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 _____ Отмахов Д.В.

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 (наблюдатель), 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

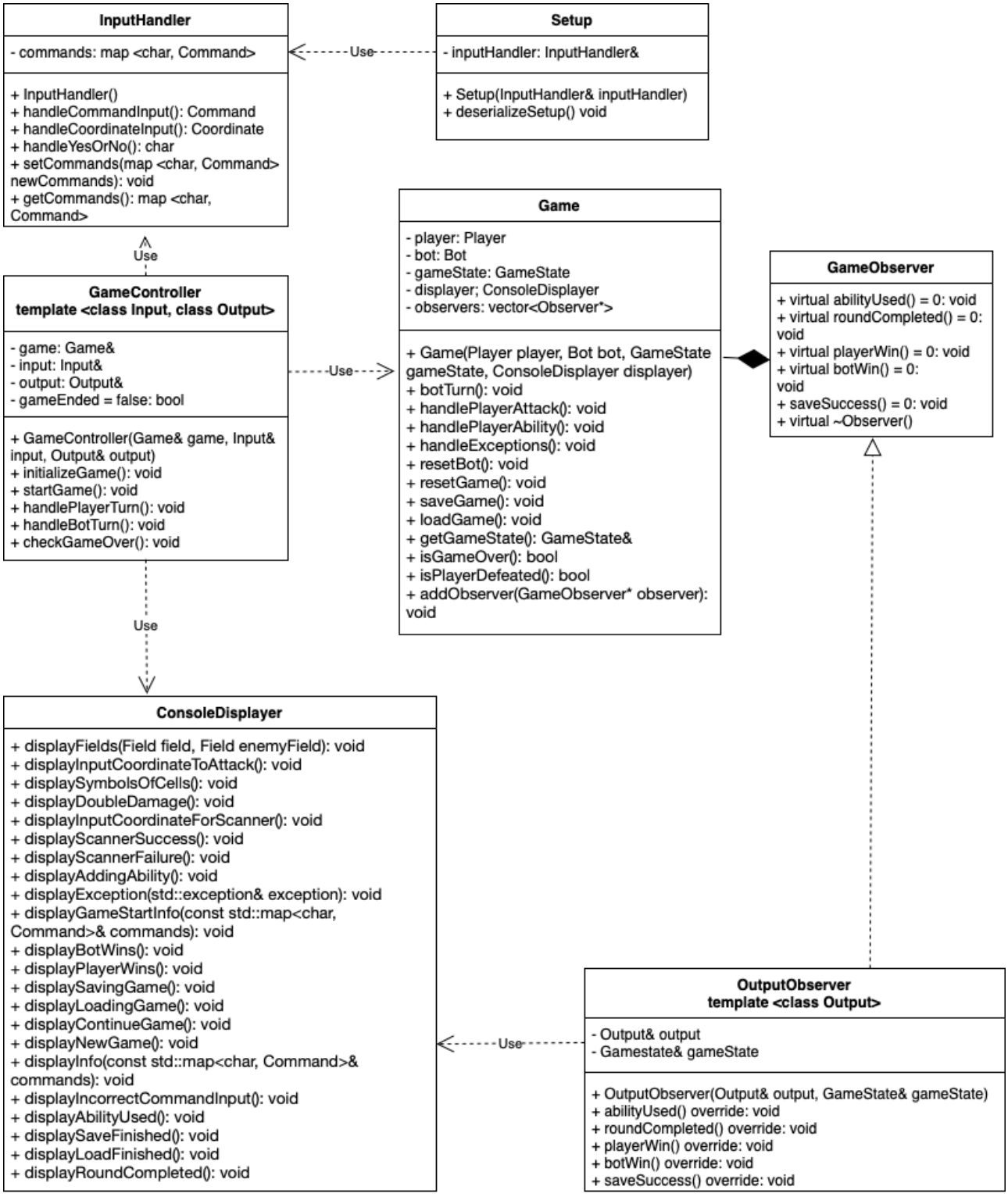

Рисунок 1 – UML-диаграмма классов

Код программы содержит реализацию классов: *GameController*, *InputHandler*, *ConsoleDisplayer*, *Setup*, *GameObserver* и *OutputObserver*.

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

Класс *Setup* дополняет класс *InputHandler* и даёт возможность задавать команды из файла.

Класс *GameObserver* является классом-интерфейсом для всех наблюдателей, в том числе *OutputObserver*.

