Лабораторная работа №6 «Морской бой»

Постановка задачи

Цель:

Разработка системы классов для реализации игры "Морской бой" между человеком и компьютером.

Основные требования:

1. **Размер поля:** 10×10 клеток.
2. **Корабли:**
    - 4 однопалубных, 3 двухпалубных, 2 трёхпалубных, 1 четырёхпалубный для каждого игрока.
3. **Правила расстановки:**
    - Многопалубные корабли — только по горизонтали/вертикали.
    - Запрет на соседство кораблей (включая диагонали).
4. **Ход игры:**
    - Первый ход за человеком.
    - При попадании ход остаётся у атакующего.
    - При убийстве корабля (полном поражении) — обязательное уведомление.
5. **Условие победы:** уничтожение всех кораблей противника.

Описание класса Ship

Назначение:

Моделирует корабль в игре, хранит его состояние (координаты, попадания) и проверяет условия уничтожения.

Методы:

1. **Конструктор** Ship(int sz, const vector<Coord>& places)
    - **Параметры:**
        - sz — размер корабля (1–4).
        - places — вектор координат клеток корабля.
    - **Функционал:** инициализирует размер, координаты и массив попаданий (hits).
2. bool isAt(const Coord& c)
    - **Параметры:** c — проверяемая координата.
    - **Возвращает:** true, если корабль занимает клетку.
    - **Сложность:** O(n), где n — размер корабля.
3. bool hit(const Coord& c)
    - **Параметры:** c — точка попадания.
    - **Возвращает:** true, если попадание зарегистрировано.
    - **Функционал:** помечает клетку как поражённую.
4. bool isKilled()
    - **Возвращает:** true, если все клетки корабля поражены.
    - **Сложность:** O(n), где n — размер корабля.

Описание класса Field

Назначение:

Управляет игровым полем: хранит состояние клеток, корабли, выстрелы, реализует логику расстановки и стрельбы.

1. Конструктор Field()

Создает пустое поле 10×10, где все клетки изначально помечены как EMPTY ('.').

Основные методы управления игровым полем:

2. void markCell(const Coord& c, Cell cellValue)

- **Параметры:**
  - c – координата клетки (row, col).
  - cellValue – новое значение (EMPTY, SHIP, HIT, MISS или KILLED).
- **Функционал:** Изменяет состояние указанной клетки.

3. void print() const

- **Функционал:** Выводит текущее состояние поля в консоль (номера строк/столбцов + символы клеток).

Методы для работы с кораблями:

4. bool inBounds(const Coord& c) const

- **Параметры:** Координата c.
- **Возвращает:** true, если координата внутри поля ($0 \le row, col \le 9$).

5. bool isFree(const vector<Coord>& positions) const

- **Параметры:** Вектор координат (positions).
- **Возвращает:** false, если клетки уже заняты кораблём (SHIP) или рядом есть корабль.
- **Сложность:** O(n × 9), где n – размер корабля (9 - проверка 8 соседних клеток + сама клетка).

6. bool placeShip(int size, Coord start, Direction dir)

- **Параметры:**
  - size – размер корабля.
  - start – начальная координата.

    - dir (HORIZONTAL/VERTICAL) – направление.
- **Возвращает:** false, если корабль нельзя разместить.
- **Функционал:**
    - Проверяет, можно ли разместить корабль через isFree.
    - Добавляет корабль в ships и помечает клетки на поле как SHIP.

Методы для генерации расположения кораблей:

7. void randomPlacement()

- **Функционал:** Автоматически расставляет корабли случайным образом (для компьютера).
- **Алгоритм:** Пытается разместить корабли (4×1, 3×2, 2×3, 1×4) в случайных позициях.

8. bool manualPlacement()

- **Функционал:** Запрашивает у игрока ввод кораблей через консоль.

Методы для обработки выстрелов:

9. pair<bool, string> makeShot(const Coord& c)

- **Параметры:** Координата выстрела (c).
- **Возвращает:**
    - pair.first (bool) – попадание.
    - pair.second (string) – сообщение:
        - "Uncorrect coordinates!" – неверные координаты.
        - "You've already shot here!" – повторный выстрел.
        - "Hit!" / "Hit! Ship was sunk!" – попадание или убийство корабля.

- "Miss!" – промах.

- **Логика:**

1. Проверяет, валидны ли координаты и можно ли стрелять.
2. Если это корабль (SHIP), меняет состояние клетки на HIT или KILLED.
3. Если корабль уничтожен (isKilled), все его клетки помечаются как KILLED.

10. bool allShipsKilled() const

- **Возвращает:** true, если все корабли на поле потоплены.

11. bool alreadyShot(const Coord& c) const

- **Возвращает:** true, если в эту клетку уже стреляли (HIT, MISS или KILLED).

12. const Ship* getShipByCoord(const Coord& c) const

- **Параметры:** Координата c.
- **Возвращает:** Указатель на корабль, если он есть в клетке, иначе nullptr.

Описание класса Player

Назначение:

Хранит данные игрока: своё поле (ownField) и поле противника (enemyField) для отметки выстрелов.

Описание класса Game

Назначение:

Управляет процессом игры: настройка, очерёдность ходов, проверка победы.

