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. 3343 Никишин С.А.

Преподаватель Жангиров Т. Р.

Санкт-Петербург

2024

Цель работы

Изучить работу шаблонов, путём усовершенствования программы из предыдущей лабораторной работы. Необходимо создать: шаблонный класс управления игрой, шаблонный класс отображения игры (надзиратель), класс считывания ввода из терминала и класс отрисовки.

Задание

a. 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.

b. 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.

c. 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.

d. 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 считывании 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

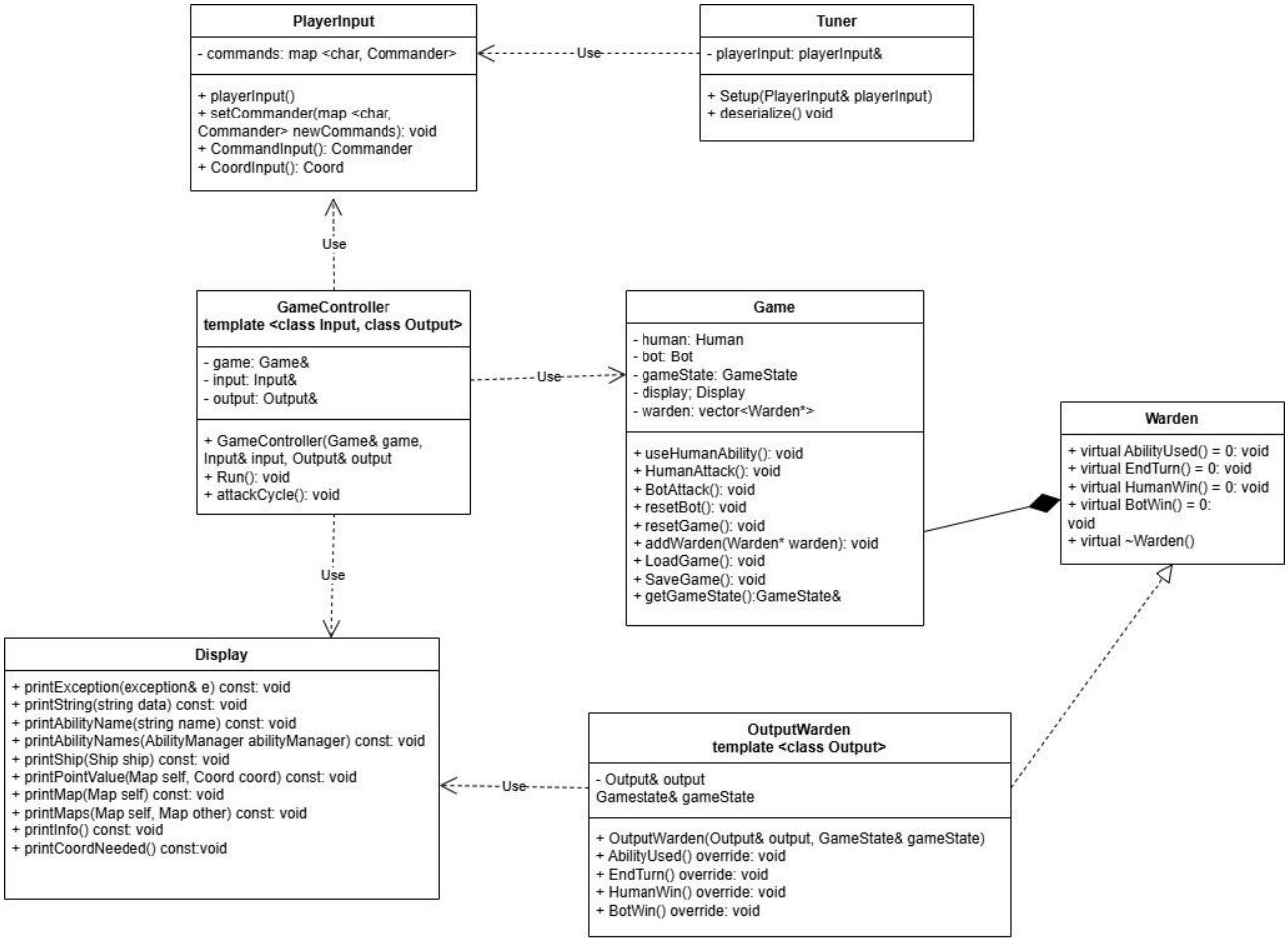

Рисунок 1 – UML-диаграмма классов

Код программы содержит реализацию классов: *GameController*, *PlayerInput*, *Tuner*, *Display*, *Warden* и *OutputWarden*.

