МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Объектно-ориентированное программирование»
Тема: Связывание классов.

Студент гр. 3344 _____ Коршунов П.И.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы.

Связать классы игры, чтобы можно было реализовать цикл игры. Сделать реализацию сохранения и загрузки игры.

Задание.
Создать класс игры, который реализует следующий игровой цикл:

Начало игры

Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку.

Компьютерный враг только наносит атаку.

В случае проигрыша пользователь начинает новую игру

В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры.

Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

Примечание:

Класс игры может знать о игровых сущностях, но не наоборот

Игровые сущности не должны сами порождать объекты состояния

Для управления самое игрой можно использовать обертки над командами

При работе с файлом используйте идиому RAII.

Выполнение работы.

Класс ***Game***

Этот класс управляет основной логикой игры, включая инициализацию, взаимодействие между игроком и ботом, и сохранение/загрузку игры.

Методы

initialize

Выполняет начальную настройку игры. Вызывается при создании объекта Game.

setupNewGame

Подготавливает игру для нового старта. Может использоваться для перезапуска.

preparePhase(bool skipPlayer = false)

Настраивает этап подготовки (расстановка кораблей).

Если ***skipPlayer*** — пропускает действия игрока, выполняя настройку только для бота.

getShipSizesFromUser

Запрашивает у пользователя размеры кораблей.

getFieldDimensionsFromUser

Запрашивает у пользователя размеры игрового поля.

playRound

Управляет ходом игры, включая последовательность действий игрока и бота.

restartGame

Перезапускает текущую игру, сбрасывая состояние.

checkVictory

Проверяет, есть ли победитель в текущей игре.

loadGame(const std::string& filename)

Загружает сохраненное состояние игры из файла.

Класс ***GameState***

Отвечает за сохранение, восстановление и воспроизведение состояния игры.

Методы

recordCommand

Записывает действие в лог команд.

Параметры: ***who*** (кто выполнял), ***operation*** (что сделано), ***parameters*** (дополнительные данные).

getCommandLog

Возвращает лог всех команд.

saveToFile

Сохраняет состояние игры в файл.

Учитывает флаги (***attackMadeFlag, abilityUsedFlag***), активные способности и потопленные корабли.

loadFromFile

Загружает состояние игры из файла.

replayCommands

Повторяет команды из лога, чтобы восстановить состояние игры.

dropCommandLog

Очищает лог команд.

executeCommand

Выполняет конкретную команду из лога, чтобы воспроизвести действие.

friend std::ostream& operator<<(std::ostream& os, const GameState& state)

Позволяет выводить состояние игры в поток (для сохранения).

friend std::istream& operator>>(std::istream& is, GameState& state)

Позволяет загружать состояние игры из потока (при загрузке).

IPlayer

Интерфейс для всех игроков (ботов и пользователей). Определяет базовые методы, которые должны быть реализованы в подклассах.

Методы

placeShips

Расставляет корабли на поле.

makeMove

Выполняет ход игрока (атака, использование способностей и т. д.).

~IPlayer

Виртуальный деструктор.

Класс ***UserPlayer***

Реализация интерфейса ***IPlayer*** для реального пользователя.

Методы

placeShips

Позволяет пользователю вручную расставить корабли.

makeMove

Обрабатывает ход пользователя, выбор атаки или способности, загрузка или создание сохранения.

setSunkedShips(int amount)

Устанавливает количество потопленных кораблей.

getSunkedShips

Возвращает количество потопленных кораблей.

dropAllFlags

Сбрасывает флаги действий (атака, способности).

dropLoadGame

Сбрасывает строку с названием файла сохранения.

setupAttackFlag

Устанавливает флаг, что атака была выполнена.

setupAbilityFlag

Устанавливает флаг, что способность была использована.

setDamageX(int damageX)

Устанавливает множитель урона.

isLoadGame

Возвращает имя файла загруженной игры.

Класс ***BotPlayer***

Реализация интерфейса ***IPlayer*** для бота.

Методы

placeShips

Автоматически расставляет корабли бота на поле.

makeMove

Выполняет ход бота, атака.