Ключевые методы:

1. void setup()
    - **Функционал:**

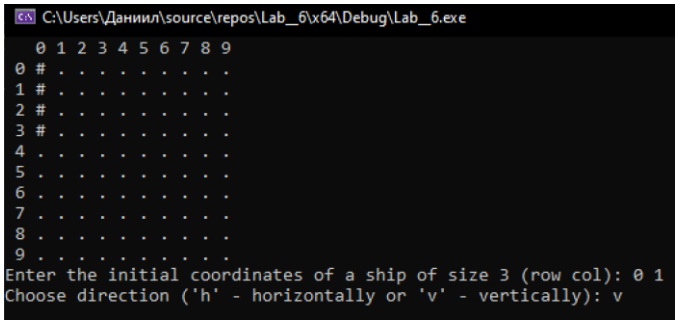
- Ручная расстановка кораблей игроком (manualPlacement).
- Автоматическая расстановка для компьютера (randomPlacement).

2. Coord computerTurn()
   - **Возвращает:** случайную валидную координату для выстрела.
   - **Сложность:** O(k), k – неизвестно, поскольку компьютер может случайно выстрелить в одну и ту же клетку повторно.
3. void play()
   - **Функционал:**
     - Цикл поочерёдных ходов.
     - Обработка выстрелов (makeShot), переход хода при промахе.
     - Проверка победы (allShipsKilled).

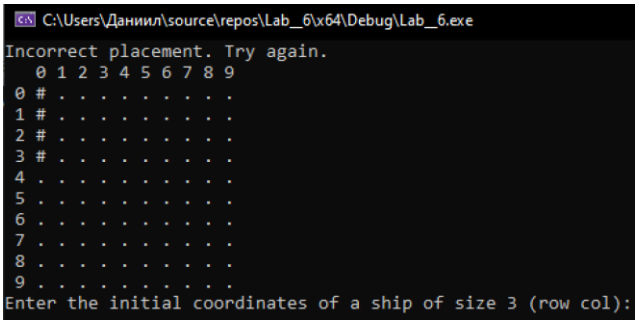
Корректность работы программы

Рассмотрим несколько крайних случаев, которые могут вызвать ошибку в программе.

1. **Нарушение правил соседства кораблей.**
   При попытке разместить корабли слишком близко, программа сообщает пользователю о некорректном расположении корабля и предоставляет пользователю повторную попытку.



Попытка некорректного размещения корабля

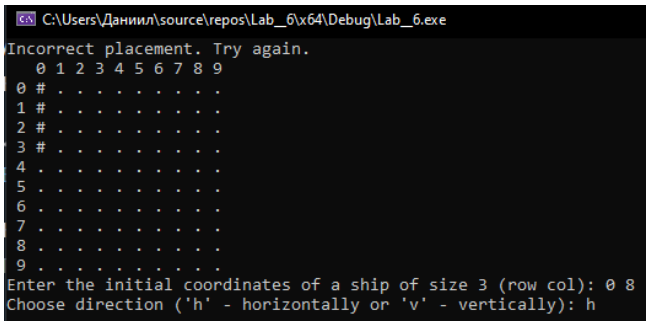


Сообщение программы о некорректных данных

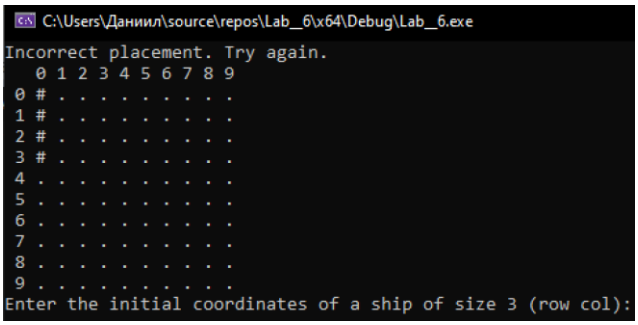
2. **Обработка выстрела за границы поля.**

   Как таковая ошибка при выстреле за пределы координат поля отсутствует. Вместо этого программа считывает первые 2 числа, которые встречаются ей в буфере ввода. Например, если при запросе координат пользователь введёт «Абв-1 !г234» или «12 13», то программа примет координаты (1, 2). После считывания координат буфер ввода будет очищен.

   Также присутствует обработка ввода направления корабля. Если пользователь вводит некорректное значение (например, ошибается буквой) то для корабля автоматически устанавливается горизонтальное направление.

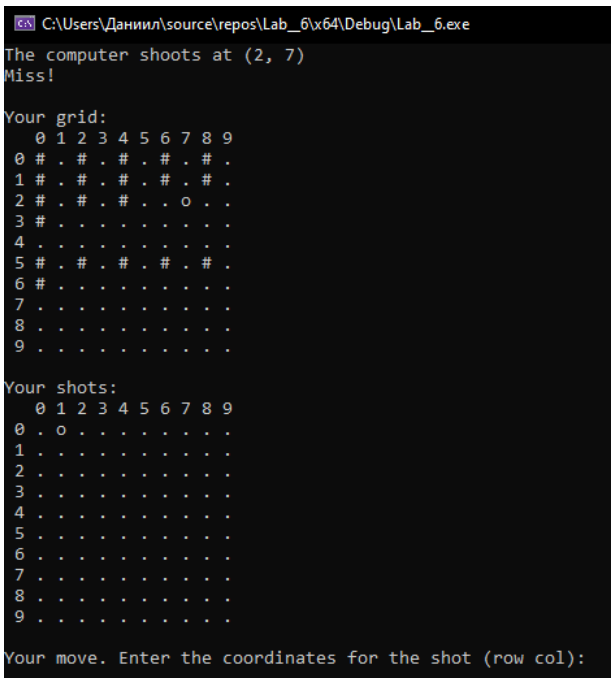


Попытка некорректного размещения корабля (часть корабля за пределами поля)



Сообщение программы о некорректных данных

Попытка ввести координаты выстрела (0, 10)



Обработка некорректных координат (0, 10) как (0, 1)