Министерство образования и науки РФ

Федеральное государственное автономное образовательное учреждение высшего образования

«Омский государственный технический университет»

|  |  |
| --- | --- |
| Факультет (институт) | *Информационных технологий и компьютерных систем* |
|  |  |
| Кафедра | *Прикладная математика и фундаментальная информатика* |
|  |  |

**Расчетно-графическая работа**

|  |  |
| --- | --- |
| по дисциплине | ***Алгоритмизация и программирование*** |
|  |  |
| на тему | Разработка программы |

Пояснительная записка

|  |  |
| --- | --- |
| **Шифр проекта** | 020-РГР-02.03.03-№ 2-ПЗ |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Студента** | | Биндюк Глеб Игоревич | | | | | |
|  |  |  |  | | фамилия, имя, отчество полностью | | | | | |
|  |  |  | Курс | *1* |  | Группа | | МО-231 | | |
|  |  |  |  |  |  | |  |  | |  |
|  | | | **Направление (специальность)** | | | | | ***02.03.03*** | | |
|  | | | *Математическое обеспечение и администрирование информационных систем* | | | | | | | |
|  |  |  | код, наименование | | | | | | | |
|  |  |  | Руководитель | | ***ст. преподаватель*** | | | | | |
|  |  |  | ученая степень, звание | | | | | |
|  |  |  | ***Федотова И.В.*** | | | | | | | |
|  |  |  | фамилия, инициалы | | | | | | | |
|  |  |  | Выполнил | |  | | | | | |
|  |  |  | дата, подпись студента | | | | | |
|  |  |  | **Работа защищена с количеством баллов** | | | | | | | |
|  |  |  |  | | | | | |  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | дата, подпись руководителя |  |  |  |

Омск 2023

Оглавление

[Введение 3](#_Toc155389398)

[Теория 4](#_Toc155389399)

[Разработка кода 5](#_Toc155389400)

[Результаты 18](#_Toc155389401)

[Заключение 22](#_Toc155389402)

[Список используемой литературы 23](#_Toc155389403)

# Введение

На сегодняшний день информационные технологии используются во многих отраслях экономики. Достаточно часто перед специалистами в сфере IT стоит задача написать программу, которая будет автоматизировать тот или иной процесс. Эта задача, с одной стороны, связана с хорошим пониманием определенного языка программирования, на котором специалист собирается писать свой код, и, с другой стороны, связана с глубинным пониманием тех действий, которые должен выполнять код. С целью получения опыта в решении данного рода проблем необходимо начать разработку программ для небольших и понятных задач. Для одной из таких задач будет разработан код в этой расчетно­­­­­­-графической работе.

# Теория

По условию задания имеем: необходимо разработать код игры “Морской бой”, где играющему необходимо потопить флот врага, верно угадав клетки на игровом поле 10х10. При этом расстановка флота на поле должна быть реализована с помощью датчика случайных чисел. Если клетка верно угадана игроком, то ставится крестик, если нет – точка.

Исходя из данного условия задачи, необходимо рассмотреть некоторые особенности игры, которые повлияют на создание кода:

* Корабли нельзя поставить вплотную друг к другу, поэтому в программе необходимо следить за тем, чтобы случайная расстановка кораблей была возможна;
* Необходимо отслеживать каждый ход, поэтому вывод поля должен производиться после каждого хода;
* Ввод клетки должен производиться игроком, поэтому в коде должен быть реализован ввод позиции игроком через консоль;
* Расстановка флота должна быть случайной, поэтому для создания кода придется воспользоваться классом, который отвечает за создание случайных чисел.

# Разработка кода

При создании кода в данной расчетно-графической работе я буду пользоваться языком программирования C#.

