

## Лабораторная работа №1

«Крестики-нолики». Выявление ключевых параметров игры, реализация игровых правил.

Цель работы: определить необходимый набор объектов для построения игрового поля игры и обеспечения ее функциональности. Реализовать игровые правила.

Задачи:

1. Проанализировать правила игры, выявить ключевые особенности.
2. Определить характеристики игрового поля, основные функции менеджера игры.
3. Выявить основные игровые объекты и их функции.
4. Создать тестовое консольное приложение для тестирования игрового поля.

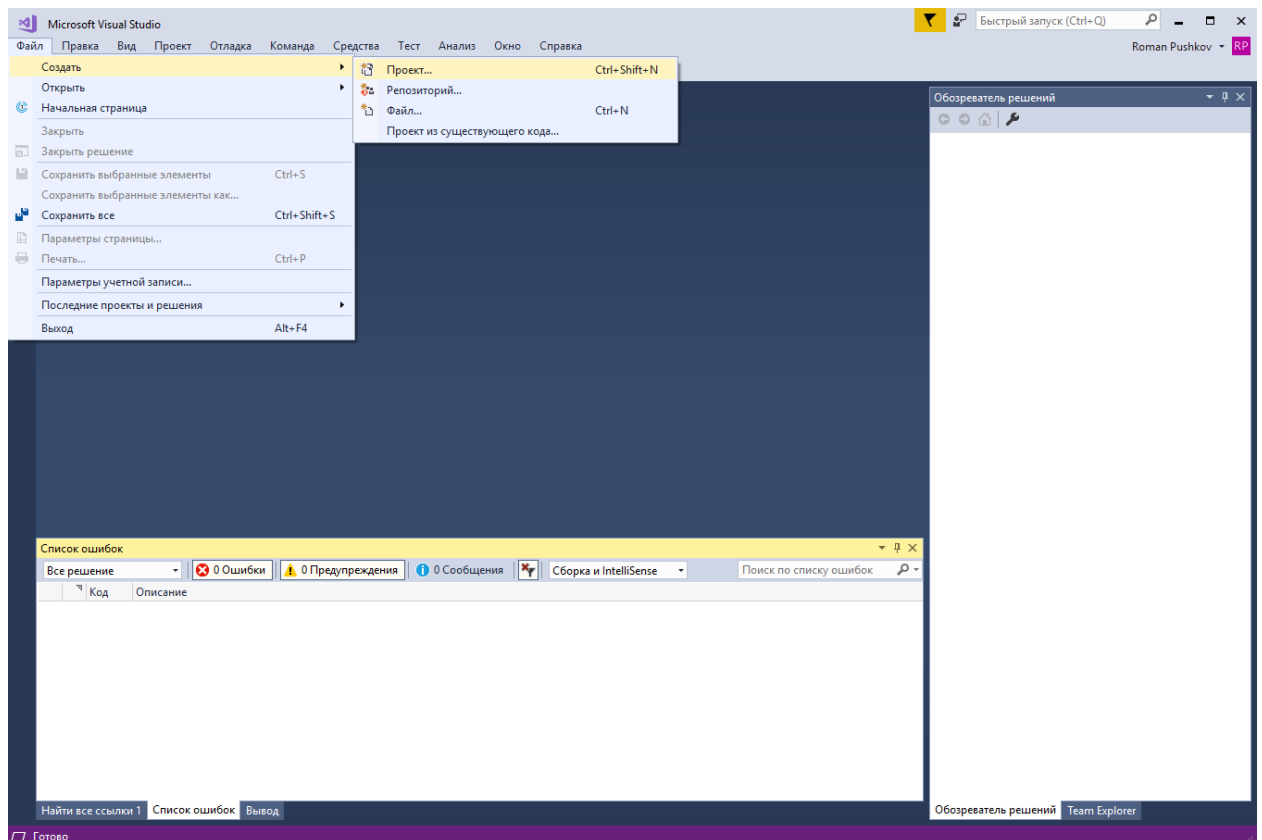
Любое приложение, взаимодействующее с пользователем и решающее какие-либо задачи, должно состоять, как минимум, из двух частей: бизнес-логики и интерфейса пользователя.

Интерфейс пользователя обеспечивает взаимодействие с пользователем – отображение данных приложения и информации, необходимой пользователю, и получение ввода от пользователя.

