**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

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Кулач Д.В. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы

Разработать гибкую, расширяемую архитектуру игры “Морской бой” с использованием шаблонов и конфигурируемого управления. Это является важным шагом в реализации проекта первой игры на языке программирования С++.

## Задание

1. Создать шаблонный класс управления игрой. Данный класс должен содержать ссылку на игру. В качестве параметра шаблона должен указываться класс, который определяет способ ввода команда, и переводящий введенную информацию в команду. Класс управления игрой, должен получать команду для выполнения, и вызывать соответствующий метод класса игры.
2. Создать шаблонный класс отображения игры. Данный класс реагирует на изменения в игре, и производит отрисовку игры. То, как происходит отрисовка игры определяется классом переданном в качестве параметра шаблона.
3. Реализовать класс считывающий ввод пользователя из терминала и преобразующий ввод в команду. Соответствие команды введенному символу должно задаваться из файла. Если невозможно считать из файла, то управление задается по умолчанию.
4. Реализовать класс, отвечающий за отрисовку поля.

**Примечание:**

* Класс отслеживания и класс отрисовки рекомендуется делать отдельными сущностями. Таким образом, класс отслеживания инициализирует отрисовку, и при необходимости можно заменить отрисовку (например, на GUI) без изменения самого отслеживания
* После считывания клавиши, считанный символ должен сразу обрабатываться, и далее работа должна проводить с сущностью, которая представляет команду.
* Для представления команды можно разработать системы классов или использовать перечисление enum.
* Хорошей практикой является создание “прослойки” между считыванием/обработкой команды и классом игры, которая сопоставляет команду и вызываемым методом игры. Существуют альтернативные решения без явной “прослойки”
* При считывания управления необходимо делать проверку, что на все команды назначена клавиша, что на одну клавишу не назначено две команды, что на одну команду не назначено две клавиши.

## Выполнение работы

Класс GameMamager

Класс GameManager управляет игрой, обрабатывая ввод пользователя, взаимодействуя с игровой логикой (Game), отображением (GameViewT) и управлением (GameControlT). Он обеспечивает гибкую и расширяемую архитектуру игры.

Поля класса:

• game: Ссылка на объект класса Game.

• game\_view: Объект класса GameViewT.

• game\_control: Объект класса GameControlT.

Методы класса:

• GameManager(Game& game, string control\_filename): Конструктор. Инициализирует поля, загружает настройки управления из файла (или устанавливает значения по умолчанию при ошибке).

• play(bool new\_game = true): Запускает игру. Если new\_game — true, вызывает inputShips для размещения кораблей игрока. В игровом цикле считывает команду, обрабатывает её через handleCommand, обновляет отображение и проверяет условия победы/поражения.

• makeNewEnemy(): Создает новое игровое поле противника со случайным расположением кораблей, используя длины кораблей из предыдущего расположения противника.

• handleCommand(char command): Обрабатывает введенную команду. Выполняет действия в зависимости от команды:

\* "help": Вызывает printHelp.

\* "print\_skills": Вызывает game\_view.printSkills.

\* "attack": Запрашивает координаты атаки и, опционально, координаты использования навыка. Вызывает game.playerTurn, затем game.enemyTurn. Обновляет отображение поля.

\* "load": Запрашивает имя файла и загружает игру из него с помощью game.load.

\* "save": Запрашивает имя файла и сохраняет игру в него с помощью game.save.

\* Другие команды: Выбрасывает исключение logic\_error.

• printHelp(): Выводит список доступных команд и их назначение, используя информацию из game\_control.

• inputShips(): отвечает за инициализацию игрового поля, размещая на нем корабли игрока и противника. Для игрока он запрашивает у пользователя координаты и ориентацию каждого корабля, проверяя корректность размещения. Для противника корабли размещаются автоматически, но с проверкой на корректность, чтобы избежать пересечений и выхода за границы поля. После размещения всех кораблей, метод обновляет состояние игры в объекте game.

Класс GameView

Класс GameView является классом, предоставляющим интерфейс для отображения игры. Конкретная реализация отображения определяется классами, переданными в качестве параметров шаблона. Это обеспечивает гибкость и расширяемость, позволяя легко менять способ отображения игры без изменения самой логики игры.

