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. 3344 _____ Кузнецов Р.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Изучить шаблонные классы в языке программирования C++. Создать шаблонные классы в текущем проекте — игре «морской бой» для отображения процесса игры и взаимодействия.

Задание.

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.

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.

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.

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.

Созданы классы: GameController, ConsoleRenderer, GameDisplay. Модифицирован класс UserInput.

1) Класс UserInput.

Класс, необходимый для считывания команд с консоли. Теперь в конструктор класс принимает название файла — список клавиш для работы с игрой. Проверяется, что такой файл существует, также что несколько клавиш не были назначены на одну команду или наоборот. Устанавливаются клавиши из файла. Иначе клавиши задаются автоматически. Сам класс представляет собой взаимодействие игры с пользователем: считывание координат для атаки и постановки корабля, считывание ориентации для постановки корабля, считывание название файла для сохранения, загрузки игр, вывод информации о возможных взаимодействиях в меню и в самой игре, установка клавиш для взаимодействия, работа с клавишами и очистка терминала и буфера.

Метод getCommand использует низкоуровневую настройку терминала. Устанавливается канонический ввод для считывания одного символа и убирается отображение символов. Настройки применяются и считывается один символ, затем настройки возвращаются к стандартным. Если в command команды в нижнем регистре, то приводится в нижний, если в верхнем то в верхний. Если символ отсутствует, возвращается команда None, то обозначает либо отсутствие символа, либо некорректность. В классе присутствует проверка на ввод числа, а не другого символа. (для атаки или постановки корабля)

2) Класс GameDisplay.

Промежуточный шаблонный класс между игрой и ее отображением в консоли. Позволяет выводить на экран изменения в игре. Т.к. это шаблонный класс, то заменить вывод из консоли например в GUI, что не составит проблем. Реализованы такие методы как: вывод поля (вражеского или игрока), вывод сообщений в консоль и вывод информации о начальных кораблях.

3) Класс ConsoleRenderer.

Класс, который реализует логику, для вывода информации на экран. Рисование поля было взято из класса GameField, но символы для вывода кораблей/пустого поля/поля без корабля/поврежденных кораблей были заменены эмодзи для более четкого определения состояния игры. Также вначале программы выводится информация о начальных кораблях, которые пользователь может разместить на поле. Есть методы для вывода любой строки в консоль.

4) Класс GameController.

Шаблонный класс, который соединяет класс ввода, класс вывода и методы игры. Как сказано выше, такой класс позволяет не задумываться о способе считывания и вывода на экран информации, засчет шаблонов. В полях содержит ссылки на указанные выше классы. В самом классе существуют такие методы как:

Начало игры — меню, здесь возможно сделать 3 вещи: начать новую игру, загрузить игру либо выйти из игры. Если пользователь начинает новую игру, либо загружает старую, действия переходят в метод самой логики игры.

Он представляет собой два бесконечных цикла, которые повторяют ходы игрока и бота. Взависимости от выбора игрока, вначале метода происходят необходимые процессы: либо ставятся корабли и игрока и противника, либо ставятся новые корабли для противника. Затем происходит вывод полей, ход игрока и ход бота, если кто-то из них победил, игра начинает заново, но по разным сценариям.

Еще один метод позволяет делать выбор в игре для пользователя: атака клетки, использование способности, сохранение, загрузка или выход из игры. Все действия проходят через класс Game, который хранит необходимые данные о полях, кораблях и тд. GameController взаимодействует с методами игры через этот класс.

Также существует метод, который реализует атаку для бота через класс Game.

