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. 3341 _____ Бойцов В.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Целью работы является изучение основ объектно-ориентированного программирования, создание архитектуры управления игрой на основе разработанных ранее классов, а также создание систем классов, ответственных за ввод и вывод информации, необходимой для игрового процесса.

Для выполнения поставленной цели требуется:

- Разработать архитектуру классов, отвечающих за управление игрой;
- Разработать систему команд, позволяющих выполнять определенные игровые действия;
- Создать систему классов, ответственных за отрисовку игры и считывание действия пользователя;
- Связать реализованные классы.

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 команды, и переводит 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, переданным 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 вызываемые методы 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.

Данная лабораторная работа подразумевает создание трёх связанных систем: в первую очередь, системы управления игрой и игровым процессом посредством полученных команд, затем системы, позволяющей трактовать ввод пользователя как команду, и, наконец, систему отслеживания игрового процесса и отображения игры. Система управления игрой должна быть достаточно гибкой, чтобы позволять организовывать игровой процесс для разного количества участников игры (ботов или игроков) и менять способы ввода команд и отображения игры без изменения непосредственно системы управления.

Для удовлетворения таких требований систему управления игрой разделена на две части – класс, отвечающий за непосредственную организацию игрового процесса, и классы, отвечающие за действия участников игры. Первый класс осуществляет контроль за игрой, реализует игровой цикл и обращается ко вторым классам, когда необходимо получить действия от участника игры.

Класс непосредственного управления игрой – *GameController*. Он хранит в себе ссылку на игру *Game& mGame* и вектор указателей на контроллеры участников игры *vector<ParticipantController*> mControllers*, о которых будет подробно сказано ниже. Основные методы, раскрывающие функционал класса:

Void startGame() – метод, который запускает игровой процесс, вызывая метод

Void RunGameCycle(), реализующий игровой цикл. в начале метода происходит запрос к контроллерам каждого из участников игры в ожидании расстановки кораблей для начала игры, затем, пока количество живых игроков в игре больше 0, происходит реализация цикла одного игрового раунда вызовом метода

Void runRoundCycle(), который отвечает за организацию раунда и отправку запросов на действие к участникам игры в течение всего раунда.

Данный класс также имеет методы, добавляющие новых игроков по контроллеру (*void addPlayerController(ParticipantController&)*) и ботов (*void*

addBots(int number)), а также метод для синхронизации сущностей игры и их контроллеров и для инициализации отслеживания игры у участников.

Для реализации выполнения игровых действий была создана система команд. В её основе лежит интерфейс *ICommand*, который определяет основное действие любой команды – метод её выполнения *virtual void execute(Game& game)*. Все команды внутри себя при вызове этого методы вызывают определенные методы игры для выполнения заложенных в них действий.

Непосредственно игровые команды реализуют интерфейс *ICommand* и представлены следующими классами:

- *PlaceShipCommand* – команда постановки корабля на поле. Считается, что корабль ставит текущий активный участник. Команда принимает в конструкторе индекс корабля, который необходимо поставить, в менеджере кораблей, координаты корабля, по которым его необходимо поставить, и его ориентацию. В методе *execute* команда вызывает метод игры *placeShip*.
- *AttackCommand* – команда атаки указанной клетки поля указанного игрока. Индекс игрока для атаки и координаты атакуемой клетки команда принимает в конструкторе, а в методе execute вызывает метод игры attack.
- *DoubleDamageCommand*, *ScannerCommand* и *ShellingCommand* – команды для использования указанной способности. В конструкторе эти команды принимают необходимые для их работы аргументы (для обстрела – индекс игрока, а для сканера – индекс игрока и координаты сканируемой области). В методе *execute* данные команды создают объект класса настроек соответствующей способности, а затем вызывают метод игры *castAbility*.
- *SaveCommand* и *LoadCommand* – команды для сохранения и загрузки соответственно. Аргументов не требуют и в методе *execute* вызывают методы игры *save* и *load* соответственно.

Исполнение данных команд инициируется игровым контроллеров, а их создание происходить через контроллеры участников игры, о который сейчас пойдет речь.

Контроллер абстрактного участника игры представлен абстрактным классом *ParticipantController*. Данный класс содержит в себе указатель на участника игры *Participant* mParticipant* (представление игрока в модели игры), а также свой класс-наблюдатель за игрой – *GameObserver mObserver*, о котором речь пойдет далее. Основной метод контроллера участника игры – *virtual ICommand* getAction()* – позволяет получать от участника игры команду, соответствующую действию, которое участник хочет совершить. Также есть ряд вспомогательных методов: *bool isReady(),* отвечающий за готовность участника к новому раунду, *void observe(Game& game),* вызывающий процесс отслеживание игровых изменений в наблюдателе участника игры.