Поля класса:

• field\_view: Объект класса FieldViewT. Это параметр, определяющий, как будет отображаться игровое поле. Он отвечает за визуализацию игрового поля (как для игрока, так и для противника). Фактическая реализация отрисовки зависит от конкретного класса, переданного в качестве параметра шаблона.

• skills\_view: Объект класса SkillsViewT. Это параметр, определяющий, как будет отображаться список навыков игрока. По аналогии с field\_view, реализация зависит от конкретного класса, переданного в качестве параметра шаблона.

Методы класса:

• GameView(): Конструктор класса. Инициализирует поля field\_view и skills\_view, создавая объекты соответствующих типов.

• printEnemyField(PlayingField playing\_field, ostream& stream = cout): Отображает поле противника. Принимает объект PlayingField и ostream (по умолчанию cout), куда будет выводиться информация. Он делегирует отрисовку методу printAlien объекта field\_view.

• printPlayerField(PlayingField playing\_field, ostream& stream = cout): Отображает поле игрока. Аналогично printEnemyField, но использует метод printOwn объекта field\_view.

• printSkills(SkillsManager\* skills\_manager, ostream& stream = cout): Отображает список навыков. Принимает указатель на SkillsManager и ostream. Делегирует отрисовку методу printSkills объекта skills\_view.

Класс IfieldView

Интерфейс IfieldView служит для определения интерфейса, который должны реализовывать производные классы.

Методы класса:

• virtual void printAlien(PlayingField playing\_field, ostream& stream = cout) = 0;: Это чисто виртуальный метод (из-за = 0), который должен быть реализован в производных классах. Он отвечает за отображение игрового поля противника.

• virtual void printOwn(PlayingField playing\_field, ostream& stream = cout) = 0;: Это также чисто виртуальный метод, который должен быть реализован в производных классах. Он отвечает за отображение игрового поля самого игрока.

• ~IFieldView() = default;: Это виртуальный деструктор по умолчанию. Он важен для правильного освобождения памяти, если производные классы выделяют динамическую память. Поскольку IFieldView — абстрактный класс, он сам не выделяет память, поэтому деструктор по умолчанию вполне подходит.

Класс IskillsView

Это интерфейс, его единственная цель — определить контракт (интерфейс) для классов, отвечающих за отображение информации о навыках.

Метод:

• virtual void printSkills(SkillsManager\* skills\_manager, ostream& stream = cout) = 0: чисто виртуальный метод (из-за = 0), поэтому он не имеет реализации в самом классе ISkillsView. Любой класс, который наследуется от ISkillsView, обязан предоставить собственную реализацию этого метода.

Класс PlayingFieldView

Класс PlayingFieldView реализует интерфейс IFieldView и отвечает за отображение игрового поля в текстовом формате, используя перечисление EnumT для определения символов, отображающих различные состояния поля.

Методы класса:

• void printAlien(PlayingField playing\_field, ostream& stream = cout): Этот метод отвечает за вывод игрового поля противника. Он итерирует по каждой ячейке поля и выводит соответствующий символ на основе состояния ячейки, используя символы, определённые в перечислении EnumT. Символы X\_DELIMITER и Y\_DELIMITER используются для разделения строк и столбцов. Метод colorize используется для добавления цветовой информации (красным цветом) к номеру строки/столбца.

• void printOwn(PlayingField playing\_field, ostream& stream = cout): Этот метод аналогичен printAlien, но отображает собственное игровое поле. Он использует зелёный цвет (\033[32m) для нумерации строк и столбцов.

• string colorize(char c, string color): Вспомогательный приватный метод, который добавляет цветовую информацию к символу c, используя ANSI-последовательности. color — это ANSI-последовательность для установки цвета. Этот метод возвращает строку, содержащую цвет, символ и код сброса цвета.

Класс SkillsManagerView

Класс SkillsManagerView реализует интерфейс ISkillsView и отвечает за отображение информации о навыках (SkillsManager) в текстовом формате, используя символы, определённые в перечислении EnumT.