Тестирование

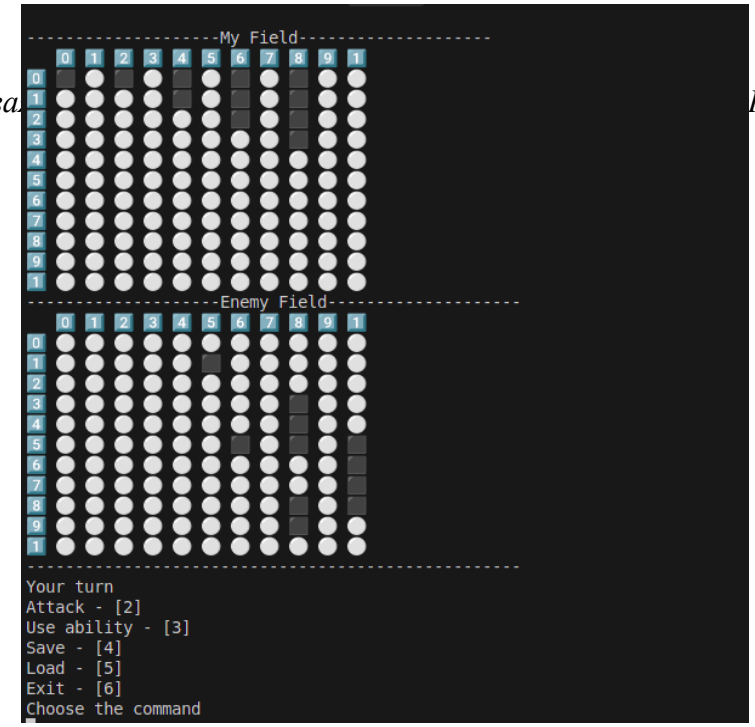
Сохранение/загрузка:

Проверка на корректность сохранения игры.

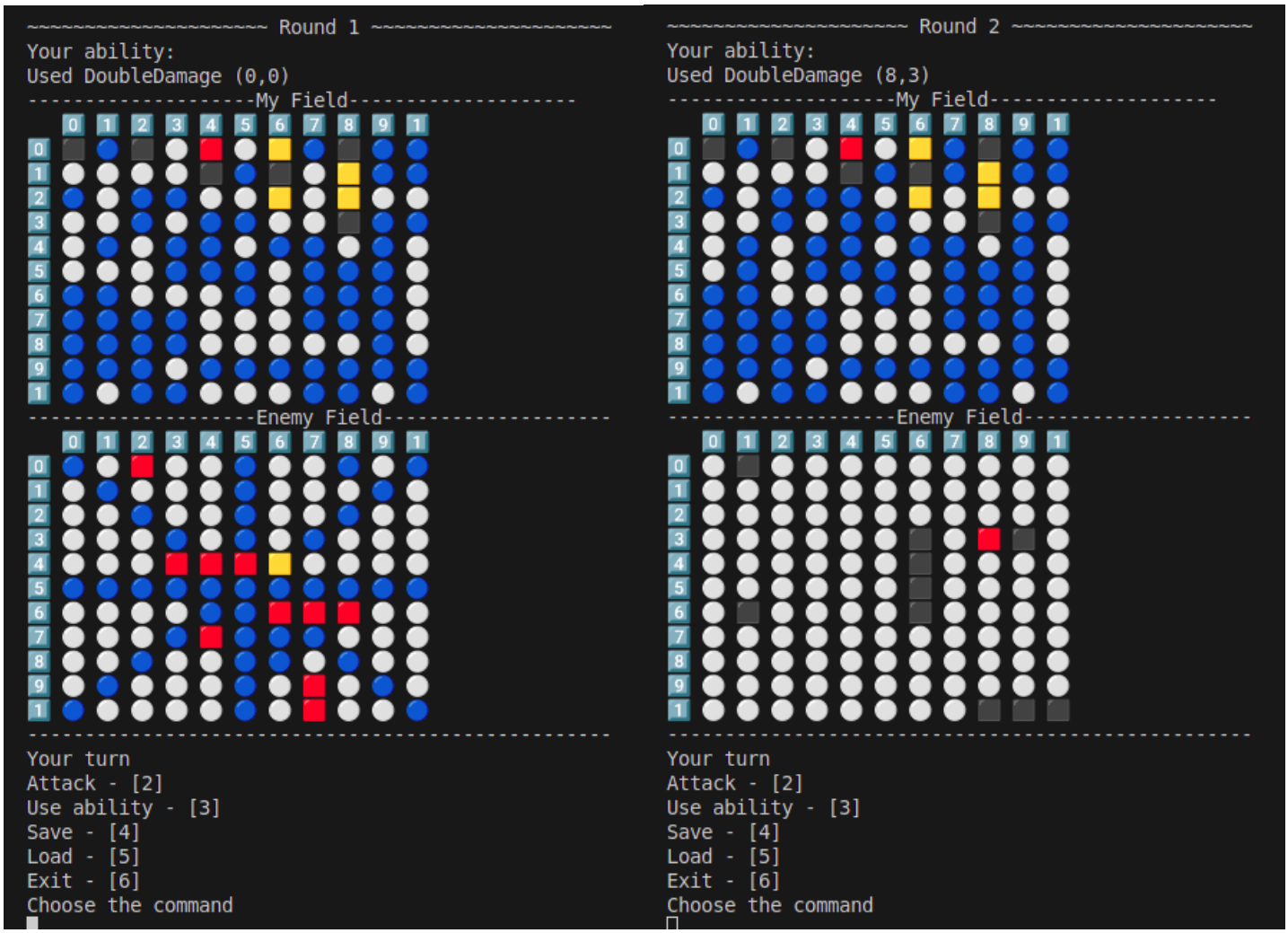
main.cpp > main() > Explain Refactor Add Docstring

```cpp
#include "Game.h"
#include "GameStates/GameState.h"
#include "GameStates/GamePlayerState.h"
#include "UserInput.h"
#include "ConsoleRenderer.h"
#include "GameDisplay.h"
#include "GameController.h"
int main()
{
    srand (time(NULL));

    Game* game = new Game();
    GameState* state = new GamePlayerState(*game);

    std::string commands_filename = "commands.json";
    UserInput input = UserInput(commands_filename);
    ConsoleRenderer renderer = ConsoleRenderer();
    GameDisplay display = GameDisplay(renderer);

    GameController controller = GameController(*game, input, display);
    controller.start(state);

    delete state;
    delete game;

    return 0;
}
```

