Лабораторная работа № 4

Тема: Многомерные массивы

Цель работы:
Научиться работать с многомерными массивами в языке C#, применять их для решения сложных задач и создания игровых приложений с интерактивным взаимодействием через консоль. Студенты должны научиться реализовывать игры с использованием многомерных массивов для хранения игрового состояния, управлять вводом пользователя, а также разрабатывать логику обработки данных для проверки условий победы и завершения игры.

Задание:
Создайте новое консольное приложение на языке C# на основе кода лабораторной работы № 3, которое выполняет следующие действия:

1. Меню выбора действий:
    - Добавьте новый пункт меню “4. Игра”.

2. Требование к игре:
    - Создайте консольное приложение на языке C#, которое реализует одну из предложенных игр с использованием многомерных массивов (по вариантам). Приложение должно обеспечивать корректное взаимодействие с пользователем, позволяя ему управлять игровым процессом, а также проверять выполнение условий победы или поражения.

3. Игровое поле:
    - Должно быть представлено в виде двумерного или трёхмерного массива.
    - Поле и его состояние должны обновляться после каждого хода игрока.

4. Взаимодействие с пользователем:
    - Игрок вводит данные через консоль, управляя процессом игры (например, перемещает фигуры, вводит координаты).
    - Необходимо реализовать проверку на корректность ввода данных.

5. Логика игры:
    - Программа должна проверять условия окончания игры: победа, поражение или ничья (если применимо).
    - По завершении игры необходимо выводить соответствующее сообщение.

6. Разбиение программы на методы. Логика игры должна быть структурирована и разбита на отдельные методы:
    - Метод для инициализации игрового поля.
    - Метод для вывода текущего состояния поля на экран.
    - Метод для ввода данных пользователем.
    - Метод для обработки хода и обновления состояния игры.

- Метод для проверки условий окончания игры (победа/поражение).

7. Дополнительные требования (их реализовывать не обязательно):
    - Для каждой игры необходимо реализовать несколько уровней сложности (например, разные размеры поля или количество допустимых ошибок).

Варианты заданий:

№	Вариант задания	Описание
1	Морской бой	**1. Цель** Создать консольное приложение на языке C#, моделирующее игру «Морской бой», в которой игрок и компьютер пытаются потопить корабли друг друга на поле 5x5. **2. Игровое поле** ● Размер игрового поля — двумерный массив 5x5, где каждая клетка представляет собой пустое пространство или часть корабля. ● Каждая клетка поля отображает текущий статус: пустая клетка, корабль, попадание или промах. **3. Типы кораблей** ● Однопалубный корабль (занимает 1 клетку). ● Двухпалубный корабль (занимает 2 клетки). ● Трёхпалубный корабль (занимает 3 клетки). **4. Размещение кораблей** ● **Игрок**: размещает свои корабли вручную, вводя координаты для каждого корабля. ● **Компьютер**: размещает свои корабли случайным образом на игровом поле, с проверкой, чтобы корабли не пересекались и не выходили за пределы поля. **5. Игровой процесс** 1. Игрок и компьютер поочередно делают ходы, стреляя по полю противника. 2. Игрок вводит координаты выстрела, которые программа преобразует в индексы массива. 3. Программа проверяет, было ли попадание в корабль или промах, и отображает соответствующий результат. 4. После каждого хода поле обновляется, показывая статус клеток: попадание или

		промах. 5. Программа должна корректно обрабатывать как успешные выстрелы (попадания), так и неудачные (промахи). **6. Победа** ● Игра завершается, когда все корабли одного из игроков уничтожены. ● Победителем становится тот, кто первым потопит все корабли противника. **8. Дополнительные функции (обязательно)** ● Обработка некорректного ввода (например, повторные или несуществующие координаты). ● Возможность повторного хода при попадании.
2	Тетрис	**1. Цель** Создать консольное приложение на языке C#, моделирующее игру «Тетрис», в которой игрок управляет падающими блоками на поле размером 10x10, стараясь не допустить, чтобы блоки заполнили игровое пространство до верхней границы. **2. Игровое поле** ● Поле представляет собой двумерный массив размером 10x10, где каждый элемент массива — это клетка, которая может быть либо пустой, либо заполненной частью блока. ● Поле обновляется с каждым шагом, отображая движение падающего блока и зафиксированные блоки на месте. **3. Фигуры** ● Блоки различных форм (например, линии, квадраты, L-образные) падают сверху поля. ● Каждая фигура начинает падение с верхней части поля и движется вниз с шагом. **4. Управление блоками** ● Игрок управляет падением блоков, перемещая их влево и вправо с помощью клавиш ввода. ● Блоки продолжают падение до тех пор, пока не достигнут дна поля или не столкнутся с другим зафиксированным блоком.

