НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА   
к курсовой работе

по дисциплине

Программирование

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Имя

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Имя

ГРУППА: Группа

Нижний Новгород

2020 г.

### Оглавление

[Введение 3](#_Toc450920511)

1. [РАЗРАБОТКА И АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ 4](#_Toc450920512)

1.1 [Описание предметной области 4](#_Toc450920515)

1.2 [Обоснования разработки 4](#_Toc450920519)

[1.2.1 Назначение разработки 4](#_Toc450920520)

[1.2.2 Область применения 4](#_Toc450920520)

[1.2.3 Функциональные требования 4](#_Toc450920520)

[1.2.4 Требования к техническим средставам 5](#_Toc450920520)

[1.2.5 Требования к информационной и программной совместимости 5](#_Toc450920520)

[1.2.6 Требования к графическому интерфейсу 5](#_Toc450920520)

1.3 [Анализ технического задания 6](#_Toc450920519)

1.4 [Обоснования разработки 8](#_Toc450920519)

2. [РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 10](#_Toc450920512)

[2.1 Разработка алгоритма работы программы 10](#_Toc450920520)

[2.2 Разработка классов 12](#_Toc450920520)

[2.2.1 Производные классы 13](#_Toc450920520)

[2.3 Программная реализация 16](#_Toc450920520)

[2.3.1 Правила именования переменных 16](#_Toc450920520)

[2.3.1 Структура программы 16](#_Toc450920520)

3. [РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 17](#_Toc450920512)

[Заключение 2](#_Toc450920523)0

[Список литературы 21](#_Toc450920523)

[Приложение 22](#_Toc450920523)

**ВВЕДЕНИЕ**

В наше время наблюдается широкое использование компьютера. Раньше компьютеры служили исключительно для выполнения различных научно-технических и экономических расчетов, и работали на них пользователи с общей компьютерной подготовкой и программисты. А сейчас компьютер стал неотъемлемой частью нашей жизни. Компьютер используется главным образом для работы, но есть определённый процент потребителей, для которых компьютер стал чем-то вроде дорогой игрушки.

Компьютерные программы пользуются большим спросом в потребительской среде. Они упрощают нашу повседневную работу на компьютере и открывают новые возможности применения этой электронной вычислительной машины.

Данная программа может использоваться как детьми, так и взрослыми. Детям она будет полезна как развивающая игрушка. Взрослым можно использовать данную программу в целях разрядки при долгой работе на компьютере.

Также данную программу, возможно, использовать в дошкольных учреждениях и начальных классах, для развития мышления у детей на определенном этапе развития, для формирования понятий «алгоритм», «последовательность действий».

1. **РАЗРАБОТКА И АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ**

### 1.1 Описание предметной области

Игра «морской бой» представляет собой настольную игру на двух участников, которые по очереди выбирают координаты на неизвестном им поле противника. Поле представляет собой набор клеток размеров 10 на 10, на которые устанавливаются корабли. Если у соперника по этим координатам имеется корабль, то корабль или его часть «топится», а попавший получает право на ещё один ход. После уничтожения корабля клетки вокруг него отмечаются как «промахи». Цель – первым поразить все корабли соперника. Всего кораблей в игре 10: 1 четырёхпалубный, 2 трёхпалубных, 3 двухпалубных и 4 однопалубных. Между кораблями должно быть расстояние как минимум 1 пустая клетка. Данный проект будет представлять собой приложение, реализующее игру «морской бой. Целевую аудиторию составляют люди, которые хотят играть в «морской бой». Игроки будут иметь возможность играть друг с другом. Данный проект будет полезен для тех, кто хочет немного отдохнуть и приятно провести время, сыграв пару боёв в классический «морской бой». Для игры не требуются какие-то специфические знания компьютера, поэтому играть в неё сможет абсолютно кто угодно.

### 1.2 Обоснования разработки

### 1.2.1 Назначение разработки

Программа предназначена для игры двух игроков на одном устройстве.