Основную задачу приложения решает «бизнес-логика» - произведение действий с введенной от пользователя информацией, сбор внешних данных, проведение необходимых расчетов.

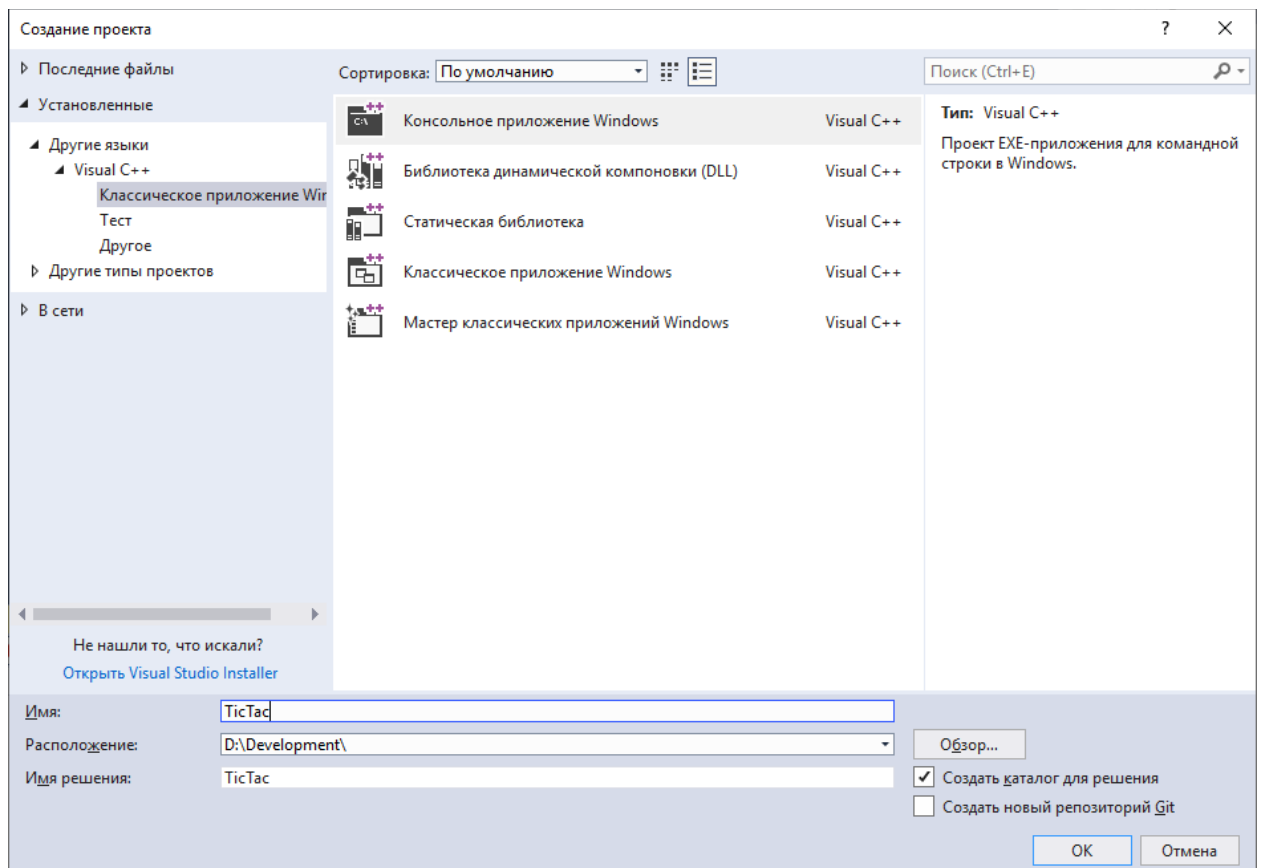
Отделение бизнес-логики от интерфейса позволяет менять и расширять компоненты отдельно друг от друга.

Создаем новый проект.