Для лучшего взаимодействия с игрой были созданы два массива 10х10: один для расстановки кораблей, второй для вывода. Массив для вывода изначально заполнен “~”, так как ни одна из клеток не открыта. Массив для взаимодействия заполнен нулями (нуль по умолчанию стоит в каждой ячейке).

|  |
| --- |
|  |
| Рисунок №1 — создание двух массивов |

Также был создан экземпляр класса Random, который отвечает за создание случайных чисел. Теперь при использовании метода Next() мы легко можем получить случайное число в заданном интервале.

|  |
| --- |
|  |
| Рисунок №2 — создание экземпляра класса Random |

Далее я интегрировал алгоритм расстановки кораблей. Алгоритм состоит из: 1) получения случайных координат корабля; 2) проверки занятости данных координат; 3) поставка корабля на поле; 4) создание границы вокруг каждого корабля, чтобы не допустить невозможной расстановки (см. раздел **Теория**). Существенным недостатком алгоритма является то, что для каждого размера корабля он индивидуален. По этой причине данный алгоритм повторяется 4 раза в коде.

Алгоритм для малых кораблей самый простой – получение случайной координаты, её проверка и постановка корабля с созданием границы вокруг корабля. При постановке корабля его координата меняется с нуля на единицу.

|  |
| --- |
|  |
| Рисунок №3 — алгоритм расстановки малых кораблей (1х1) |

Алгоритм для средних кораблей имеет некоторые отличия от алгоритма для малых. Во-первых, в алгоритме для средних кораблей сначала проверяется, как будет поставлен корабль – вертикально или горизонтально (также с помощью случайного числа: если 0, то корабль располагается по горизонтали; если 1, то по вертикали). Во-вторых, в алгоритме есть частные случаи (горизонтальные частные случаи отличаются от вертикальных): если случайная координата находится на границе поля, то вариант расстановки корабля единственный. Например, если первая координата горизонтального корабля 1 А, то второй может быть только 2 А (0 А координаты нет). В-третьих, если случай постановки обычный, то с помощью случайного числа выбирается в какую сторону (по вертикали – вверх или вниз; по горизонтали – вправо или влево) будет поставлена вторая координата относительно первой клетки (первой клетки корабля). Этап создания границы корабля реализован с помощью гибкого метода Around, который одинаково верно работает для всех размеров корабля. Метод будет рассмотрен позже.

|  |  |  |
| --- | --- | --- |
|  |  | |
| Рисунки №4–5 — алгоритм расстановки средних кораблей по горизонтали (1х2) | | |
|  | |  |
| Рисунки №6–7 — алгоритм расстановки средних кораблей по вертикали (2х1) | | |

Алгоритм для больших кораблей имеет некоторые отличия от алгоритма для средних кораблей. Во-первых, добавляется третья пара координат, которая высчитывается относительно координаты начала корабля и конца. Во-вторых, в условной ветке “других случаев” добавляется еще одно условие: координата конца не должна быть на пределами поля (в случае возникновения исключительной ситуации координаты меняются на случайные).

|  |
| --- |
|  |
| Рисунок №8 — алгоритм расстановки больших кораблей по горизонтали (1х3) (частные случаи) |
|  |
| Рисунок №9 — алгоритм расстановки больших кораблей по горизонтали (1х3) (обычный случай) |
|  |
| Рисунок №10 — алгоритм расстановки больших кораблей по вертикали (3х1) (частные случаи) |
|  |
| Рисунок №11 — алгоритм расстановки больших кораблей по вертикали (3х1) (обычный случай) |

Алгоритм для огромного корабля не имеет принципиальных отличий от алгоритма для больших кораблей (кроме увеличения числа условий и координат).

|  |
| --- |
|  |
| Рисунок №12 — алгоритм расстановки огромного корабля по горизонтали (1х4) (частные случаи) |
|  |
| Рисунок №13 — алгоритм расстановки огромного корабля по горизонтали (1х4) (обычный случай) |
|  |
| Рисунок №14 — алгоритм расстановки огромного корабля по вертикали (4х1) (частные случаи) |
|  |
| Рисунок №15 — алгоритм расстановки огромного корабля по вертикали (4х1) (обычный случай) |