### 1.2.2 Область применения

Программа используется исключительно для досуга и в личных целях. Не предполагает коммерческого применения.

### 1.2.3. Функциональные требования к системе

* Игра должна быть понятна любому пользователю, не имеющего профильного образования;
* Программа должна уметь уведомлять игрока об очередности ходов;
* Программа должна выполнять действия, которые соответствуют правилам игры.

### 1.2.4 Требования к техническим средствам

Для того чтобы использовать программу, Вам нужен компьютер со следующим аппаратным и программным обеспечением:

* Процессор Pentium (или аналогичный) от 1.6 Ггц;
* Рекомендуемый объем оперативной памяти 512 Мб;
* Видеоадаптер SVGA;
* SVGA монитор;
* 1280х800 – разрешение экрана;
* Операционная система Windows Me/2000/XP;
* Мышь.

### 1.2.5 Требования к техническим средствам

Требования к техническим средствам определяются установленной операционной системой. Программа не является ресурсоемкой и не предъявляет никаких особых требований к техническим средствам.

### 1.2.6 Требования к информационной и программной совместимости

Программное обеспечение должно функционировать на операционных системах семейства MS Windows XP/Vista/7/10. Функционирование программы под операционной системой MS Windows 98/Me также возможно, но тестироваться в рамках данного проекта не будет.

### 1.2.7 Требования к графическому интерфейсу

Система должна обладать интуитивно понятным и Windows-совместимым графическим интерфейсом.

Особенности:

* Наличие игрового поля;
* Наличие контекстного меню.

### 1.3 АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ

Для решения поставленной задачи выбран язык программирования С++.

C++ — компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности.

Для реализации проекта нам потребуется только стандартная библиотека iostream

Частью стандартной библиотеки C++ является библиотека iostream – объектно-ориентированная иерархия классов, где используется и множественное, и виртуальное наследование. В ней реализована поддержка для файлового ввода/вывода данных встроенных типов. Кроме того, разработчики классов могут расширять эту библиотеку для чтения и записи новых типов данных.

Для использования библиотеки iostream в программе необходимо включить заголовочный файл.

#include <iostream>

Листинг 1 – подключение библиотеки

Операции ввода/вывода выполняются с помощью классов istream (потоковый ввод) и ostream (потоковый вывод). Третий класс, iostream, является производным от них и поддерживает двунаправленный ввод/вывод. Для удобства в библиотеке определены три стандартных объекта-потока:

* cin – объект класса istream, соответствующий стандартному вводу. В общем случае он позволяет читать данные с терминала пользователя;
* cout – объект класса ostream, соответствующий стандартному выводу. В общем случае он позволяет выводить данные на терминал пользователя;
* cerr – объект класса ostream, соответствующий стандартному выводу для ошибок. В этот поток мы направляем сообщения об ошибках программы.

Вывод осуществляется, как правило, с помощью перегруженного оператора сдвига влево («), а ввод – с помощью оператора сдвига вправо (»):

#include <iostream>

#include <string>

**int** main()

{

string in\_string;

// вывести литерал на терминал пользователя

cout «"Введите свое имя, пожалуйста: ";

// прочитать ответ пользователя в in\_string

cin » in\_string;

**if** ( in\_string.empty() )

// вывести сообщение об ошибке на терминал пользователя

cerr « "ошибка: введенная строка пуста!\n";

**else** cout « "Привет, " « in\_string « "!\n";

}

Листинг 2 – Пример перегруженных операторов

Назначение операторов легче запомнить, если считать, что каждый "указывает" в сторону перемещения данных. Например,

» x

перемещает данные в x, а

« x

перемещает данные из x.

### 1.4 ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА

Общие принципы:

* Интерфейс должен быть интуитивно понятным. Таким, чтобы пользователю не требовалось объяснять, как им пользоваться.
* Следует с осторожностью предоставлять пользователю возможность, по установке личных настроек.
* Разработка интерфейса исходя из принципа наименьшего возможного количества действий со стороны пользователя.
* Интерфейс должен быть ориентированным на человека, т.е. отвечать нуждам человека и учитывать его слабости. Нужно постоянно думать о том, с какими трудностями может столкнуться пользователь.
* Простое должно оставаться простым. Не усложняйте интерфейсы. Необходимо сделать интерфейс проще и понятнее.