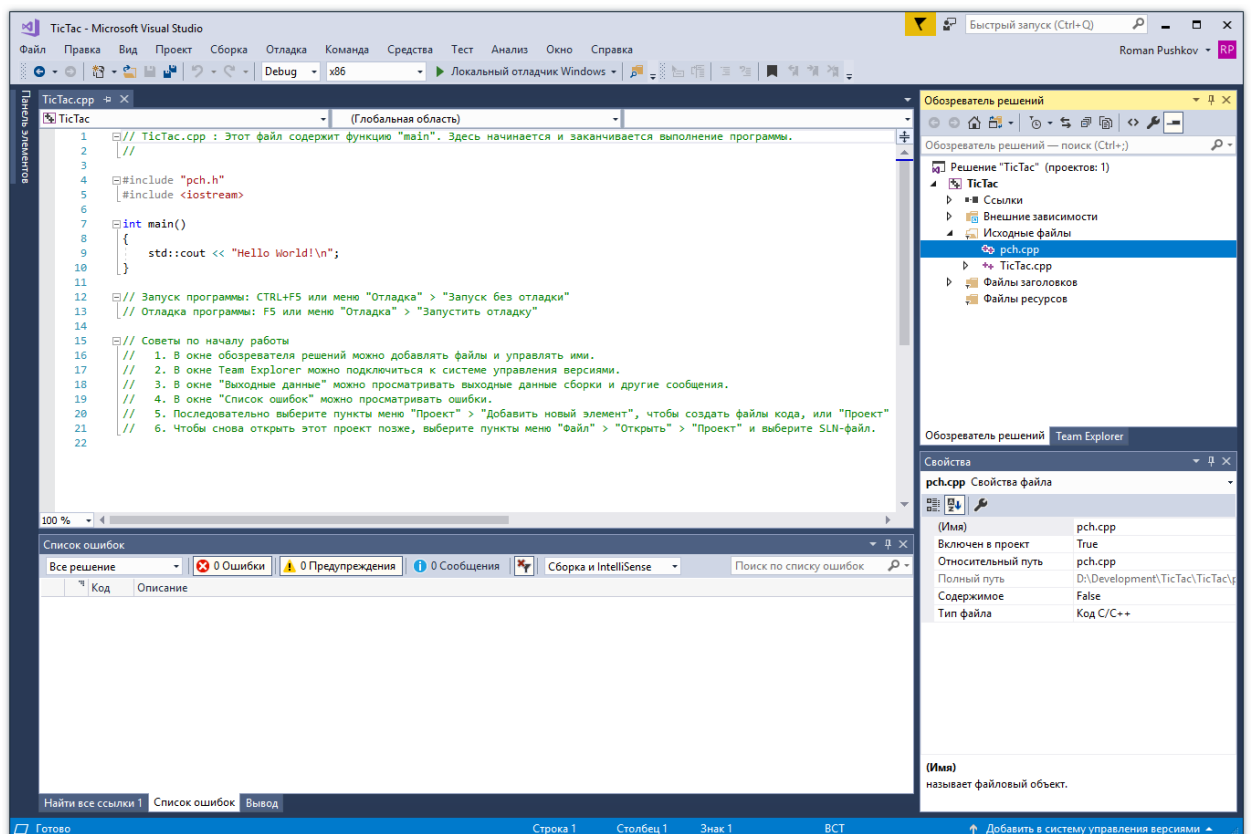


Выбираем консольное приложение Windows, язык C++. Выбор консольного приложения обоснован тем, что разработку бизнес-логики удобнее производить без привязки к интерфейсу, так как интерфейс во многом определяется уже проработанной функциональностью. Разработать