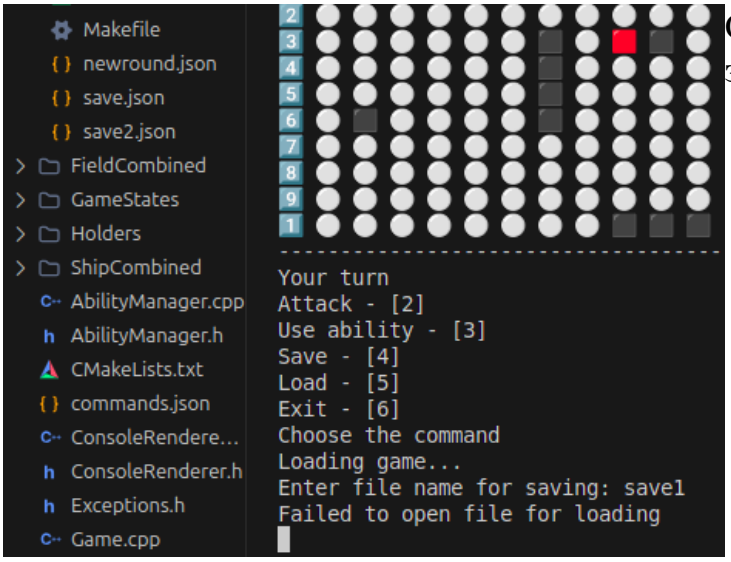
Рисунок 2 — Нова *Размещение кораблей*



Как происходит игра:



Возможные ошибки при вводе.