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

- *Game& game* – ссылка на класс игры;
- *Input& input* – ссылка на шаблонный класс ввода;
- *Output& output* – ссылка на шаблонный класс вывода;
- *Bool gameEnded = false* – флаг состояния игры.

И следующие методы:

- *void initializeGame()* – метод инициализации игры;
- *void startGame()* – метод начала игры, реализует цикл игры;
- *void handlePlayerTurn()* – обрабатывает ход игрока;
- *void handleBotTurn()* – обрабатывает ход бота;
- *void checkGameOver()* – отвечает за начало новой игры или ее продолжение при победе или проигрыше игрока.

Класс *Setup* отвечает за загрузку команд из файла. Он имеет следующие поля и методы:

- *InputHandler& inputHandler* – ссылка на обработчик входных данных;
- *void deserializeSetup()* – загружает новые команды из файла, если они валидные.

Класс *InputHandler* преобразует консольный ввод в команды для обработки. Он имеет следующие поля и методы:

- *map <char, Command> commands* – словарь, где ключи – кнопки, а значения показывают, за какие команды они отвечают;
- *Command handleCommandInput()* – обрабатывает полученную инструкцию и преобразует её в команду. Если такой команды нет, возвращает информационную команду;
- *Coordinate handleCoordinateInput()* – обрабатывает координаты с консоли и возвращает их;
- *char handleYesOrNo()* – обрабатывает ввод пользователя (для продолжения игры);
- *void setCommands(map <char, Command> newCommands)* – заменяет команды по умолчанию (для считывания из файла).
- *map <char, Command> getCommands()* – возвращает словарь текущих команд.

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

- *void displayFields(Field field, Field enemyField)* – отображает два поля одно игрока, другое бота (со скрытыми кораблями);
- *void displayInputCoordinateToAttack()* – отображает запрос координат для атаки;
- *void displaySymbolsOfCells()* – отображает значения клеток поля;
- *void displayDoubleDamage()* – отображает, что в следующей атаке будет использоваться двойной урон;

- *void displayInputCoordinateForScanner()* – отображает запрос координат для сканирования;
- *void displayScannerSuccess()* – отображает успешный результат сканера;
- *void displayScannerFailure()* – отображает безуспешный результат сканера;
- *void displayAddingAbility()* – отображает добавление новой способности;
- *void displayException(std::exception& exception)* – отображает исключение;
- *void displayGameStartInfo(const std::map<char, Command>& commands)* – отображает информацию о командах;
- *void displayBotWins()* – отображает победу игрока;
- *void displayPlayerWins()* – отображает победу бота;
- *void displaySavingGame()* – отображает, что идет сохранение игры;
- *void displayLoadingGame()* – отображает, что идет загрузка игры;
- *void displayContinueGame()* – отображает вопрос игроку о желании продолжить игру;
- *void displayNewGame()* – отображает вопрос игроку о желании начать новую игру;
- *void displayInfo(const std::map<char, Command>& commands)* – отображает информацию о командах;
- *void displayIncorrectCommandInput()* – отображает, что введенная команда некорректна;
- *void displayAbilityUsed()* – отображает, что использована способность;
- *void displaySaveFinished()* – отображает, что сохранение завершено;
- *void displayLoadFinished()* – отображает, что загрузка завершена;
- *void displayRoundCompleted()* – отображает, что раунд завершен.

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

- *virual void abilityUsed() = 0* – виртуальный метод, вызываемый при использовании способности;

- *virtual void roundCompleted() = 0* – виртуальный метод, вызываемый после окончания раунда;
- *virtual void playerWin() = 0* – виртуальный метод, вызываемый в случае победы игрока;
- *virtual void botWin() = 0* – виртуальный метод, вызываемый в случае победы бота;
- *virtual void saveSuccess()* – виртуальный метод, вызываемый при успешном сохранении игры.

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

- *Output& output* – ссылка на шаблонный класс вывода.
- *GameState& gameState* – ссылка на состояние игры.
- *virual void abilityUsed() = 0* – метод, вызываемый при использовании способности;
- *virtual void roundCompleted() = 0* – метод, вызываемый после окончания раунда;
- *virtual void playerWin() = 0* – метод, вызываемый в случае победы игрока;
- *virtual void botWin() = 0* – метод, вызываемый в случае победы бота;
- *virtual void saveSuccess()* – метод, вызываемый при успешном сохранении игры.