Разработка интерфейса:

На главном экране необходимо разместить вывод поля для игры, у которого должна быть размерность 10х10. Каждый элемент поля занимает два символа при выводе на экран. Сам элемент и правая его правая грань, выводимая элементом 124 таблицы ascii для С++(Рисунок \*).

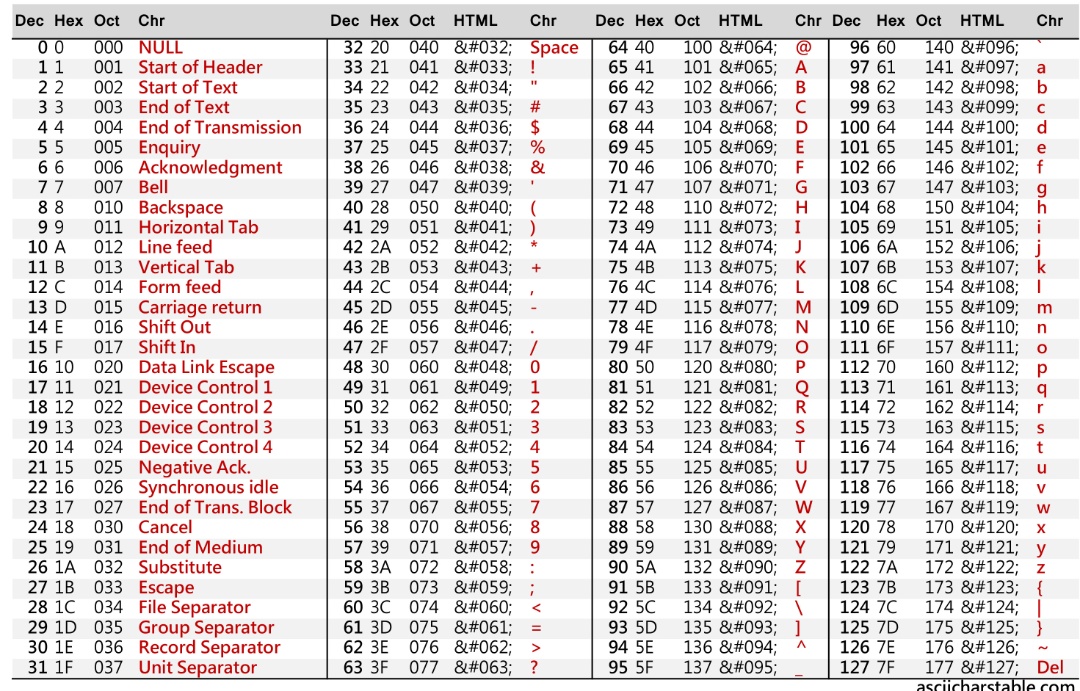


Рисунок 1 - Таблица ascii

Соответственно в ширину интерфейс программы будет занимать 22 символа. В высоту интерфейс составляет 13 символа (игровое поле) плюс строка вывода информации о ходящем игроке, а так же строка вводимой координаты. В сумме 15 строк. На рисунке представлен разработанный интерфейс программы.