От данного класса наследуются класс *BotController*, отвечающий за контроллер бота, и шаблонный класс *PlayerController*, где в качестве параметра шаблона передаётся способ ввода информации игроком. Контроллер бота в методе *getAction* обращается за информацией к своему наблюдателю, получает из него необходимую информацию, анализирует её и выдаёт свою команду, а игрок обращается к своему классу *Input* за получением команды от ввода реального игрока.

В качестве способа ввода данных игроком по умолчанию используется класс *CLIInput*, реализующий консольный ввод действий игроком. Его поля – контейнеры *unordered_map* для хранения клавиш и соответствующих им на данный момент команд *mBindedKeys* и команд и соответствующих им лямбда-функций для сбора дополнительных параметров *mCommandsParser*. Основной метод класса – *ICommand* readCommand()*. Данный метод считывает нажатую клавишу, сопоставляет её с командой через *mBindedKeys*, а затем собирает команду с помощью вызова соответствующей лямбда-функции из *mCommandsParser*. Эти функции могут при необходимости вызывать вспомогательные методы класса – *Coords readCoords()*, *int readIndex()* и *Orientation readOrientation()* для считывания координат, какого-нибудь индекса и ориентации корабля. Также класс имеет метод *void rebindKeysFromFile(string fileName)*, который позволяет считать из файла новые настройки соответствия

клавиш и команд. Метод путём добавления команд в новый контейнер *unordered_map* проверяет правильность новых настроек и применяет их целиком только в том случае, если они правильные.

Осталось описать систему отображения игры. Начнём с низшего уровня абстракции: исходный способ вывода на экран игры – класс *ConsoleDisplayer*. В нем реализован ряд методов для вывода в консоль основных игровых сущностей – корабля, менеджера кораблей, своего и неприятельского поля, всем игры в целом.

Для использования конкретных классов вывода игры на экран используются классы отображения. Интерфейсный класс *IGameDisplayer* описывает поведение такого класса – методы *void display(GameInfo&, int)* для вывода всей игры в обычной её фазе, *void displayShipPositioning(GameInfo&, int)* для вывода своего поля в момент расстановки кораблей, *void displayAbilityResult(AbilityResult&)* для отображения результата какой-то способности, а также методы *void informNewRound()* и *informNewGame(),* использующиеся для оповещения о начале нового раунда или новой игры.

В разработанной архитектуре данный интерфейс реализован шаблонным классом *ConcreteGameDisplayer*, где в качестве параметра шаблона передаётся способ отображения игры *Displayer*. Данный класс вызывает у *Displayer* его соответствующие методы вывода игры на экран.

Связь между отображением и контроллерами игроков построена с помощью класса *GameObserver*, который реализует наблюдение за игрой. В качестве полей данного класса содержатся ссылка на игру *mGame* и указатель на класс отображения *mDisplayer*, а также некоторые служебные поля для регистрации того, что уже было отслежено (например, начало нового раунда). Основной метод данного класса – *void track(int pIndex),* который отслеживает положение дел в игре и при необходимости производит отрисовку. Этот метод класса вызывается игровым контроллером после каждого действия игроков, чтобы обновить данные, известные игрокам.

Диаграмма классов, разработанных в ходе выполнения лабораторной работы, представлена на рис.1.