Классы *GameController*, *OutputWarden*, *PlayerInput* и *Display* были добавлены согласно заданию. *GameController* – это шаблонный класс управления игрой, он получает переведённую информацию из консоли в команду и вызывает необходимые методы класса игры. *OutputWarden* – класс, реагирующий на различные события в игре и выводящий различную информацию на их основе. *PlayerInput* отвечает за считывание ввода из консоли и его преобразование в команды. *Display* отрисовывает поля и выводит большую часть информации в консоль.

Класс *Tuner* дополняет класс *PlayerInput* и даёт возможность получать соответствие команды введённому символу из файла.

Класс Warden является классом-интерфейсом для всех наблюдателей, в том числе *OutputWarden*.

GameController является шаблонным классом для управления игрой. Он имеет следующие поля:

- *Game& game* – ссылка на класс игры.
- *Input& input* – ссылка на шаблонный класс ввода.
- *Output& output* – ссылка на шаблонный класс вывода.

И следующие методы:

- *void Run()* – реализует цикл игры, вызывая определённые методы игры в зависимости от полученной команды.
- *void attackCycle()* – проводит цикл атаки игрока и бота.

Класс *PlayerInput* преобразует консольный ввод в команды для обработки. Он имеет следующее поле:

- *map <char, Commander> commands* – словарь, где ключи – кнопки, а значения показывают, за какие команды они отвечают.

И следующие методы:

- *void setCommander(map <char, Commander> newCommands* – заменяет команды по умолчанию (для считывания из файла).
- *Commander CommandInput()* – обрабатывает полученную инструкцию и преобразует её в команду. Если такой команды нет, возвращает информационную команду.
- *Coord CoordInput()* – обрабатывает координаты с консоли и возвращает их.

Класс *Tuner* отвечает за загрузку команд из файла. Он имеет следующее поле:

- *PlayerInput & playerInput* – ссылка на обработчик входных данных.

И метод для загрузки из файла:

- *void deserializeSetup()* – загружает новые команды из файла, если они валидные.

Класс *Display* является отрисовщиком поля и других объектов. Он имеет следующие методы:

- *void printException(exception& e) const* – выводит исключение.
- *void printString(string data) const* – выводит произвольную поданную строку.
- *void printAbilityName(string name) const* – выводит имя поданной способности.
- *void printAbilityNames(AbilityManager abilityManager) const* – выводит все доступные способности.
- *void printShip(Ship ship) const* – выводит всю информацию о корабле.
- *void printPointlValue(Map self, Coord coord) const* – выводит информацию о ячейке поля.
- *void printMap(Map self) const* – выводит одно поле.
- *void printMaps(Map self, Map other) const* – выводит оба поля.
- *void printInfo() const* – выводит информационную справку о кнопках по умолчанию.
- *void printCoordNeeded() const* – выводит сообщение о необходимости ввода координат.

Класс *Warden* является классом-шаблоном для всех наблюдателей. Он имеет следующие виртуальные методы:

- *virual void abilityUsed() = 0* – виртуальный метод, вызываемый при использовании способности.
- *virtual void EndTurn() = 0* – виртуальный метод, вызываемый после хода игрока и бота.
- *virtual void HumanWin() = 0* – виртуальный метод, вызываемый в случае победы игрока.

- *virtual voidBotWin() = 0* – виртуальный метод, вызываемый в случае победы бота.

Шаблонный класс *OutputWarden* наследуется от класса *Warden* и отвечает за вывод информации в консоль при определённых событиях. Он имеет следующие поля:

- *Output& output* – ссылка на шаблонный класс вывода.
- *GameState& gameState* – ссылка на состояние игры.