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.

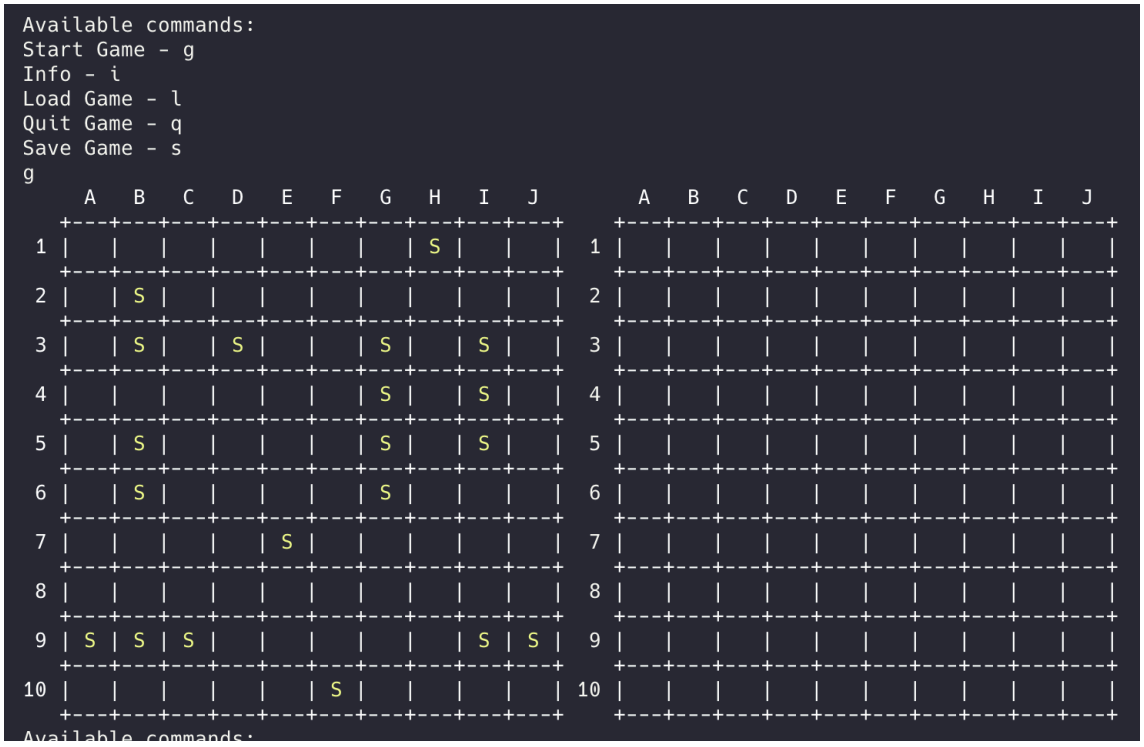

Рисунок 2 – Начало игры

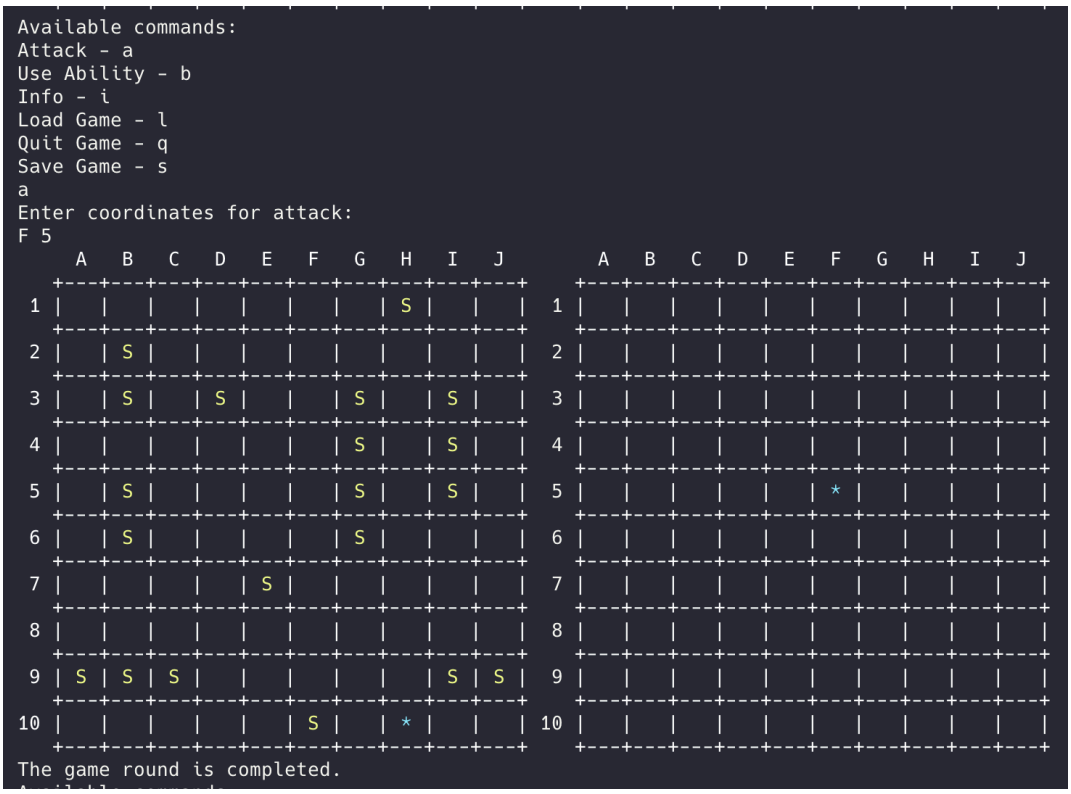

Рисунок 3 – Выполнение атаки

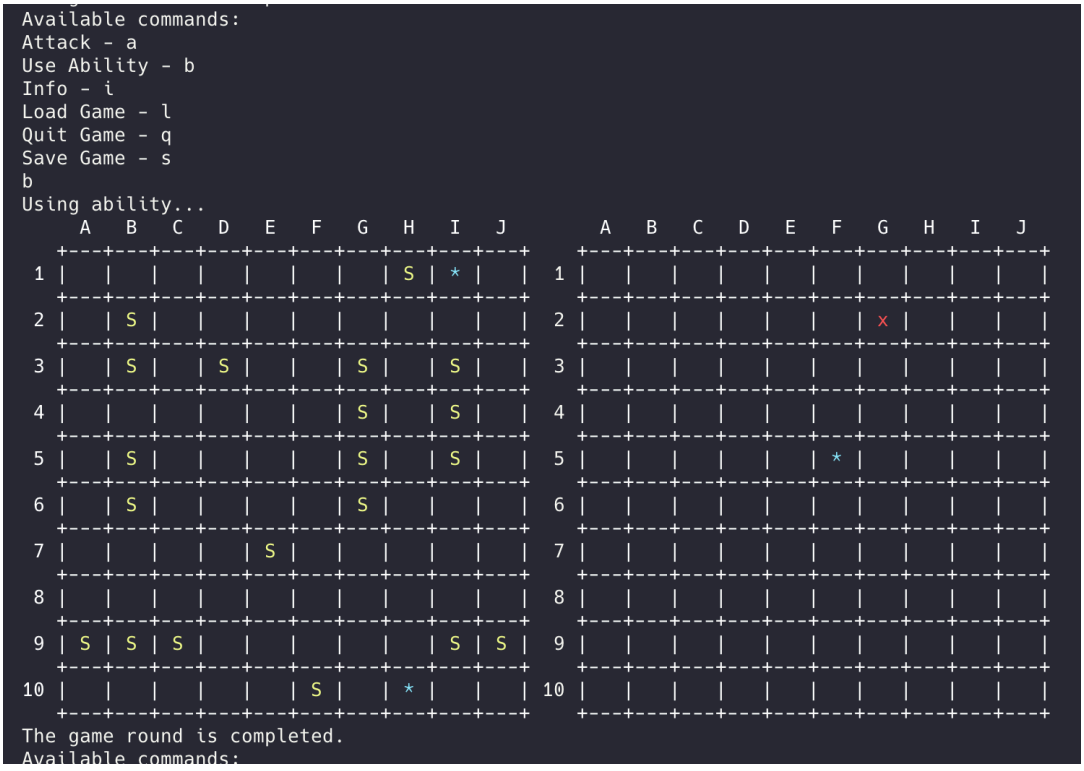

Рисунок 4 – Использование способности

```
Available commands:
Attack - a
Use Ability - b
Info - i
Load Game - l
Quit Game - q
Save Game - s
s
Saving game...
The game has been saved successfully.
```