Возвращаясь к реализации алгоритма в целом, стоит отдельно отметить метод Around, “закрашивающий” область корабля (включая его границы) цифрой “2”. Цифра 2 была выбрана закрашивающей специально: в случае проверки свободности клетки клетка считалась бы занятой (чтобы не допускать невозможных комбинаций кораблей), а в случае выбора клетки в процессе игры она не считалась бы выигрышной (т.к. условие попадания – соответствие клетки цифре 1).

|  |
| --- |
|  |
| Рисунок №16 — метод, “закрашивающий” область корабля (для дальнейших взаимодействий с областью) |

Также в коде добавлен вспомогательный вывод поля игры, помогающий удостовериться в правильности работы алгоритма. (0 – пустая клетка, 1 – клетка, на которой находится часть корабля, 2 – окрестность корабля).

|  |
| --- |
|  |
| Рисунок №17 — вспомогательный вывод поля (для проверки работы алгоритма) |

В заключение, рассмотрим основной цикл программы, где и будет происходить взаимодействие пользователя с игрой. Добавлена переменная points, которая отслеживает, сколько частей кораблей было уничтожено игроком (всего их 20). Также добавлены небольшие выводимые строки, которые давали бы возможность пользователю понять, что делать и что произошло (попал он по кораблям или нет). Стоит отметить, что на ввод подается строка, разделяющаяся на число и на символ, который потом конвертируется в число с помощью вспомогательного массива Letters. Для удобства работы с массивами числа уменьшаются на 1 при конвертации (нумерация массивов начинается с нуля, а не единицы). В конце программы добавлен цикл, создающий игровое поле после остановки цикла (когда уничтожены все корабли).

|  |
| --- |
|  |
| Рисунок №18 — основной цикл игры (с выводом поля и вводом) и конечный вывод |

# Результаты

При запуске кода игры всплывают 2 поля игры (первое поле можно убрать при использовании): первое показывает расположение кораблей, чтобы убедиться в верности алгоритмов, а второе – основное поле, которое будет меняться в течение игры. Также будет выведена строка с требованием написать координату, на которую вы хотите напасть.

|  |
| --- |
|  |
| Рисунок №19 — терминал после запуска кода (верхнее поле - опциональное, а второе – игровое) |

Если ввести какую либо координату через пробел (например, “1 А”), то выведется надпись, показывающая попали ли вы или нет. А после надписи игровое после снова появится в терминале с “закрашенной” позицией “1 А” (опциональное поле появляется только при запуске программы и никак потом не появляется в терминале снова).

|  |
| --- |
|  |
| Рисунок №20 — терминал после совершения хода (в случае попадания) |
|  |
| Рисунок №21 — терминал после совершения хода (в случае промаха) |

Когда весь флот врага повержен, игра выводит игровое поле в последний раз и ждет нажатия любой кнопки для отключения.

|  |
| --- |
|  |
| Рисунок №22 — терминал после потопления всех судов в игре пользователем |

# Заключение

В настоящее время человечество все больше нуждается в повсеместной автоматизации. Мы все больше внедряем программы в промышленность, экономику с целью получить достоверные прогнозы, оптимальные варианты решения той или иной проблемы.

В программах очень часто используются гибкие алгоритмы, дающие возможность подстраивать программу под определенного пользователя без колоссальных трудов. Каждый алгоритм решает свою задачу. И к каждому алгоритму нужен свой подход. Любой алгоритм нужно изучить, проработать и довести до оптимального варианта. И уже на основе него создавать программу, сердцем которой станет некоторая идея или алгоритм, который был проработан специально для данной задачи.

В ходе расчетно-графической работы были решены следующие задачи:

* Был разработан алгоритм автоматической случайной расстановки флота на игровом поле;
* Был создан цикл, соединяющий в себе элементы взаимодействия пользователя с игрой и вывода игрового поля в терминал;

# Список используемой литературы

Интернет источник: metanit.com; https://metanit.com/sharp/tutorial/; 04.01.2024; 29.09.2012;

Интернет источник: learn.microsoft.com; https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/; 03.01.2024; 02.05.1991;