**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 |  | Охрименко Д.И. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы.

Организовать связь между классами для реализации игрового процесса. Сохранить игру в файл для дальнейшего продолжения.

**Задание.**

Создать класс игры, который реализует следующий игровой цикл:

* 1. Начало игры
  2. Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
  3. В случае проигрыша пользователь начинает новую игру
  4. 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.

1. 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 ***Ship:***

Метод *Void* ***Restore ():*** Восстанавливает здоровье сегмента корабля.

Обновлениекласса ***Map:***

***SetSize*:**

Этот метод устанавливает размеры карты, принимая высоту и ширину в качестве параметров.

***GetCellState*:**

Метод получает состояние ячейки на координатах (x, y) в карте.

***SetCellState*:**

Этот метод устанавливает состояние ячейки на координатах (x, y).

***Clear*:**

Метод очищает карту, устанавливая для каждой ячейки состояние unknown.

Обновлениекласса ***ShipManager***

**Новые методы класса**

***ClearShips***

Этот метод отвечает за очистку всех кораблей, которые хранятся в массиве ship\_array.

***RestoreShips***

метод восстанавливает состояние всех кораблей, находящихся в массиве ship\_array.

Класс **Game** представляет собой основной класс игры "Морской бой". Он управляет всеми аспектами игры, включая инициализацию, ход игры, сохранение и загрузку состояния игры.

Публичные методы

***Game*():** Конструктор класса, вызывает метод initializeGame() для инициализации игры.

**~*Game*():** Деструктор класса, вызывает метод cleanUp() для освобождения памяти.

***cleanUp*():** Освобождает память, выделенную для объектов игры.

***initializeGame*():** Инициализирует игру, создавая необходимые объекты и устанавливая начальные значения.

***gameStart*():** Начинает игру, выводя сообщение о начале игры и вызывая метод **saveWithArrangement()** для размещения кораблей игрока.

***gameOver*():** Завершает игру, выводя сообщение о завершении игры.

***Round*():** Управляет ходом игры, обрабатывая ходы игроков и проверяя условия завершения раунда.

***nextRound*():** Переходит к следующему раунду, восстанавливая корабли противника и начиная новый раунд.

***newGame*():** Запрашивает у игрока, хочет ли он начать новую игру, и выполняет соответствующие действия.

***saveWithArrangement*():** Позволяет игроку разместить свои корабли, сохранить или загрузить игру, или продолжить игру.

***printMaps*():** Выводит карты игрока и противника.

***saveGame*():** Сохраняет состояние игры в файл.

***loadGame*():** Загружает состояние игры из файла.

Класс **Serializable** предназначен для сериализации и десериализации состояния игры в формате JSON. Этот класс используется для сохранения и загрузки игры, что позволяет игрокам продолжить игру с того места, на котором они остановились.

Конструктор инициализирует объект **Serializable**, принимая указатели на объекты Map, ShipManager, AbilityManager и CoordHolder.

Метод ***serialize***() выполняет сериализацию состояния игры в формат JSON. Он создает объект JSON и заполняет его данными о кораблях, карте и способностях.

Метод ***deserialize***() выполняет десериализацию состояния игры из формата JSON. Он принимает объект JSON и восстанавливает состояние игры на основе данных из этого объекта.

класс ***GameState***, который отвечает за управление состоянием игры, включая сохранение и загрузку данных о состоянии игры.

**Save** Этот метод создает JSON-объект, который содержит состояние игры для обоих игроков.

**Load**  Метод загружает состояние игры из указанного файла.

Он читает данные из файла и десериализует их обратно в объекты игры с помощью метода deserialize() класса Serializable.

Операторы ввода/вывода

operator<<

• Этот оператор позволяет выводить состояние игры в поток (например, в консоль).

• Он создает JSON-объект с текущим состоянием игры и выводит его с отступами для удобства чтения.

Operator>>

• Этот оператор позволяет загружать состояние игры из потока.

• Он читает данные из потока и десериализует их в объекты состояния игры.