		5. Фиксация блоков ● Когда блок касается дна игрового поля или другого зафиксированного блока, он прекращает падение и остаётся на месте. ● После этого появляется новая фигура, которая начинает падать с верхней части поля. **6. Удаление рядов** ● Если какой-либо ряд поля полностью заполняется блоками, этот ряд удаляется, а все ряды выше смещаются вниз на одну строку. ● Удаление ряда приносит игроку очки и освобождает пространство для новых блоков. **7. Конец игры** ● Игра завершается, если зафиксированные блоки достигают верхней границы поля, и новый блок больше не может появиться на поле. ● Цель игрока — максимально долго удерживать игру, не давая блокам достичь верха. **8. Дополнительные функции (обязательно)** ● Подсчёт очков за каждый удалённый ряд. ● Обработка ввода для управления движением и поворотом блоков.
3	Сапёр	**1. Цель** Создать консольное приложение на языке C#, моделирующее игру «Сапёр», в которой игрок должен открыть все безопасные клетки поля, избегая мин. **2. Игровое поле** ● Поле представляет собой двумерный массив размером 5x5, где каждая клетка может быть либо пустой (безопасной), либо заминированной. ● Всего на поле случайным образом размещены 5 мин. ● Игровое поле обновляется после каждого хода, отображая уже открытые клетки и количество мин вокруг открытых безопасных клеток.

		3. Размещение мин ● 5 мин случайным образом размещаются на поле при старте игры. ● Мины не могут размещаться на одной клетке и не должны быть видны до их открытия. **4. Игровой процесс** 1. Игрок вводит координаты клетки, которую хочет открыть (например, B3). 2. Программа проверяет содержимое выбранной клетки: ○ Если это мина, игрок проигрывает, и игра завершается. ○ Если клетка безопасна, программа отображает количество мин, находящихся вокруг неё (восемь соседних клеток). 3. Игра продолжается до тех пор, пока игрок не откроет все безопасные клетки или не наткнётся на мину. **5. Победа и поражение** ● Игрок выигрывает, если все безопасные клетки открыты, при этом не попав на мину. ● Игра проиграна, если игрок открывает клетку с миной. **6. Дополнительные функции (обязательно)** ● Валидация пользовательского ввода (например, проверка координат на корректность). ● Отображение всех мин на поле, если игрок наткнулся на одну из них. ● Возможность повторной игры после победы или поражения.
4	Четыре в ряд	**1. Цель** Создать консольное приложение на языке C#, моделирующее игру «Четыре в ряд», в которой два игрока поочередно бросают фишки в столбцы поля размером 6x7 с целью собрать линию из четырёх фишек. **2. Игровое поле** ● Поле представляет собой двумерный массив размером 6x7, где каждая клетка либо пуста,

		либо содержит фишку одного из двух игроков. ● Игра начинается с пустого поля, и по мере хода игроков фишки заполняют столбцы от нижней строки к верхней. **3. Игровой процесс** 1. Игроки поочередно вводят номер столбца (от 1 до 7), в который хотят сбросить свою фишку. 2. Фишка падает в выбранный столбец и занимает самое нижнее доступное место. 3. После каждого хода проверяется, не собрал ли кто-то из игроков линию из четырёх фишек по горизонтали, вертикали или диагонали. 4. Если один из игроков собирает линию из четырёх фишек, он выигрывает. **4. Конец игры** ● Игра завершается, если один из игроков собрал линию из четырёх фишек по горизонтали, вертикали или диагонали. ● Игра также заканчивается, если все клетки поля заполнены, и ни один из игроков не собрал линию (ничья). **5. Правила проверки победы** ● После каждого хода проводится проверка: ○ По горизонтали: проверяются все последовательные четыре фишки в одном ряду. ○ По вертикали: проверяются все последовательные четыре фишки в одном столбце. ○ По диагоналям: проверяются все возможные диагональные линии, идущие слева направо и справа налево. **6. Дополнительные функции (обязательно)** ● Валидация вводимых данных (например, проверка, что столбец не переполнен и что введённое значение корректно). ● Отображение состояния игрового поля после каждого хода. ● Поддержка режима новой игры после завершения текущей партии.
5	Крестики-нолики на большом поле	**1. Цель**

