**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.**

Студентка гр. 3343 _____ Синицкая Д.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

**Цель работы.**

Создать класс игры, который реализует полноценный игровой цикл, позволяющий управлять игрой и обрабатывать взаимодействие между пользователем и компьютерным противником.

**Задание.**

a) Создать класс игры, который реализует следующий игровой цикл:

i) Начало игры

ii) Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.

iii) В случае проигрыша пользователь начинает новую игру

vi) 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.

b) 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 *GameSetup* — отвечает за начальную настройку игры, включая параметры игрового поля и кораблей. Используется *GameSession* для получения настроек.

Методы:

- *read_start_data()* — взаимодействует с пользователем для ввода параметров игры.
- *get_field_size(), get_ships_count(), get_sizes()* — предоставляют доступ к параметрам настройки игры.
- *serialize()* — сохраняет состояние игры.
- *deserialize()* — загружает состояние игры.

Класс *GameSession* — реализует игровой процесс, логику ходов, стрельбы, обновления состояния игроков и использование способностей. Работает с *AbstractPlayer* (игрок и противник). Использует *GameSetup* для конфигурации параметров игры. Взаимодействует с *AbilitiesManager* для управления способностями игроков.

Методы:

- *run_game_step()* — управляет логикой одного хода игры.
- *place_ships()* — организует размещение кораблей для игроков.
- *execute_shot()* — обрабатывает выстрел одного игрока по другому.
- *serialize()* — сохраняет состояние игры.
- *deserialize()* — восстанавливает состояние игры.
- *use_ability()* —активирует способности через AbilitiesManager.

Класс *GameState* — обеспечивает восстановление игры после её завершения или прерывания. Работает с *GameSession* для сохранения/восстановления игры.

Методы:

- *save(const char* filename)* — сохраняет состояние игры в файл.
- *load(const char* filename)* — загружает состояние игры из файла.
- Перегрузки *operator<<* и *operator>>* для потоковой сериализации.

Взаимосвязь классов.

*GameSession* и *GameSetup*:

Класс *GameSession* зависит от *GameSetup*, настройки игры передаются в конструктор *GameSession*. Класс *GameSetup* предоставляет информацию о размере поля, количестве кораблей и их размерах, которые потом используются для создания объектов игроков.

*GameSession* и *AbilitiesManager*:

*GameSession* содержит ссылку на *AbilitiesManager*, что позволяет менеджеру способностей взаимодействовать с состоянием противника и вызывать способности, влияющие на игру.

*GameSession* и *Player/Enemy*:

Игроки в игре реализованы через классы *Player* и *Enemy*, которые абстрактируются через *AbstractPlayer*, что позволяет поддерживать общую логику для игроков. Класс *GameSession* управляет очередностью ходов и обработкой выстрелов между игроками.

**Выводы.**

В ходе выполнения лабораторной работы был реализован класс игры, который управляет игровым циклом, охватывающим все основные этапы взаимодействия игрока с компьютерным противником.

## ПРИЛОЖЕНИЕ А
## ТЕСТИРОВАНИЕ

Входные данные: поле 3*3, 1 корабль длины 2 расположение вертикальное по координатам C1 C2. Игрок ход на A1, противник ход на B2.