**class *AlreadyAttackedException* : public *IException*** Реализует исключения, связанные с повторной атакой по клетке

Класс **Player** является абстрактным классом, который определяет основные методы и свойства для всех типов игроков (человека и компьютера).

Защищенные члены:

Map\* *map*: Указатель на объект карты игрока.

ShipManager\* *shipManager*: Указатель на объект менеджера кораблей игрока.

GameState\* *gameState*: Указатель на объект состояния игры.

Публичные методы:

***Player***(Map\* map, ShipManager\* shipManager, GameState\* gameState): Конструктор, инициализирующий указатели на карту, менеджер кораблей и состояние игры.

virtual void ***placeShips***() = 0: Чисто виртуальный метод для размещения кораблей. Реализуется в производных классах.

virtual void ***makeMove***(Map\* opponentMap, AbilityResponce\* responce, AbilityManager\* skillManager) = 0: Чисто виртуальный метод для совершения хода. Реализуется в производных классах.

bool ***allShipsDestroyed***(): Метод, проверяющий, уничтожены ли все корабли игрока.

void ***saveGame***(): Метод для сохранения игры.

void ***loadGame***(): Метод для загрузки игры.

Класс ***HumanPlayer*** наследуется от Player и представляет собой игрока-человека.

Публичные методы:

***HumanPlayer***(Map\* map, ShipManager\* shipManager, GameState\* gameState): Конструктор, вызывающий конструктор базового класса.

void ***placeShips***() override: Переопределенный метод для размещения кораблей. Игрок размещает корабли вручную.

void ***makeMove***(Map\* opponentMap, AbilityResponce\* responce, AbilityManager\* skillManager) override: Переопределенный метод для совершения хода. Игрок может атаковать, использовать способность, сохранить или загрузить игру.

Приватные методы:

void ***myAttack***(Map\* opponentMap, AbilityResponce\* responce, AbilityManager\* skillManager): Метод для атаки противника. Игрок вводит координаты для атаки.

void ***useAbility***(AbilityManager\* skillManager, AbilityResponce\* responce): Метод для использования способности. Игрок активирует способность, если она доступна.

Класс **ComputerPlayer** наследуется от Player и представляет собой игрока-компьютера.

Публичные методы:

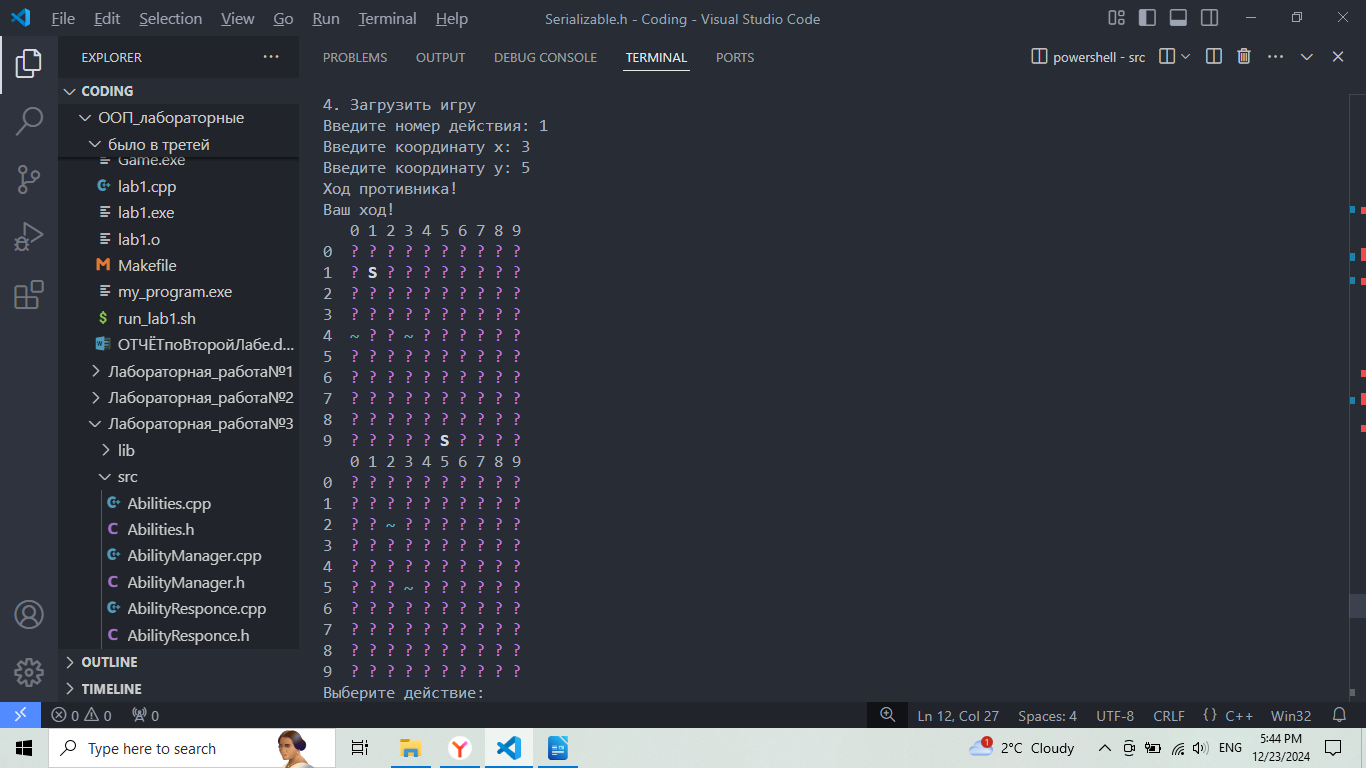
***ComputerPlayer***(Map\* map, ShipManager\* shipManager, GameState\* gameState): Конструктор, вызывающий конструктор базового класса.