		Создать консольное приложение на языке C#, моделирующее игру «Крестики-нолики» на большом поле 10x10, где два игрока поочередно размещают свои символы (X и O), стремясь собрать линию из пяти символов. **2. Игровое поле** ● Поле представляет собой двумерный массив размером 10x10. ● В начале игры поле пустое, и по мере ходов заполняется крестиками (X) и ноликами (O). **3. Игровой процесс** 1. Игроки поочередно вводят координаты клетки, куда хотят поставить свой символ (например, A3 или B5). 2. Символ (X или O) размещается в выбранной клетке. 3. После каждого хода проверяется, не собрал ли кто-то из игроков линию из пяти символов по горизонтали, вертикали или диагонали. 4. Если игрок собрал линию из пяти символов, он выигрывает. **4. Конец игры** ● Игра завершается, когда один из игроков соберёт линию из пяти символов подряд по горизонтали, вертикали или диагонали. ● Если все клетки поля заполнены, но линия не была собрана, игра заканчивается вничью. **5. Правила проверки победы** ● После каждого хода программа проверяет: ○ По горизонтали: все возможные последовательные комбинации из пяти символов в строках. ○ По вертикали: все возможные последовательные комбинации из пяти символов в столбцах. ○ По диагоналям: все возможные диагональные линии слева направо и справа налево с пятью символами. **6. Дополнительные функции (обязательно)** ● Валидация пользовательского ввода (проверка координат на корректность и что клетка не занята). ● Отображение текущего состояния игрового

		поля после каждого хода. ● Возможность начать новую игру после завершения текущей.
6	Змейка	**1. Цель** Создать консольное приложение на языке C#, моделирующее игру «Змейка», в которой игрок управляет змейкой и старается собрать как можно больше еды, избегая столкновений. **2. Игровое поле** ● Поле представляет собой двумерный массив размером 20x20. ● Изначально на поле только змейка (из одного сегмента) и одна случайно сгенерированная еда. **3. Игровой процесс** 1. Игрок управляет направлением змейки с помощью клавиш: вверх, вниз, влево или вправо. 2. Еда появляется в случайной свободной клетке на поле. 3. Когда змейка собирает еду, она увеличивается на один сегмент. 4. Змейка продолжает двигаться, следуя введённым пользователем направлениям, и её длина увеличивается с каждой съеденной едой. **4. Конец игры** ● Игра завершается, если змейка врезается в стену или в своё собственное тело. ● Игра может быть перезапущена после завершения. **5. Правила генерации еды** ● Еда появляется в случайной клетке поля, которая не занята телом змейки. ● После того как змейка съедает еду, на поле генерируется новая еда. **6. Дополнительные функции (обязательно)** ● Валидация вводимых данных (корректная обработка клавиш для управления). ● Отображение текущего состояния поля: змейка, свободные клетки и еда.

		• Подсчёт текущего результата (количество собранной еды).
7	Шахматы	**Цель**: Перемещать фигуры по шахматным правилам. **Игровое поле**: двумерный массив 8x8. **Правила**: • Игроки поочередно ходят фигурами. • Достаточно реализовать пешки, ладьи, кони и короля. **Конец игры**: Игра заканчивается при победе или ничье (мат, пат).
8	Змейка с препятствиями	**1. Цель** Создать консольное приложение на языке C#, моделирующее игру «Змейка с препятствиями», где игрок управляет змейкой, собирая еду и избегая столкновений с препятствиями и своим телом. **2. Игровое поле** • Поле представляет собой двумерный массив размером 20x20. • На поле случайным образом размещаются препятствия, которые остаются неподвижными в течение всей игры. • Змейка стартует в случайной позиции, а еда генерируется в случайной свободной клетке. **3. Игровой процесс** 1. Игрок управляет направлением змейки с помощью клавиш (вверх, вниз, влево, вправо). 2. Еда появляется в случайной свободной клетке, которая не занята препятствием или телом змейки. 3. При сборе еды змейка увеличивается на один сегмент. 4. Змейка продолжает двигаться, следуя вводимым игроком направлениям, при этом игрок должен избегать столкновения с препятствиями или телом змейки. **4. Конец игры** • Игра завершается, если змейка врезается в