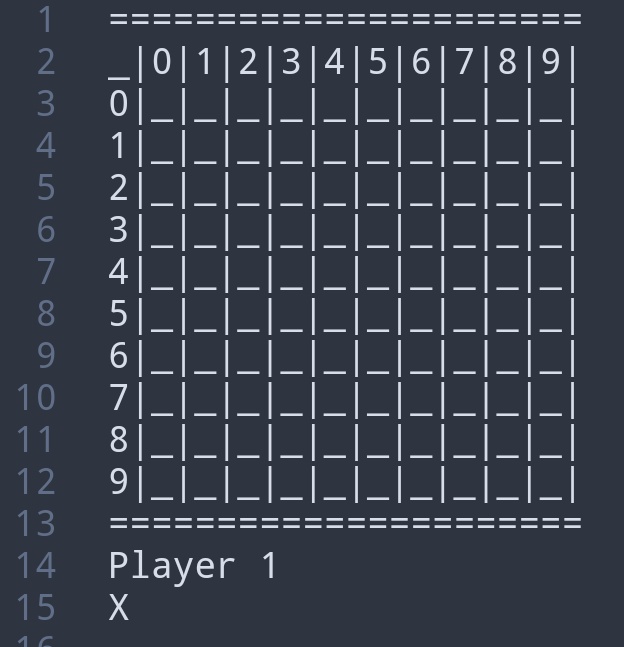


Рисунок 2 – Разработанный интерфейс программы

### 2 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

**2.1 РАЗРАБОТКА АЛГОРИТМА РАБОТЫ ПРОГРАММЫ**

Алгоритм программы состоит в том, что сначала пользователь запускает программу, корабли расставляются автоматически. После делает первый ход, от результатов которого программа определяет кому переходит ход. Подробнее данный алгоритм раскрыт в рисунке 1.