И следующие методы:

- *void abilityUsed() override* – при использовании способности вызывается и выводит поля и названия доступных способностей.
- *void EndTurn() override* – при окончании хода бота вызывается и выводит поля.
- *void HumanWin() override* – при победе игрока вызывается и выводит информацию о его победе.
- *void BotWin() override* – при победе бота вызывается и выводит информацию о его победе.

Тестирование:

Происходит симуляция игры между игроком (слева) и ботом (справа), для этого используется большая часть реализованных методов внутри классов. Поле игрока изначально открыто, а вражеское скрыто. В начале хода игрок может использовать одну случайную способность или сразу перейти к атаке вражеского поля.

В классе *GameController* реализована логика игры, которая позволяет выбирать действия в зависимости от команд пользователя и вызывать методы *Game*. Класс управления игрой с помощью команд может: провести обычную атаку, использовать способность и атаковать, загрузить игру, получив состояния кораблей, поля и способностей; сохранить игру, уже записав состояния игровых сущностей; выйти из игры.

При победе игрок продолжает игру с сохранением его поля и с новым противником. В случае победы бота, игру можно продолжить, обнулив вообще всё.

Рисунок 2 – Начало игры

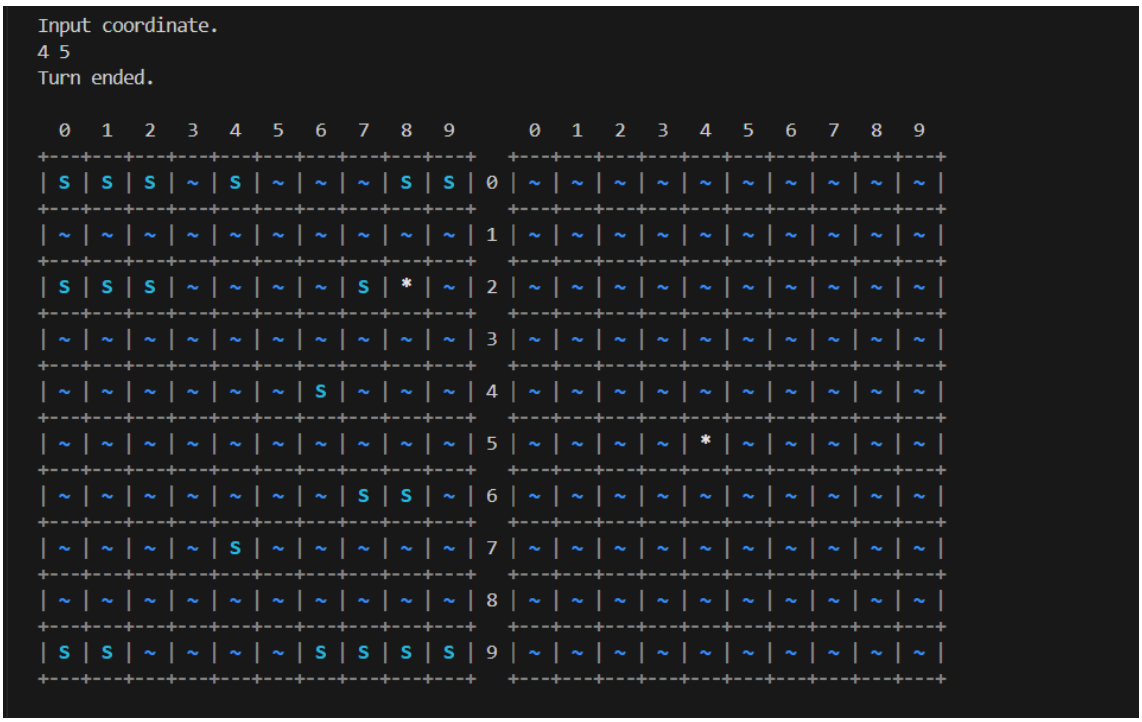

Рисунок 3 – Выполнение атаки

Рисунок 4 – Игра сохранена и закрыта.

Рисунок 5 – Игра загружена

Выводы

Во время выполнения лабораторной работы, была изучена работа шаблонных классов и созданы соответствующие заданию классы.