Методы:

• void printSkills(SkillsManager\* skills\_manager, ostream& stream = cout): Этот метод отвечает за вывод информации о навыках. Он итерирует по вектору навыков, полученному из skills\_manager->getSkills(). Для каждого навыка он использует abi::\_\_cxa\_demangle для получения читаемого имени типа навыка (удаляет лишние символы из имени типа, полученного с помощью typeid), а затем выводит его в поток stream, добавляя разделитель SKILL\_DELIMITER из перечисления EnumT после каждого навыка.

Класс AControl

Класс AControl представляет собой абстрактный базовый класс для управления игрой. Он определяет интерфейс для классов, которые будут обрабатывать пользовательский ввод и преобразовывать его в команды игры.

Поля класса:

• commands: Карта (map), которая сопоставляет строковые названия команд (string) с соответствующими символами (char), используемыми для ввода команд пользователем. Инициализируется в конструкторе.

• reverse\_commands: Карта (map), которая является обратной к commands. Она сопоставляет символы ввода (char) со строковыми названиями команд (string). Инициализируется в конструкторе.

Методы класса:

• AControl(): Конструктор класса. Инициализирует карты commands и reverse\_commands значениями по умолчанию.

• ~AControl(): Виртуальный деструктор. Необходим для корректного удаления объектов производных классов.

• virtual void load(string filename) = 0;: Чисто виртуальная функция. Она отвечает за загрузку настроек управления из файла filename.

• virtual void setDefault() = 0;: Чисто виртуальная функция. Она должна отвечает за установку значений по умолчанию для настроек управления, если загрузка из файла не удалась.

• virtual char operator[](string command) = 0;: Чисто виртуальная функция, перегружающая оператор индексирования []. Она должна возвращать символ, соответствующий строковому названию команды.

• virtual string parseCommand(char command) = 0;: Чисто виртуальная функция. Она должна возвращать строковое название команды, соответствующее введенному символу.

Класс GameControl

Класс GameControl наследуется от абстрактного класса AControl и предоставляет конкретную реализацию для обработки команд игры. Он отвечает за загрузку настроек управления из файла, установку значений по умолчанию и преобразование между символами ввода и строковыми названиями команд.

Методы класса:

• GameControl(): Конструктор класса. Поскольку AControl инициализирует commands и reverse\_commands в своём конструкторе, здесь дополнительные действия не требуются.

• ~GameControl(): Деструктор класса.

• void load(string filename): Этот метод загружает настройки управления из файла filename. Он должен прочитать файл, парсить его содержимое и заполнять карты commands и reverse\_commands.

• void setDefault(): Этот метод устанавливает значения по умолчанию для карты commands и reverse\_commands, если загрузка из файла не удалась.

• char operator[](string command): Перегрузка оператора []. Возвращает символ (char), соответствующий переданному строковому имени команды (command). Если команда не найдена, должна быть обработана ошибка).

• string parseCommand(char command): Преобразует символ ввода (command) в строковое имя команды. Возвращает строковое представление команды, соответствующее переданному символу. Обработка ошибок (если символ не найден) реализована.

• private: vector<string> split(const string &s, char delim): Вспомогательный приватный метод для разделения строки s на подстроки по разделителю delim.

main()

Код создаёт и запускает простую игру. Он определяет перечисления для символов отображения игрового поля и навыков, затем создаёт объекты, представляющие игровое поле, корабли, менеджеры кораблей и навыков, игру и менеджер игры. Инициализируется менеджер игры с использованием PlayingFieldView для отображения поля (с символами из SHIP\_VIEW\_SYMBOLS) и SkillsManagerView для отображения навыков (с символами из SKILL\_VIEW\_SYMBOLS). Наконец, запускается игровой цикл с помощью метода play() менеджера игры. Обработка исключений используется для вывода сообщений об ошибках.

Программный код приведен в приложении А.

## UML-диаграмма классов

## 

## Выводы

В ходе выполнения лабораторной работы была разработана гибкая и расширяемая архитектура игры «Морской бой» на языке C++. Было успешно применено объектно-ориентированное программирование с использованием шаблонов проектирования, что позволило создать модульную и легко модифицируемую систему. Реализовано конфигурируемое управление игрой, обеспечивающее гибкость настройки параметров игры. Разработанная архитектура заложила прочный фундамент для дальнейшего развития проекта и демонстрирует понимание принципов проектирования современных программных систем на С++. Успешная реализация проекта является важным шагом в освоении языка программирования C++ и создании игровых приложений.