		препятствие или в своё собственное тело. ● Возможность перезапустить игру после завершения. **5. Правила генерации препятствий и еды** ● Препятствия генерируются случайным образом при старте игры, и остаются на своих позициях. ● Еда появляется в случайных клетках, не занятых змейкой или препятствиями. ● После съедания еды змейкой, новая еда появляется в другой свободной клетке. **6. Дополнительные функции (обязательно)** ● Валидация пользовательского ввода (обработка управления). ● Отображение игрового поля, включая змейку, препятствия и еду. ● Подсчёт результата (собранная еда).
9	Магический квадрат	**1. Цель** Создать консольное приложение на языке C#, моделирующее игру «Магический квадрат», где игрок должен разместить числа от 1 до 9 на поле 3x3 таким образом, чтобы сумма чисел в каждой строке, столбце и диагонали была одинаковой. **2. Игровое поле** ● Поле представляет собой двумерный массив размером 3x3, в котором игрок будет размещать числа от 1 до 9. ● Изначально все клетки пусты. **3. Игровой процесс** 1. Игрок вводит число от 1 до 9 и координаты для его размещения на игровом поле. 2. Число может быть использовано только один раз. 3. После каждого хода программа проверяет, удовлетворяет ли текущее расположение чисел условию магического квадрата (сумма чисел в каждой строке, каждом столбце и обеих диагоналях должна быть равной). **4. Конец игры** ● Игра завершается, когда игрок успешно соберёт магический квадрат, т.е. когда сумма в

		каждой строке, столбце и диагонали будет одинаковой. ● Также игра завершается, если все числа расставлены, но магический квадрат не собран. **5. Правила проверки магического квадрата** ● Магический квадрат для поля 3x3 — это такая расстановка чисел от 1 до 9, при которой сумма в каждой строке, столбце и диагонали равна 15. ● После каждого введённого числа программа проверяет текущее состояние поля, сверяя суммы строк, столбцов и диагоналей. **6. Дополнительные функции (обязательно)** ● Валидация ввода игрока (числа должны быть в диапазоне от 1 до 9, координаты должны быть в пределах поля, каждое число может быть использовано только один раз). ● Отображение текущего состояния поля после каждого хода. ● Сообщение об успешном завершении игры, когда магический квадрат собран.
10	Поиск сокровищ	**1. Цель** Создать консольное приложение на языке C#, моделирующее игру «Поиск сокровищ», где игрок ищет спрятанные сокровища на игровом поле, избегая ловушек. **2. Игровое поле** ● Поле представляет собой двумерный массив размером 10x10. ● На поле случайным образом размещены сокровища и ловушки. ● Количество сокровищ и ловушек задаётся программой при инициализации игры. **3. Игровой процесс** 1. Игрок вводит координаты (например, A3), чтобы произвести поиск в соответствующей клетке поля. 2. Если в указанной клетке находится сокровище, оно засчитывается как найденное, и клетка помечается как пустая. 3. Если в указанной клетке находится ловушка,

		игра сразу же завершается поражением. 4. Если клетка пустая, игрок может продолжить поиски. **4. Конец игры** ● Игра завершается, когда игрок находит все спрятанные сокровища, и это считается победой. ● Игра завершается, если игрок попадает на ловушку — это считается поражением. **5. Правила размещения сокровищ и ловушек** ● Сокровища и ловушки случайным образом размещаются на игровом поле при старте игры. ● Количество сокровищ и ловушек фиксировано, чтобы обеспечить баланс игры (например, 10 сокровищ и 5 ловушек). **6. Дополнительные функции (обязательно)** ● Валидация пользовательского ввода: проверка правильности координат и того, что игрок не выбирает одну и ту же клетку дважды. ● Отображение игрового поля с уже исследованными клетками. ● Подсчёт количества найденных сокровищ и вывод результата после каждого хода. ● Сообщение о победе, если все сокровища найдены, или о поражении, если игрок попал в ловушку.