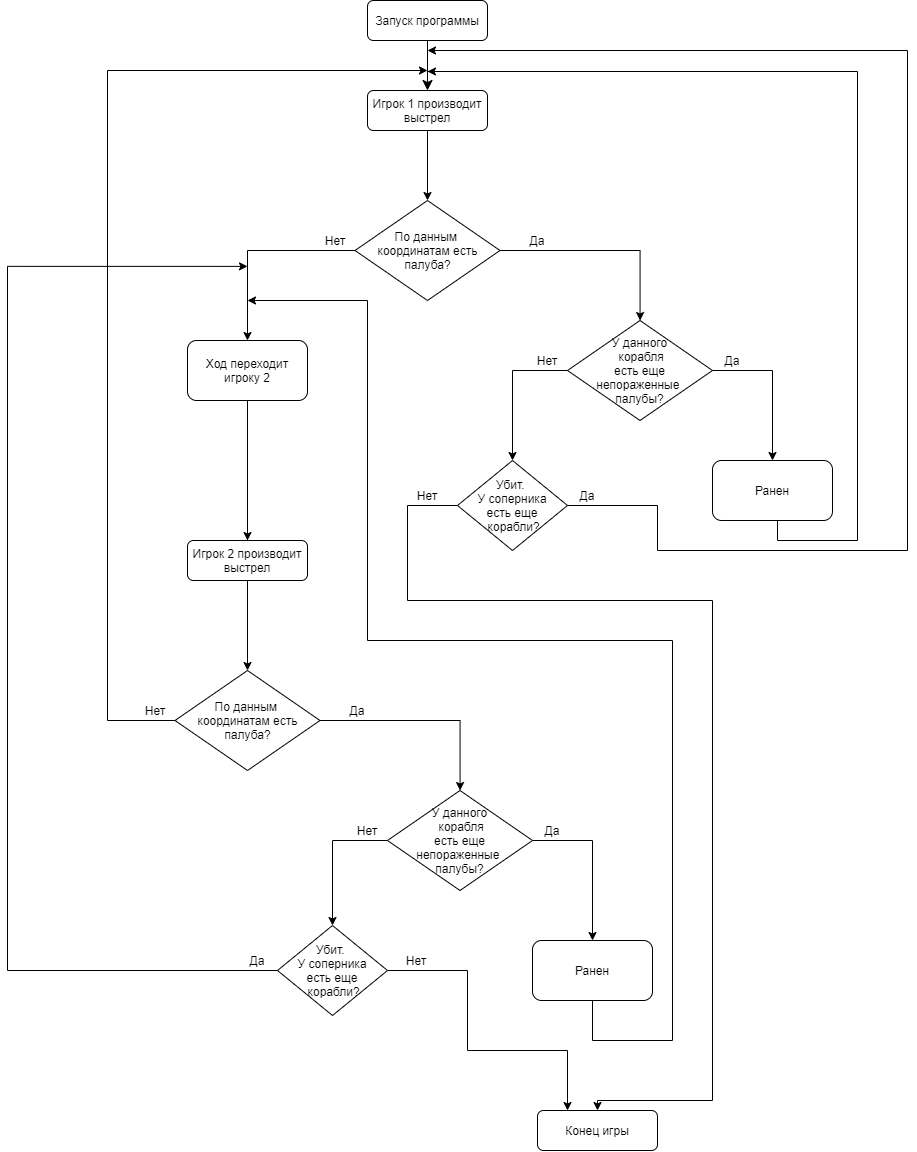


Рисунок 3 – Блок-схема алгоритма.

Для большей наглядности алгоритм оформлен в таблице типичный ход событий.

Таблица 1. Типичный ход событий

|  |  |
| --- | --- |
| Действия исполнителя | Отклик системы |
| 1. Игрок запускает программу   3. Игрок 1 выбирает координаты клетки для выстрела.  5. Игрок 2 выбирает координаты клетки для выстрела. (Повторяется пункт 4) | 2. Система расставляет корабли и предлагает игроку 1 сделать выстрел.  4. Система определяет находится ли палуба по данным координатам.  А) Палуба имеется. Система проверят имеется ли еще не пораженные палубы.  1) Палуб больше нет – убит.  2) Палубы есть – ранен.  Б) Клетка пуста. Ход переходит другому игроку.  6) Кораблей на поле больше нет – определение победителя. |

Таким образом, получаются два варианта использования – ход игрока 1 и ход игрока 2. Целесообразно также предусмотреть Повтор хода в случае попадания в корабль.