Рисунок 5 – Сохранение игры

```
Available commands:
Attack - a
Use Ability - b
Info - i
Load Game - l
Quit Game - q
Save Game - s
l
Loading game...
```

Рисунок 6 – Загрузка игры

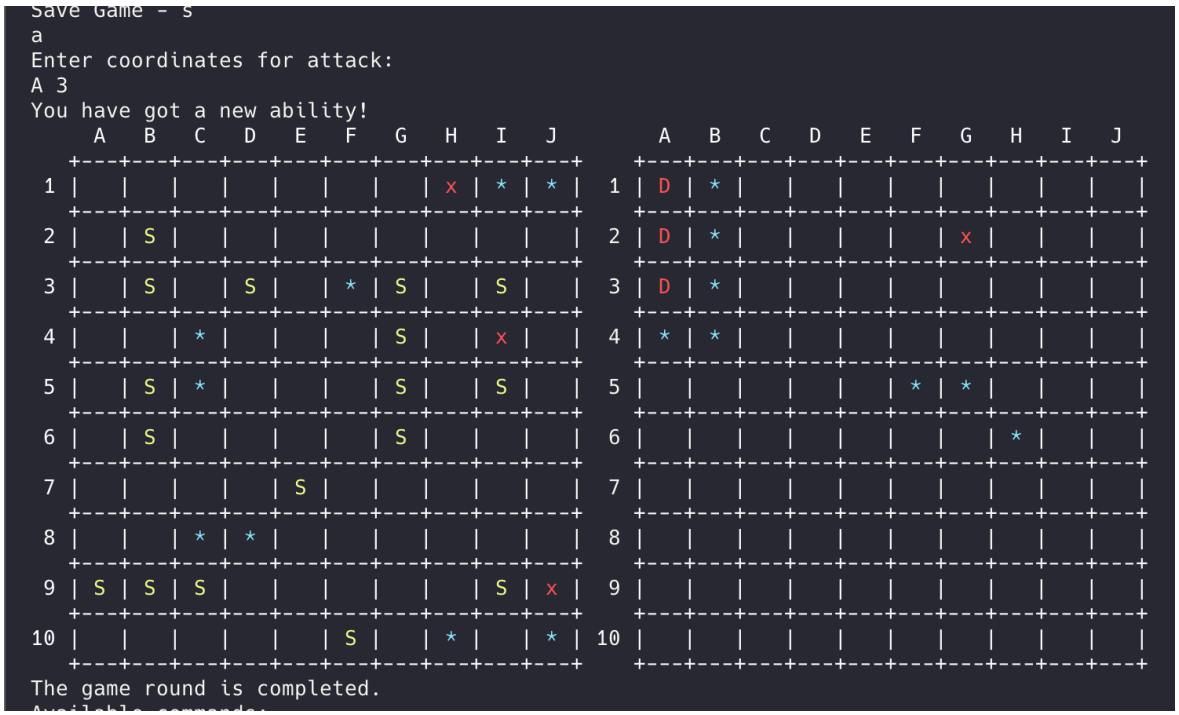