Практические задачи простые (1 обязательно вместо теории):

1. **Сумма элементов в двумерном массиве**. Напишите метод, который принимает двумерный массив целых чисел и возвращает сумму всех его элементов.
2. **Максимальный элемент в трёхмерном массиве**. Создайте метод, который принимает трёхмерный массив целых чисел и находит максимальное значение.
3. **Транспонирование матрицы**. Напишите метод, который принимает двумерную матрицу и возвращает её транспонированную (меняет местами строки и столбцы).
4. **Сумма каждой строки в двумерном массиве**. Реализуйте метод, который принимает двумерный массив целых чисел и возвращает новый массив, содержащий сумму элементов каждой строки.
5. **Сумма каждого столбца в двумерном массиве**. Напишите метод, который вычисляет сумму каждого столбца в двумерном массиве.
6. **Поиск значения в двумерном массиве**. Реализуйте метод, который принимает двумерный массив и число, и проверяет, содержится ли это число в массиве.

7. **Преобразование двумерного массива в одномерный**. Напишите метод, который преобразует двумерный массив в одномерный, копируя его элементы построчно.
8. **Умножение двух матриц**. Создайте метод, который умножает две матрицы (двумерные массивы) и возвращает результат в виде новой матрицы.
9. **Проверка симметрии двумерного массива**. Напишите метод, который проверяет, является ли двумерный массив (матрица) симметричным (т.е. совпадает ли он со своей транспонированной матрицей).
10. **Диагональные элементы квадратной матрицы**. Реализуйте метод, который принимает квадратную матрицу (двумерный массив) и возвращает массив, содержащий элементы на её главной диагонали.
11. **Сортировка строк в двумерном массиве**. Напишите метод, который сортирует каждую строку двумерного массива целых чисел по возрастанию.
12. **Сумма диагональных элементов в двумерном массиве**. Создайте метод, который вычисляет сумму элементов на главной диагонали квадратной матрицы (двумерного массива).
13. **Проверка на единичную матрицу**. Напишите метод, который определяет, является ли заданная квадратная матрица (двумерный массив) единичной (1 на диагонали и 0 в остальных местах).
14. **Реверс строк двумерного массива**. Реализуйте метод, который разворачивает порядок элементов в каждой строке двумерного массива.
15. **Минимум в каждом столбце двумерного массива**. Напишите метод, который находит и возвращает минимальное значение из каждого столбца двумерного массива.

Практические задачи посложнее (1 обязательно):

1. **Сумма элементов по главной и побочной диагоналям в двумерном массиве.** Напишите метод, который принимает квадратный двумерный массив и возвращает сумму элементов по главной и побочной диагоналям. При пересечении диагоналей не учитывайте повторяющийся элемент.
2. **Максимальный элемент в трёхмерном массиве с координатами**. Создайте метод, который находит максимальный элемент в трёхмерном массиве и возвращает как его значение, так и координаты (индексы) этого элемента.
3. **Нормализация матрицы.** Реализуйте метод, который нормализует двумерную матрицу, преобразуя каждый элемент в значение от 0 до 1 на основе минимального и максимального значений массива.
4. **Сумма элементов каждой строки и столбца с использованием многомерного массива.** Напишите метод, который принимает двумерный массив и возвращает два одномерных массива: один содержит суммы строк, а другой — суммы столбцов.
5. **Произведение двух матриц с проверкой совместимости.** Реализуйте метод для умножения двух матриц, проверяя перед этим, совместимы ли они по размеру для выполнения умножения. Верните результат в виде новой матрицы, или сообщите об ошибке, если умножение невозможно.
6. **Нахождение локальных максимумов в двумерном массиве.** Напишите метод, который находит все элементы в двумерном массиве, являющиеся

локальными максимумами (элемент больше своих соседей по горизонтали и вертикали).

7. **Обратная матрица (инверсия) для квадратной матрицы.** Создайте метод, который принимает квадратную матрицу и вычисляет её обратную матрицу, если это возможно. В случае невозможности инверсии верните сообщение об ошибке.
8. **Поиск всех седловых точек в двумерном массиве.** Напишите метод, который находит все седловые точки в двумерной матрице. Седловая точка — это элемент, который является минимальным в своей строке, но максимальным в своём столбце.

Дополнительные материалы для подготовки:

- **https://metanit.com/sharp/tutorial/2.4.php**
- **https://metanit.com/sharp/tutorial/2.7.php**