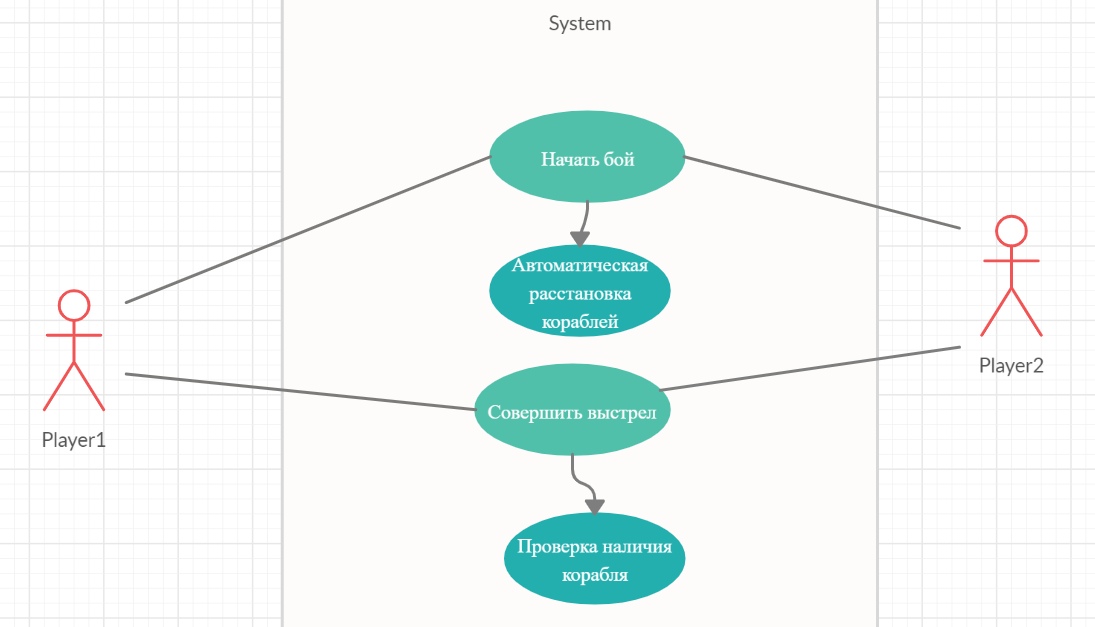


Рисунок 4 - Диаграмма вариантов использования

**2.2 РАЗРАБОТКА КЛАССОВ**

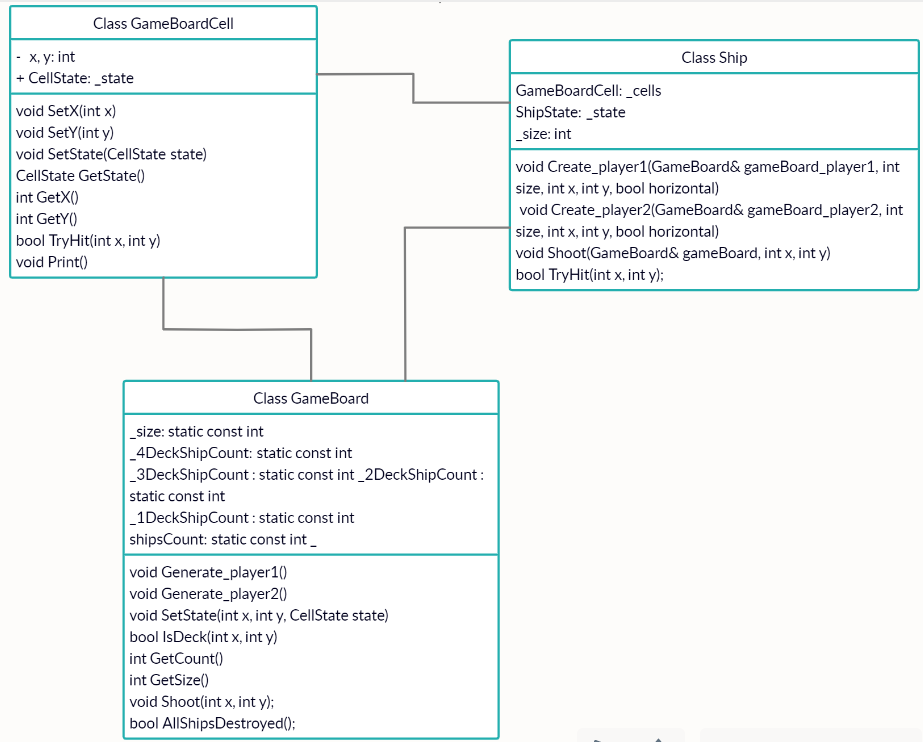


Рисунок 5 – UML диаграмма классов

|  |  |  |  |
| --- | --- | --- | --- |
| В данной программе было разработано 3 класса. | | | |
| Базовый класс программы – класс GameBoardCell, он хранит информацию о клетке игрового поля. В нем определены виртуальные функции – установка координат, состояние клетки, возврат состояния и координат текущей клетки, функция печати и проверка на попадание в клетку. | | | |
|  |  |  |  | |
| К атрибутам класса GameBoardCell относится: координаты клетки и её состояние. | | | |

**class** GameBoardCell { // класс - клетка игрового поля

**private**:

**int** \_x, \_y; // координаты клетки

CellState \_state; // состояние клетки

**public**:

GameBoardCell(**int** x = 0, **int** y = 0, CellState state = Empty) // Констркутор игрового поля. создаем поле

{

\_x = x;

\_y = y;

\_state = state;

}

// функция устанавливает x-кординату клети

**void** SetX(**int** x)

{ \_x = x; }

// функция устанавливает y-кординату клети

**void** SetY(**int** y)

{ \_y = y; }

// функция устанавливает состояние клети

**void** SetState(CellState state)

{ \_state = state; }

// функция возвращает состояние текуйщей клетки

CellState GetState()

{ **return** \_state; }

// функция возвращает x текуйщей клетки

**int** GetX()

{ **return** \_x; }

// функция возвращает y текуйщей клетки

**int** GetY()

{ **return** \_y; }

// функция - проверка попадания в клетку

**bool** TryHit(**int** x, **int** y)

{ **return** x == \_x && y == \_y && \_state != HitDeck; }

// функция печати одной клетки, в зависимости от ее состояния

**void** Print()

{

**if** (\_state == Empty)

cout << '\_';