Рисунок 7 – Уничтожение корабля

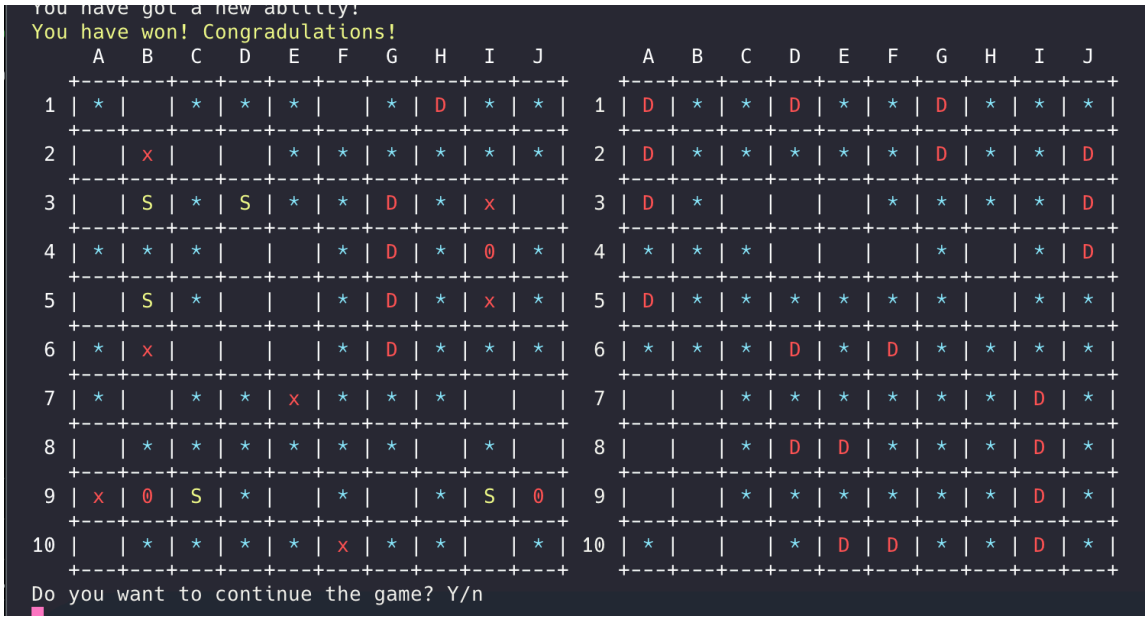

Рисунок 8 – Окончание игры

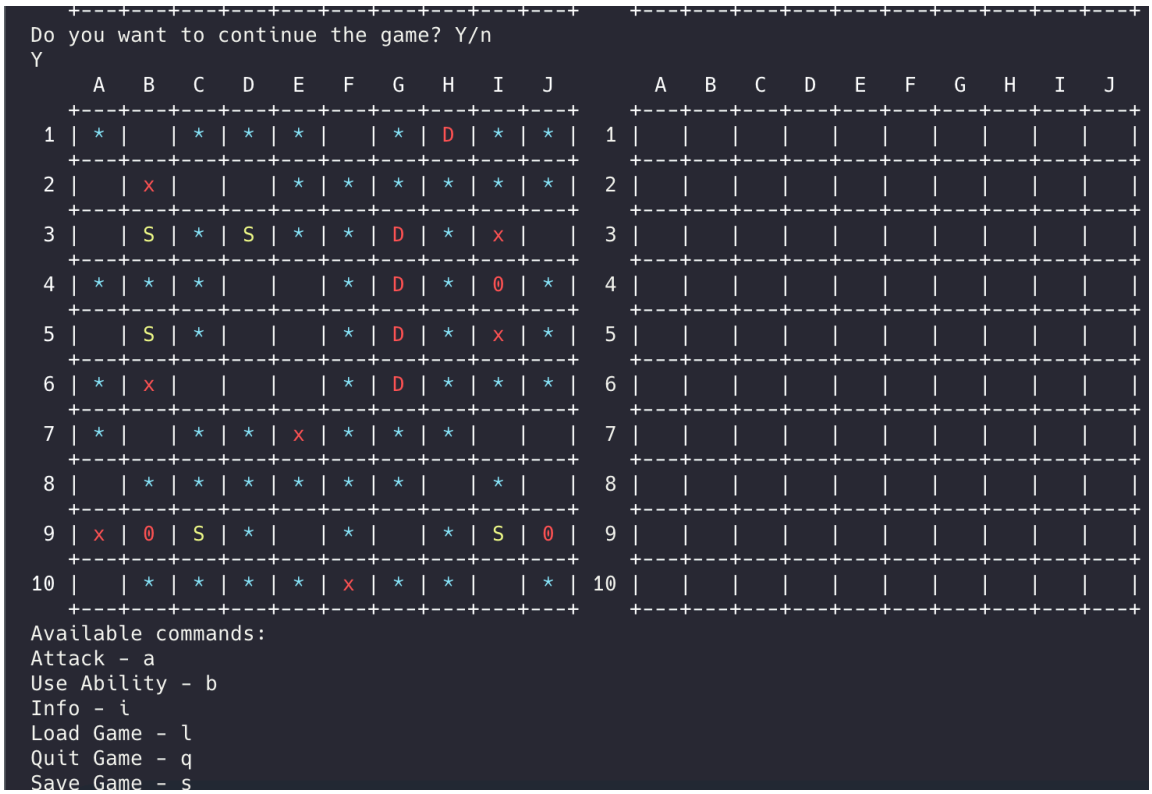

Рисунок 9 – Продолжение игры

Выводы

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