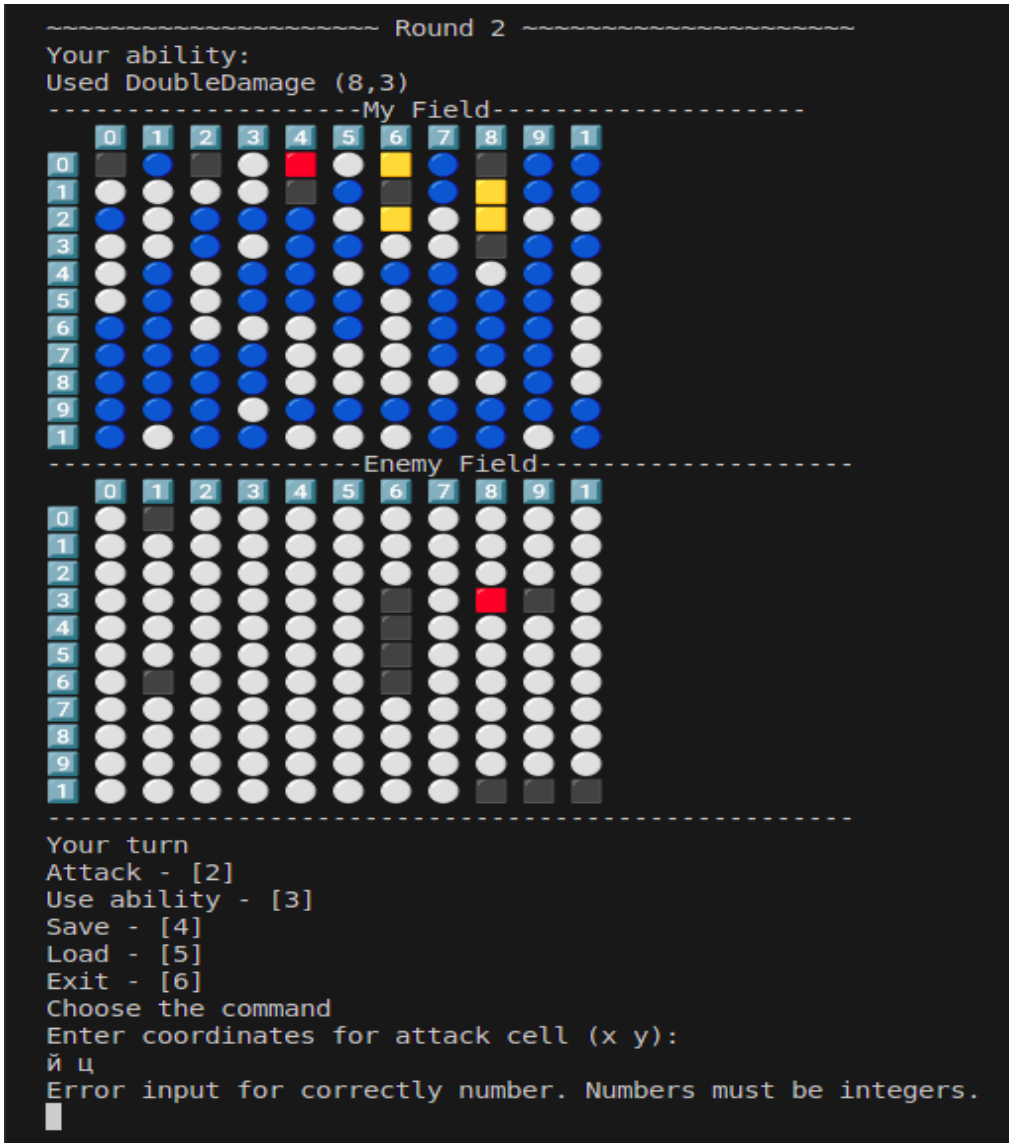
Отсутствует искомый файл для загрузки игры.

Попытка ввести некорректные координаты.
Версия 1

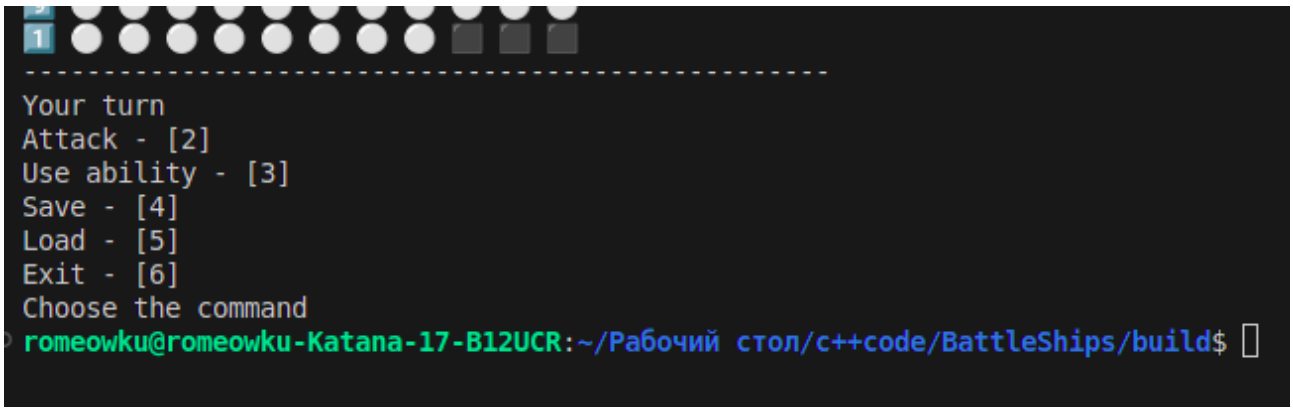
Попытка ввести некорректные координаты
Версия 2

Результат для координат (11, 11): игнорирование команды.

Результат для букв:



Выход из игры:



Выводы.

Были изучены шаблонные классы в языке программирования C++. Были созданы шаблонные классы в текущем проекте — игре «морской бой» для отображения процесса игры и взаимодействия.

UML-диаграмма реализованных классов.