Тестирование

```
Главное меню:
1) Начать новую игру
2) Загрузить сохранение
3) Завершить игру
Введите ваш выбор (1-3): 1
Введите ширину поля: 10
Введите высоту поля: 19
Введите количество кораблей (от 1 до 10): 1
Введите размер корабля #1 (от 1 до 4): 1
Фаза подготовки: Расставьте свои корабли.
Постановка корабля 1 размера 1
Введите координаты начала (x y): 0 0
Введите ориентацию корабля (h - горизонтальная, v - вертикальная): h
Бот расставляет свои корабли...
Фаза подготовки завершена.
Ход игрока:

Ваши опции:
1) Атаковать ячейку
2) Использовать способность
3) Сохранить игру
4) Загрузить игру
5) Вывести своё поле
6) Вывести поле противника
7) Вывести доступные способности
8) Завершить ход
9) Выйти в меню
Введите ваш выбор (1-9): 7
Доступные способности:
- RandomFire
- DoubleDamage
- Scanner
```

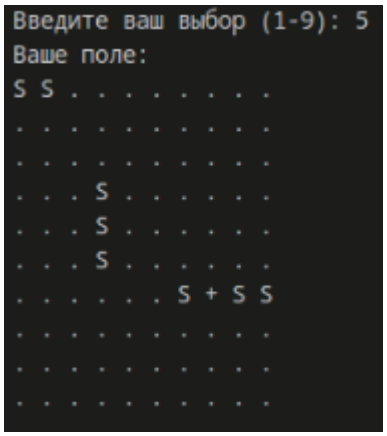
```
Введите ваш выбор (1-9): 1
Введите координаты для атаки (x y):  0 0
Нет попадания по этим координатам.
```

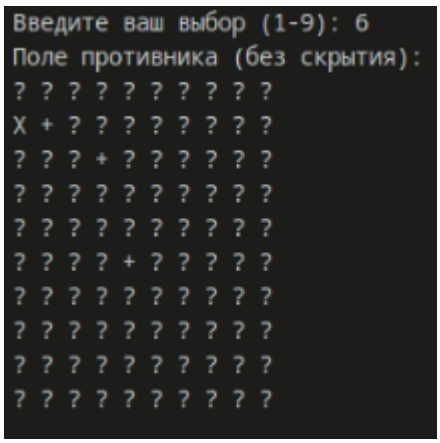
```
Введите ваш выбор (1-9): 5
Ваше поле:
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . S S S S . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
```

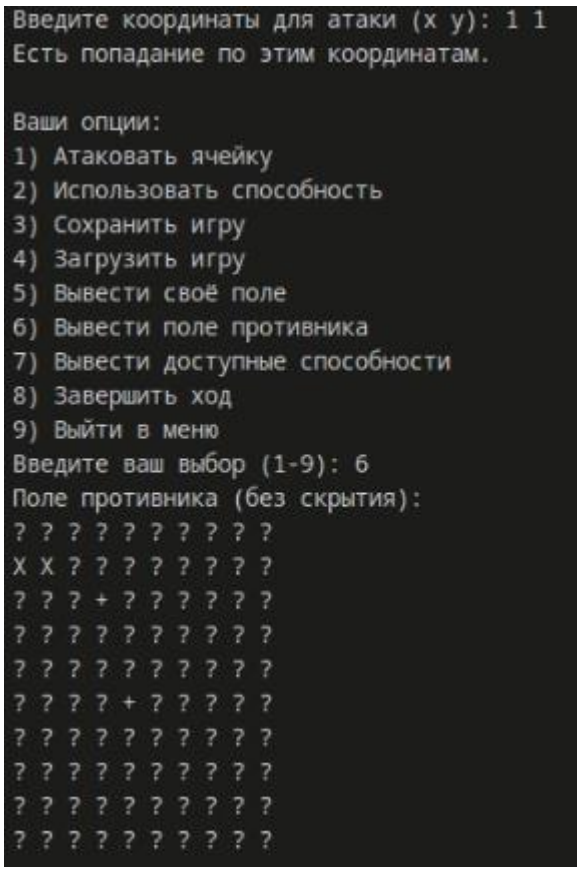
```
Введите ваш выбор (1-9): 6
Поле противника (без скрытия):
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
```

Тестирование создания поля, расстановки кораблей, атаки, вывода полей.

```
Игра загружена из файла: tst.txt
Skip set_field
Skip set_ships
Skip attackMadeFlag
Skip abilityUsedFlag
Загрузили игру из файла
Ход игрока:
```







Тестирование загрузки сохранения из файла во время раунда, проверка корректности загрузки сохранения путем атаки вражеского поля.

Выводы.

Создана архитектура, обеспечивающая модульность и расширяемость игры. Интерфейс IPlayer унифицирует действия игроков, позволяя одинаково управлять пользователем и ботом. Классы UserPlayer и BotPlayer реализуют конкретное поведение игроков, обеспечивая гибкость и возможность добавления новых типов игроков. GameState отвечает за хранение, воспроизведение и управление командами, что позволяет сохранять, загружать и отлаживать игру. Основной класс Game управляет логикой игры, взаимодействием игроков, состоянием и проверкой победы.

UML-диаграмма реализованных классов.