**else** **if** (\_state == HitDeck)

cout << 'X';

**else** **if** (\_state == Deck)

cout << '=';

**else**

cout << '\*';

}

};

Листинг 3 – Класс GameBoardCell

### 2.2.2 Производные классы

В классе GameBoard определено несколько собственных полей – размер игрового поля, число кораблей с четырьмя палубами, с тремя, двумя и одной соответственно. Так же добавлены функции для работы с этими полями, такие как:

* Функции создания игрового поля
* Функции установки статуса клетки игрового поля
* Функция определения палубы
* Функция печати игрового поля
* Функция возврата числа клеток игровой области
* Функция выстрела в игровое поле
* Функция проверки уничтожения всех кораблей

Кроме того, в классе определены собственные конструкторы и деструкторы.

class GameBoard {

private:

**static** **const** **int** \_size = 10; // размер игорового поля

**static** **const** **int** \_4DeckShipCount = 1; // число 4-х палубных

**static** **const** **int** \_3DeckShipCount = 2; // число 3-х палубных

**static** **const** **int** \_2DeckShipCount = 3; // число 2-х палубных

**static** **const** **int** \_1DeckShipCount = 4; // число 1-х палубных

**static** **const** **int** \_shipsCount = \_4DeckShipCount + \_3DeckShipCount + \_2DeckShipCount + \_1DeckShipCount; // число кораблей

GameBoardCell \_cells[\_size][\_size]; // клетки игрового поля

Ship \_ships[\_shipsCount]; // корабли

// функция создание игрового поля

**void** Generate\_player1();

**void** Generate\_player2();

public:

GameBoard(){

Generate\_player1();

Generate\_player2();

}

~GameBoard() { }

// функция установки статуса клетки игровго поля

**void** SetState(**int** x, **int** y, CellState state)

{ \_cells[y][x].SetState(state); }

// функция возвращает является ли клетка палубой

bool IsDeck(**int** x, **int** y)

{ **return** \_cells[y][x].GetState() == Deck || \_cells[y][x].GetState() == HitDeck; }

// функция печати игрового поля

**void** Print();

// функция возвращает число кдеток игровой области

**int** GetCount()

{ **return** \_size \* \_size; }

// функция возвращает размер игровой области

**int** GetSize()

{ **return** \_size; }

// функция - выстрел в игрове поле

**void** Shoot(**int** x, **int** y);

// функция провекри - уничтожены ли все корабли на поле

bool AllShipsDestroyed();

};

Листинг 4 - Класс GameBoard

В классе Ship собственными полями является клетки корабля, состояние корабля и его размер. Так же определены следующие функции:

Функции создания кораблей

Функция проверки состояния корабля

Функция выстрела по кораблю

class Ship

{

private:

GameBoardCell\* \_cells; // клетки корабля

ShipState \_state; // состояние корабля

**int** \_size; // Размер корабля

public:

Ship() // Констркутор класса корабля

{

\_cells = NULL;

\_size = 0;

}

**void** Create\_player1(GameBoard& gameBoard\_player1, **int** size, **int** x, **int** y, bool horizontal); // Создание корабля для игрока 1

**void** Create\_player2(GameBoard& gameBoard\_player2, **int** size, **int** x, **int** y, bool horizontal); // Создание корабля для игрока 2

~Ship(); // деструктор

// функция вощвращает состояние корабля, выполнив проверку целостности всех палуб

ShipState GetState();

// функция выстрела по корабрю, возвращает состояние корябля, после выстрела

**void** Shoot(GameBoard& gameBoard, **int** x, **int** y);

bool TryHit(**int** x, **int** y);

};

Листинг 5 – Класс Ship

### 2.3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

### 2.3.1 Правила именования переменных

В ходе работы над курсовым проектом были выработаны следующие правила по именованию переменных и написанию кода.