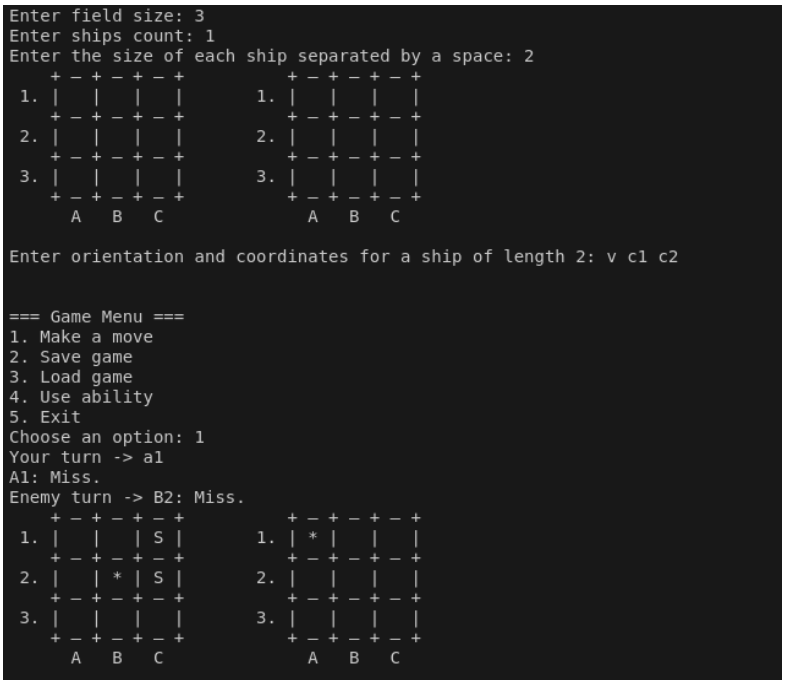


Рисунок 1 — результат тестирования.

Входные данные: 2.

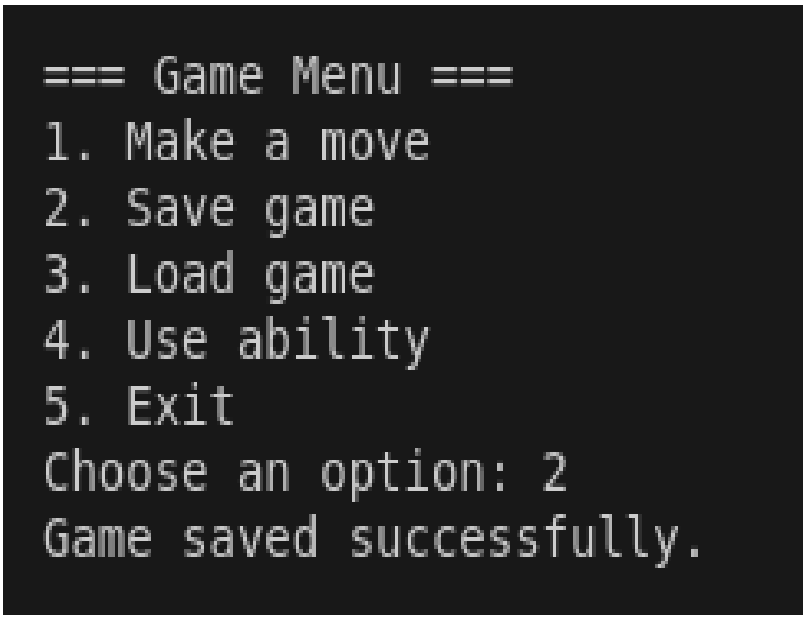


Рисунок 2 — результат тестирования.

Входные данные: 3.

```
=== Game Menu ===
1. Make a move
2. Save game
3. Load game
4. Use ability
5. Exit
Choose an option: 3
Game loaded successfully.
```

Рисунок 3 — результат тестирования.

Входные данные: 4.

```
=== Game Menu ===
1. Make a move
2. Save game
3. Load game
4. Use ability
5. Exit
Choose an option: 4
Now using: Shelling ability.
A2: Target hit.

=== Game Menu ===
1. Make a move
2. Save game
3. Load game
4. Use ability
5. Exit
Choose an option: 1
Your turn -> a2
A2: Segment destroyed.
Enemy turn -> B1: Miss.
    + – + – + – +          + – + – + – +
 1. |   | * | S |       1. |   |   |   |
    + – + – + – +          + – + – + – +
 2. |   |   | S |       2. | X |   |   |
    + – + – + – +          + – + – + – +
 3. |   |   |   |       3. |   |   |   |
    + – + – + – +          + – + – + – +
      A   B   C              A   B   C
```

Рисунок 4 — результат тестирования.

# ПРИЛОЖЕНИЕ В

# UML-ДИАГРАММА