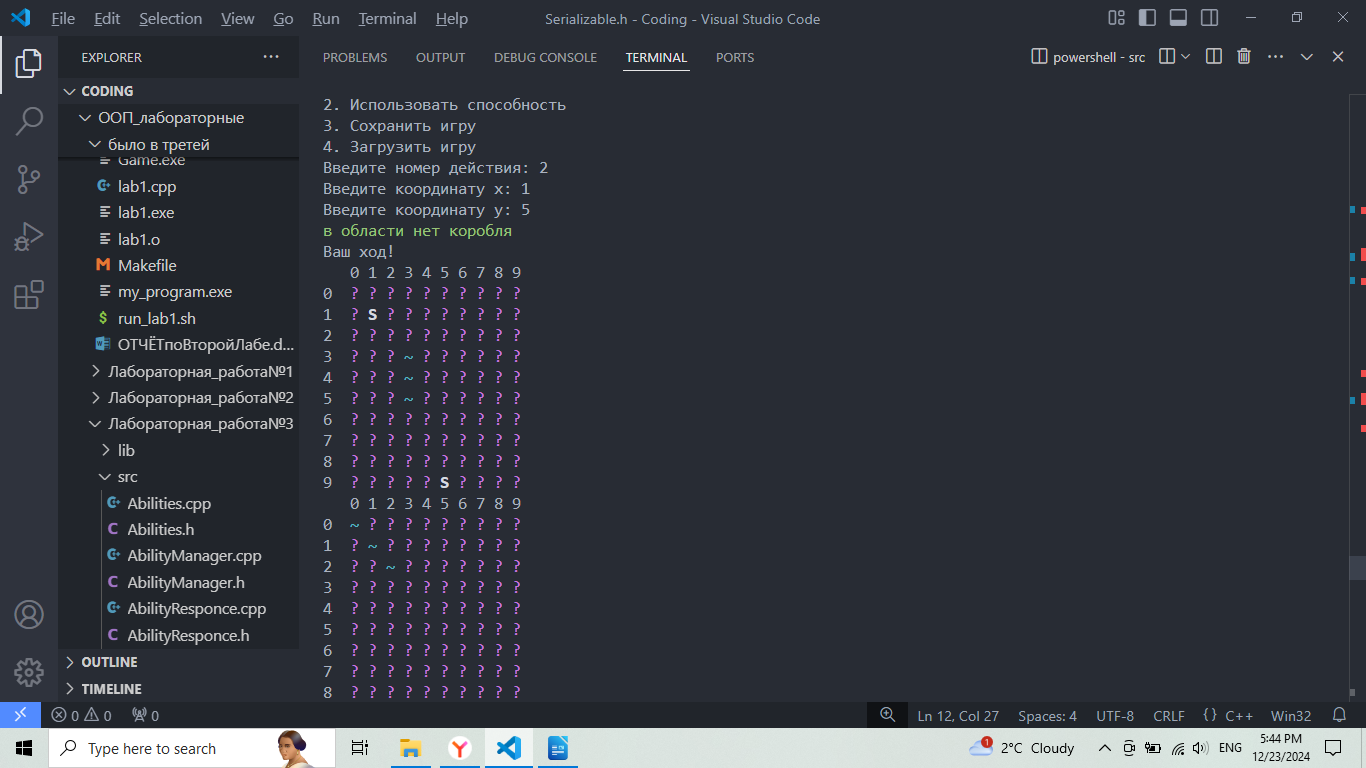
void ***placeShips***() override: Переопределенный метод для размещения кораблей. Компьютер размещает корабли случайным образом.