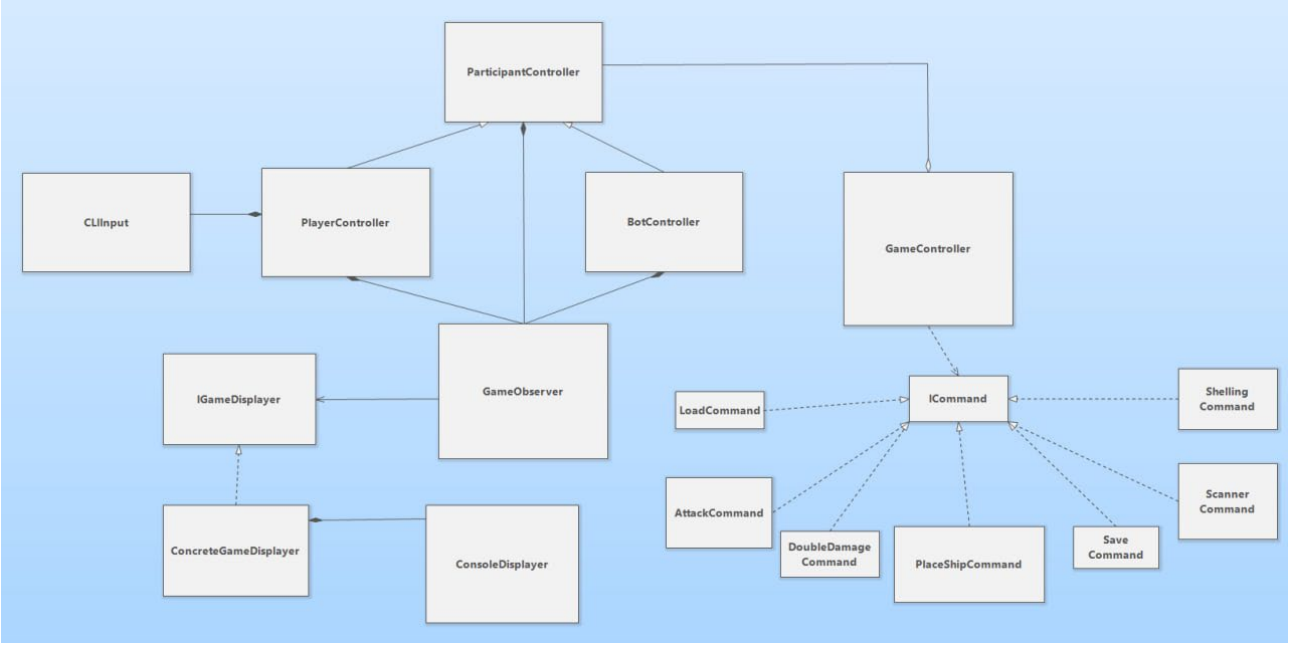

Рисунок 1 – диаграмма классов

Итоговая архитектура позволяет как заменять способы ввода/вывода информации, так и организовывать игровой процесс для разного количества участников игры. Также все участники относительно игрового контроллера и игры практически одинаковы, а способы получения от них команд идентичны, что позволяет, например, заменять логику бота без изменения структуры программы и основных классов. Более того, логика работы бота может быть задана извне.

Результаты тестирования см. в приложении А.

Выводы.

В результате выполнения лабораторной работы были изучены основы Объектно-ориентированного программирования на языке C++. Была придумана архитектура, объединяющая разработанную ранее систему классов для реализации игрового процесса, были написаны классы, позволяющие управлять игрой, вызывать игровые действия с помощью системы команд, а также отрисовывать игру или считывать ввод пользователя любым подходящим способом. Реализованная архитектура достаточно масштабируема и позволяет организовывать игровой процесс для произвольного количество игроков и компьютерных соперников.

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Тестирование написанных классов было выполнено в виде небольшой программы, результат работы которой представлен ниже:

```
Key bindings updated successfully.

Current binded keys are:
f - AttackCommand
q - DoubleDamageCommand
l - LoadCommand
r - PlaceShipCommand
s - SaveCommand
e - ScannerCommand
w - ShellingCommand



New Game has started!


Inactive ships in manager: 1
Ship (0 , 0) condition:
        2
Active ships in manager: 0

..........
..........
..........
..........
..........
..........
..........
..........
..........
..........


Enter Command:
r 0 2 5 h

Enter ship index: <i>

Enter  coordinates: <x y>

Enter ship orientation: <v> for vertical and <h> for horizontal
```

```
Move 0
This is you! #0
Shelling
Inactive ships in manager: 0
Active ships in manager: 1
Ship (2 , 5) condition:
        2

..........
..........
..........
..........
..........
..0.......
..........
..........
..........
..........

Enemy #1
Shelling
Inactive ships in manager: 0
Active ships in manager: 1
Ship (0 , 0) condition:
        2

----------
----------
----------
----------
----------
----------
----------
----------
----------
----------


Enter Command:
f 1 0 0

Enter target player index: <i>

Enter  coordinates: <x y>

Move 1
This is you! #0
Shelling
Inactive ships in manager: 0
Active ships in manager: 1
Ship (2 , 5) condition:
```

```
        2

..........
..........
..........
..........
..........
..0.......
..........
..........
..........
..........

Enemy #1
Shelling
Inactive ships in manager: 0
Active ships in manager: 1
Ship (0 , 0) condition:
        1

x---------
----------
----------
----------
----------
----------
----------
----------
----------
----------


Move 1
This is you! #0
Shelling
Inactive ships in manager: 0
Active ships in manager: 1
Ship (2 , 5) condition:
        2

..........
..........
..........
..........
..........
..0.......
..........
..........
..........
..........
```

```
Enemy #1
Shelling
Inactive ships in manager: 0
Active ships in manager: 1
Ship (0 , 0) condition:
        1

x---------
----------
----------
----------
----------
----------
----------
----------
----------
----------


Move 2
This is you! #0
Shelling
Inactive ships in manager: 0
Active ships in manager: 1
Ship (2 , 5) condition:
        2

..........
..........
..........
..........
..........
..0.......
..........
.*........
..........
..........

Enemy #1
Shelling
Inactive ships in manager: 0
Active ships in manager: 1
Ship (0 , 0) condition:
        1

x---------
----------
----------
----------
----------
----------
```

```
----------
----------
----------
----------


Move 2
This is you! #0
Shelling
Inactive ships in manager: 0
Active ships in manager: 1
Ship (2 , 5) condition:
        2

..........
..........
..........
..........
..........
..0.......
..........
.*........
..........
..........

Enemy #1
Shelling
Inactive ships in manager: 0
Active ships in manager: 1
Ship (0 , 0) condition:
        1

x---------
----------
----------
----------
----------
----------
----------
----------
----------
----------
```