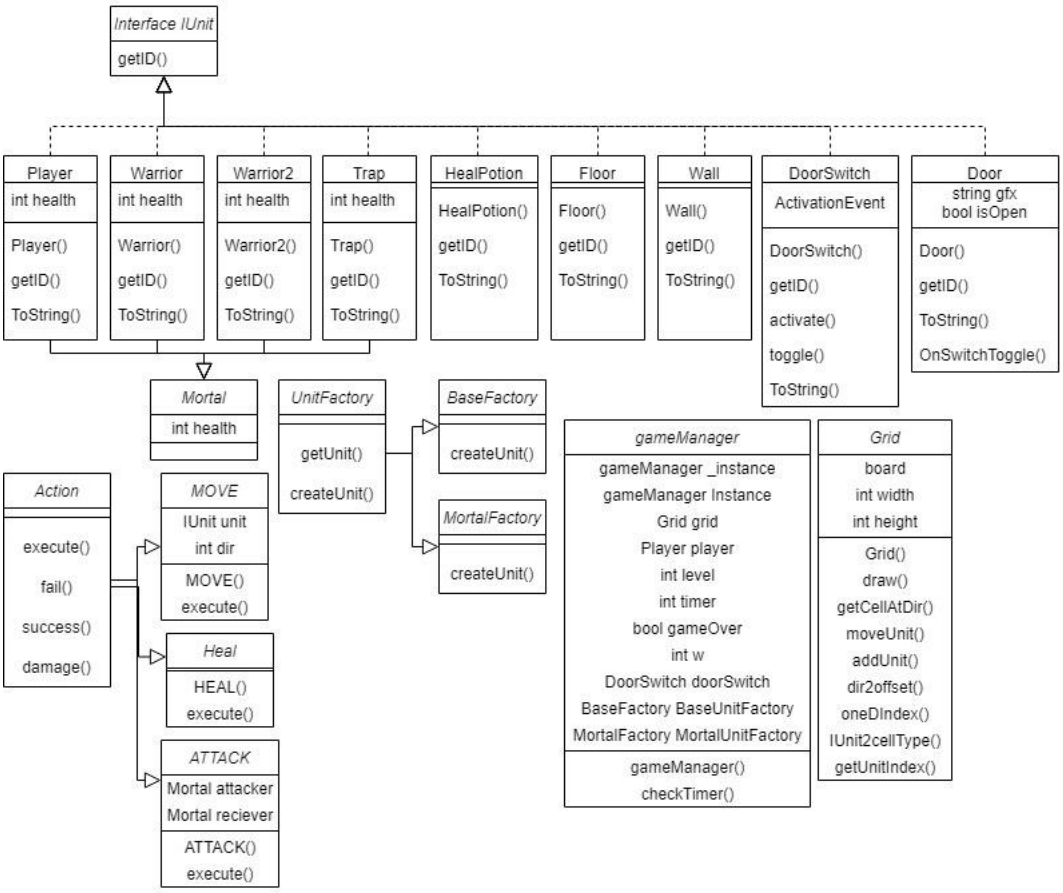

Рисунок 1 - Диаграмма Классов

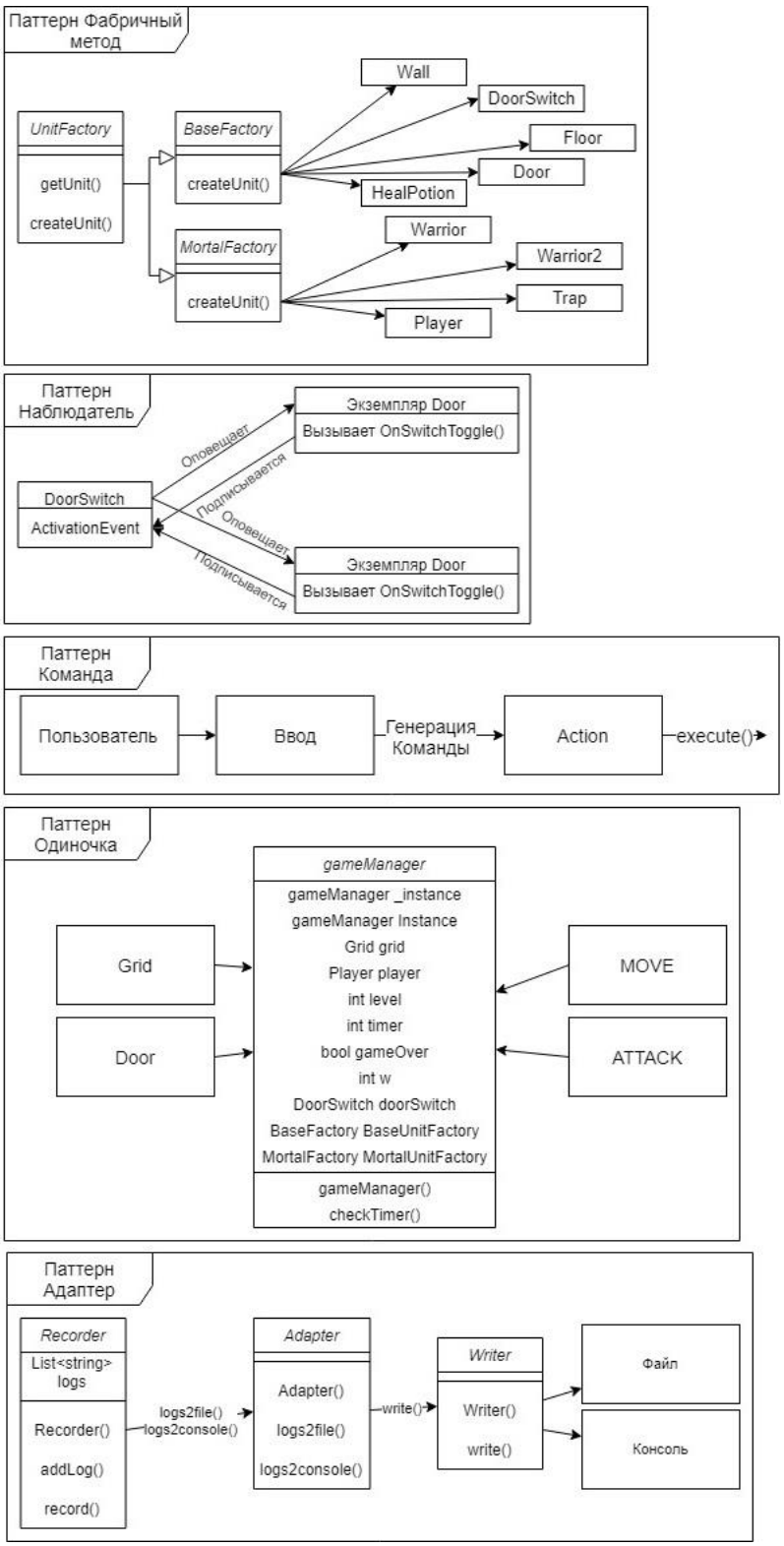

Рисунок 2 - Применяемые Паттерны

РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

Обозначения:

- Игрок – @
- Стена – #
- Пустые клетки –.
- Двери – H, после открытия - _
- Враг – I (отнимает 2 единицы здоровья)
- Враг 2 – V (отнимает 4 единицы здоровья)
- Ловушка –:
- Хилка – +
- Переключатель дверей – %

```
# # # # # #
H @ . I : H
# . I : . #
# I I I . #
H I . . % H
# # # # # #
Здоровье: 20 | Время: 10 | Уровень: 0
Перемещение:
w - Шаг вверх
d - Шаг вправо
s - Шаг вниз
a - Шаг влево
0 - Выйти
```

Рисунок 1 - Демонстрационный пример поля 1

```
# # # # # # # #
_ . . I : : _
# : . . . I #
_ : . : . : _
# . . . . . #
_ I . . I @ _
# # # # # # # #
УСПЕХ: Двери открываются, идите в них
Здоровье: 16 | Время: 2 | Уровень: 1
Перемещение:
w - Шаг вверх
d - Шаг вправо
s - Шаг вниз
a - Шаг влево
0 - Выйти
```

Рисунок 2 - Демонстрационный пример поля 2

```
# # # # # # # # # # # #
# . . . V : . . . . #
# . . . . . . V : : #
# . @ V V V . : . . #
# . . : . V . + V V #
H . . V . : V . V : H
H : V V V V : . V . H
H : . : . . . . V + H
# . . . V . : . . : #
# V . . V . V V . % #
# # # # # # # # # # #
УРОН: Игрок потерял 4 здоровья, но победил врага
Здоровье: 13 | Время: 7 | Уровень: 5
Перемещение:
w - Шаг вверх
d - Шаг вправо
s - Шаг вниз
a - Шаг влево
0 - Выйти
```

Рисунок 3 - Демонстрационный пример поля 3