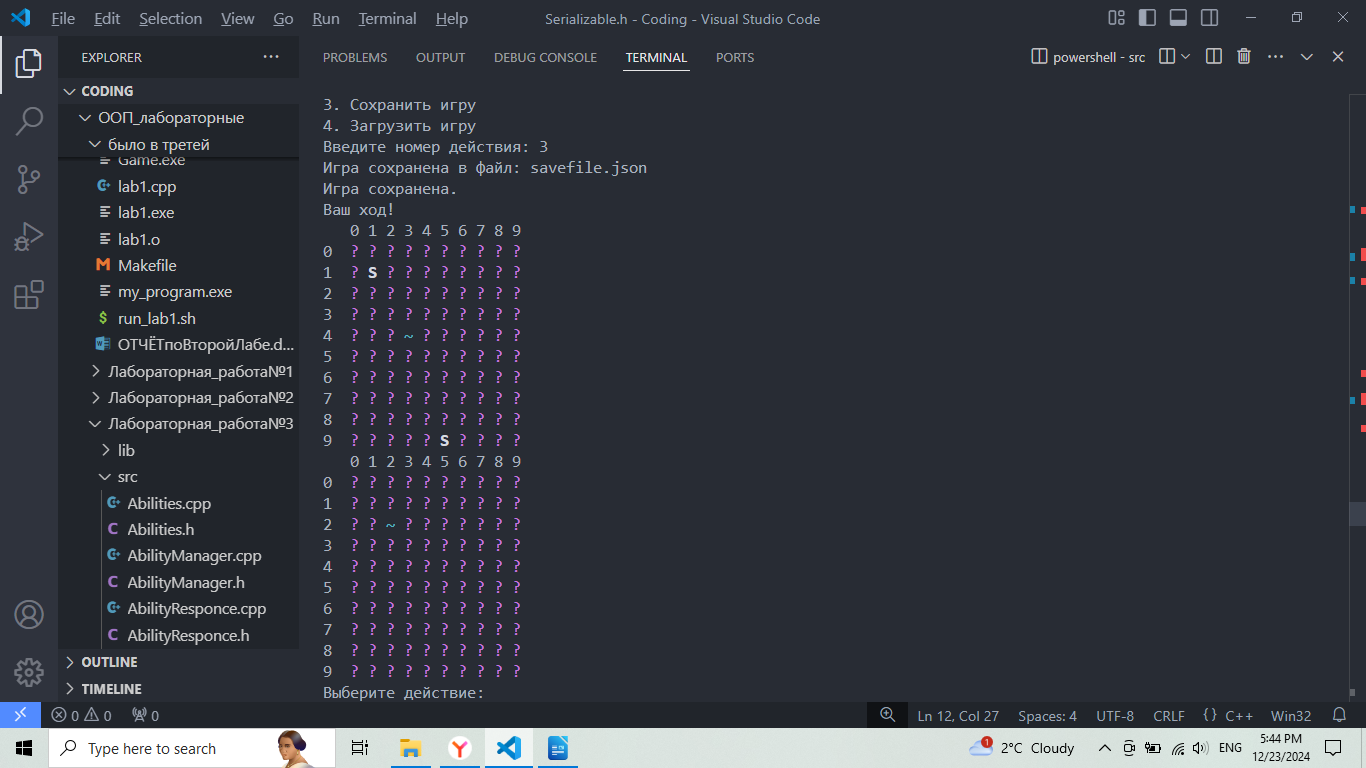
Void ***makeMove***(Map\* opponentMap, AbilityResponce\* responce, AbilityManager\* skillManager) override: Переопределенный метод для совершения хода. Компьютер атакует случайные клетки на карте противника.