* Все переменные в программе имеют осмысленное название.
* Имена всех пользовательских классов начинаются с заглавной буквы, имена функций классов также начинаются с заглавной буквы.
* Текст программы структурирован, вложенные операторы смещены вправо относительно заголовков.

### 2.3.2 Структура программы

Каждый класс программы реализован в своем модуле, функция main – в отдельном файле.

Программа состоит из следующих модулей.

1. GameBoardCell – описание и определение функций базового класса информации о клетке поля и её состояния
2. GameBoard – описание и определение функций класса игрового поля.
3. Ship – описание и определение функций класса короблей.

**3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Для установки программы скомпилировать программу в любом доступном компиляторе.

После запуска программы будут отображены два поля и игроку 1 предложат ввести координаты для входа:

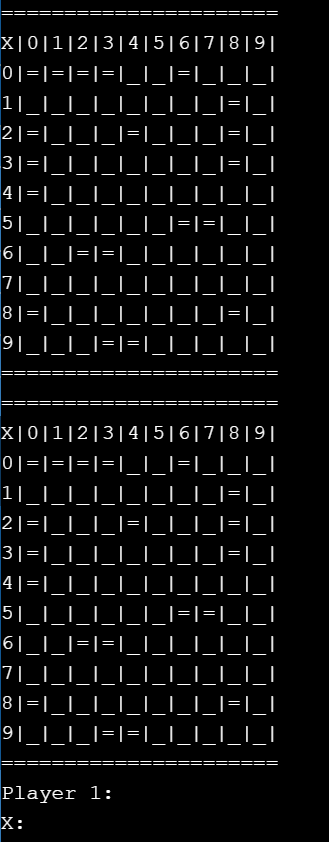


Рисунок 6 – Стартовый вывод

В случае промаха программа печатает поле и передает право хода игроку номер 2. В случае, если же игрок номер 1 попадает по кораблю соперника, ему предоставляется право повторного хода.

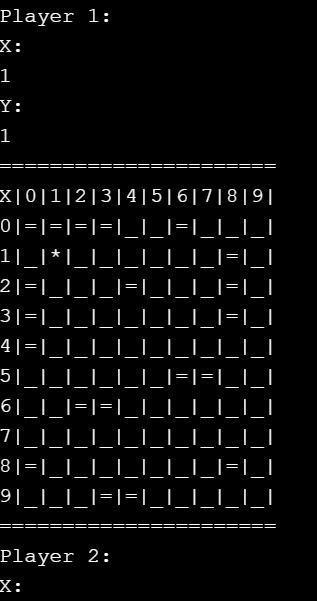


Рисунок 7 – Выстрел по координатам 1:1 игрока номер 1

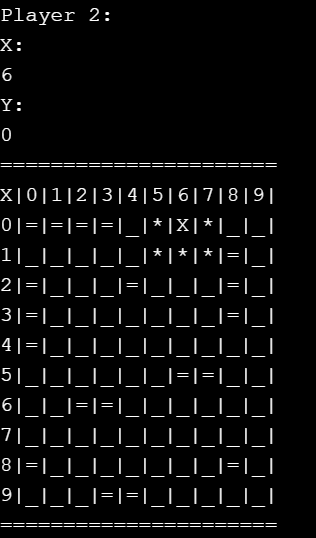


Рисунок 8 – Выстрел по координатам 0:6 игрока номер 2

Как показано на рисунке 8, после успешного выстрела на поле вместо корабля образуется символ убитой палубы и точки вокруг убитого корабля.

**ЗАКЛЮЧЕНИЕ**

В ходе курсового проектирования была разработана программа для игры двух пользователей между собой в «Морской бой».

К достоинствам системы можно отнести простоту работы с программой, быстрый вывод информации, простой интерфейс.

К недостаткам – как ни странно, к недостаткам так же можно отнести простату интерфейса.

Данный проект может дальше развиваться, можно добавлять новые функции, а так же проводить оптимизацию и рефакторинг текущих.

В целом задание, поставленное на курсовое проектирование, выполнено в полном объеме. Проект полностью соответствует техническому заданию.