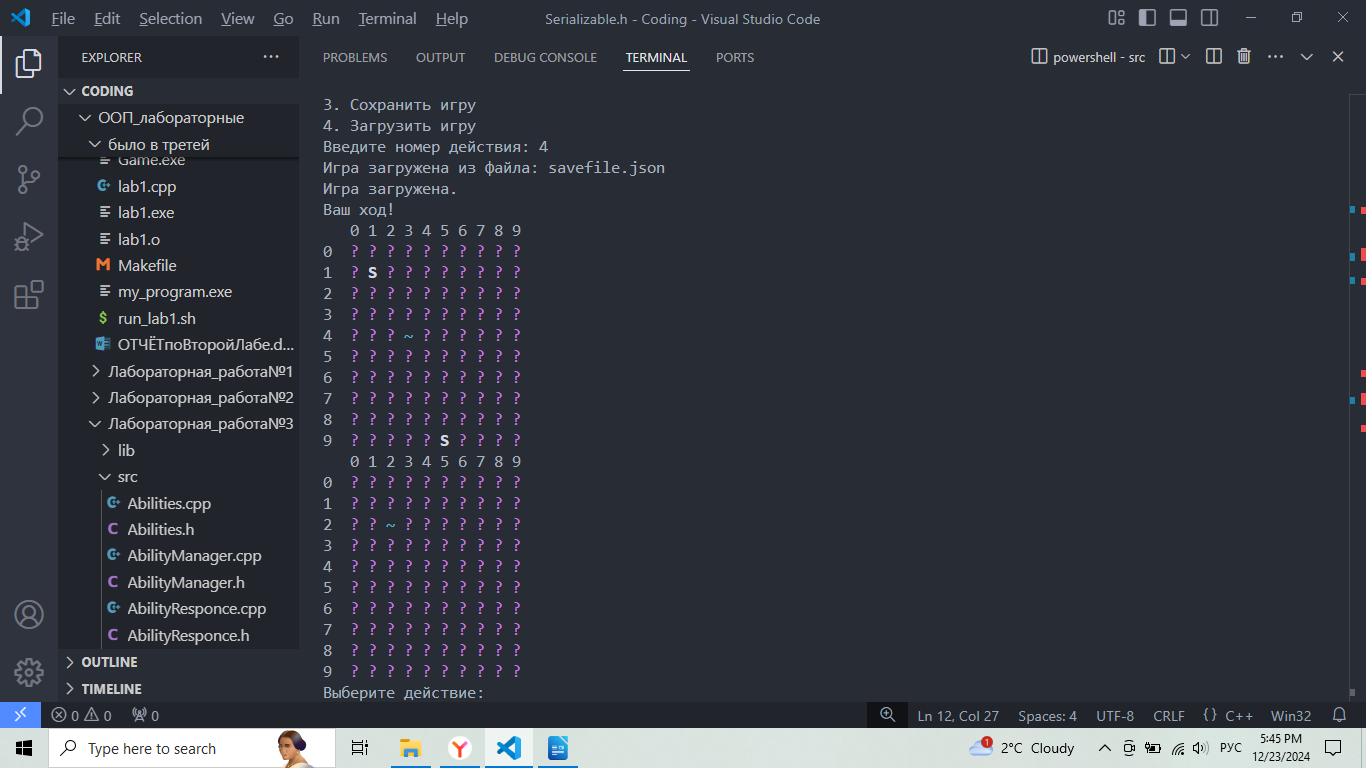
**Тестирование**

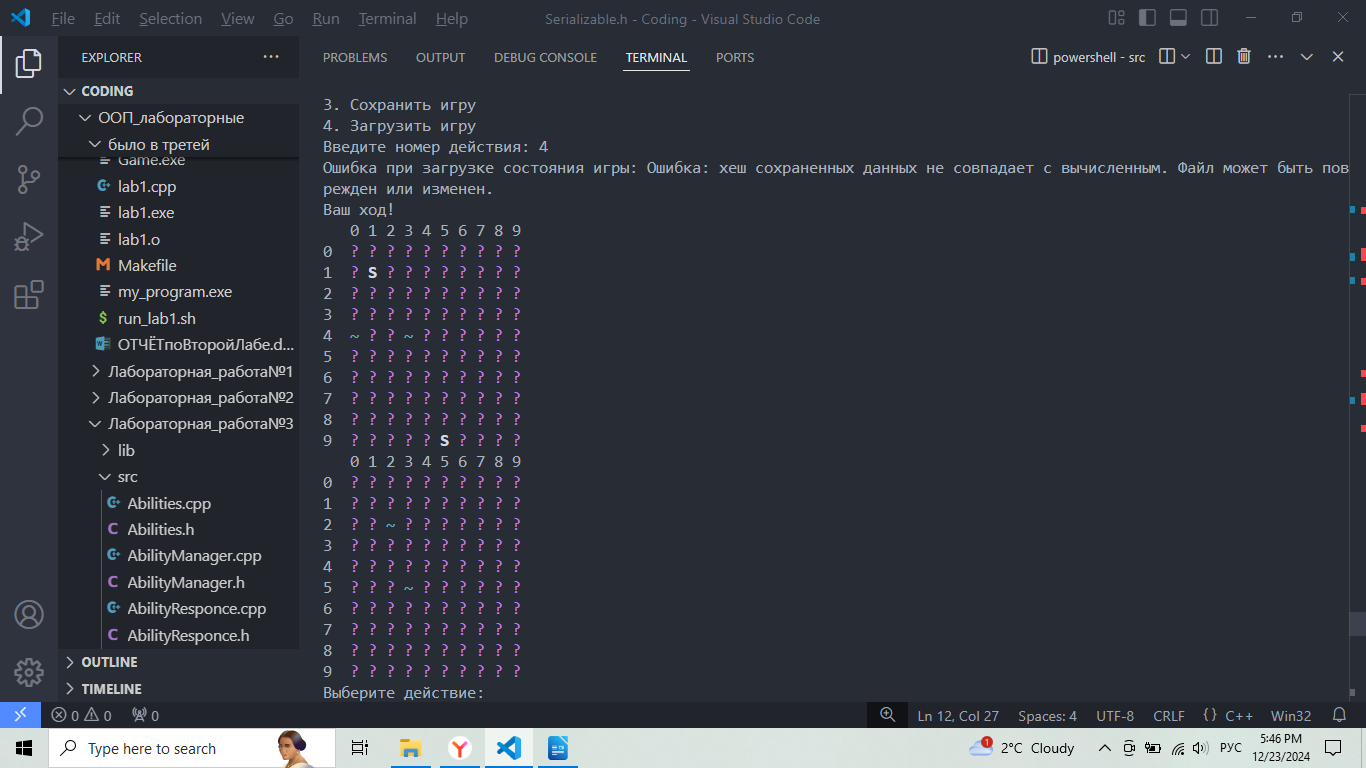
Проведём тестирование игрового процесса и предоставим результаты в виде скриншотов из консоли.

Рисунок 1 — *игровой процесс*

Рисунок 2 — *Активация способности*

Рисунок 3 —*сохранение игры*

Рисунок 4 —*загрузка игры*

Рисунок 5 —*если изменить файл* json *то загрузиться не удастся*

## Выводы.

Получен опыт в связывании классов, что позволило значительно улучшить структуру и управляемость кода. Исследованы способы работы с файлами.

***UML*-диаграмма реализованных классов.**