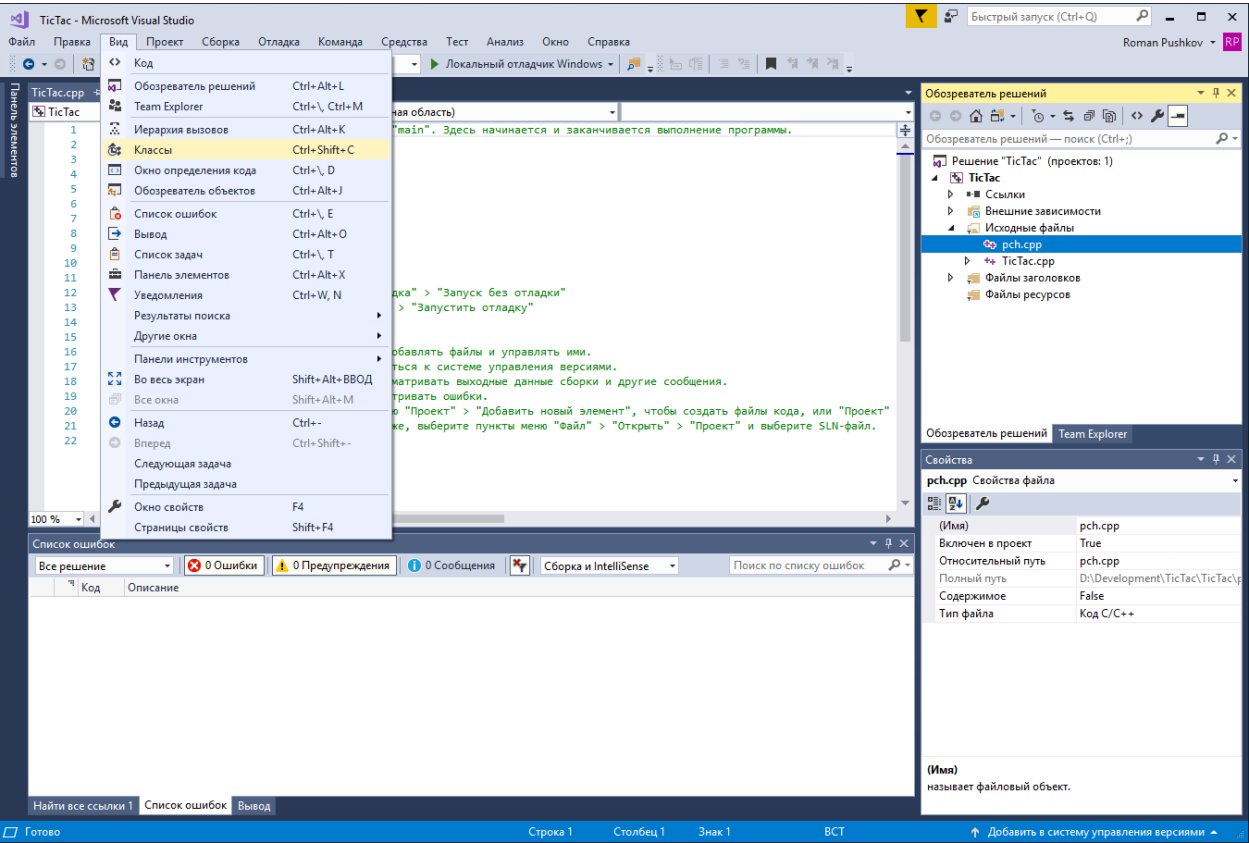
интерфейс является достаточно трудоемкой задачей, однако по итогам разработки бизнес-логики могут выявиться радикальные изменения.



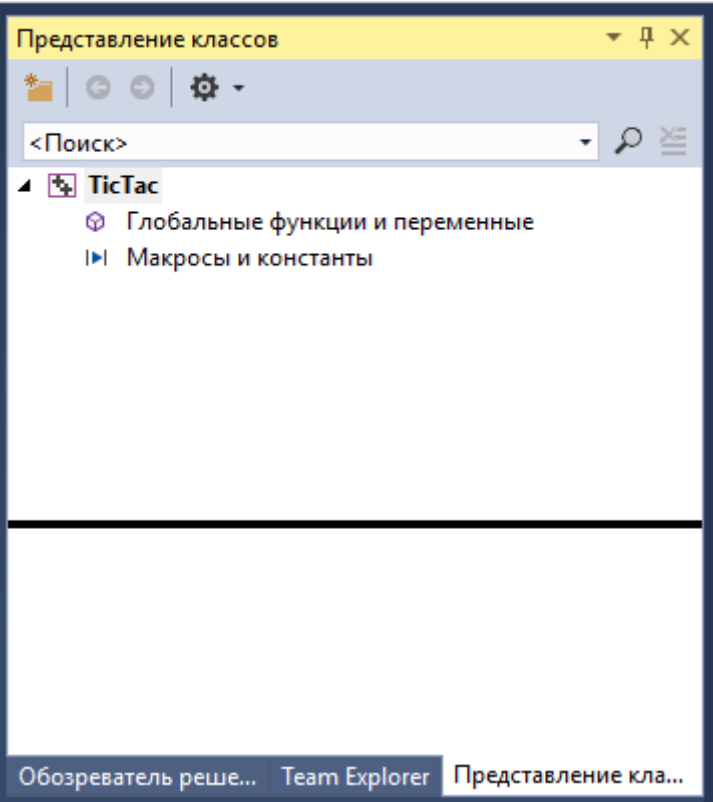
Сформирован пустой проект.



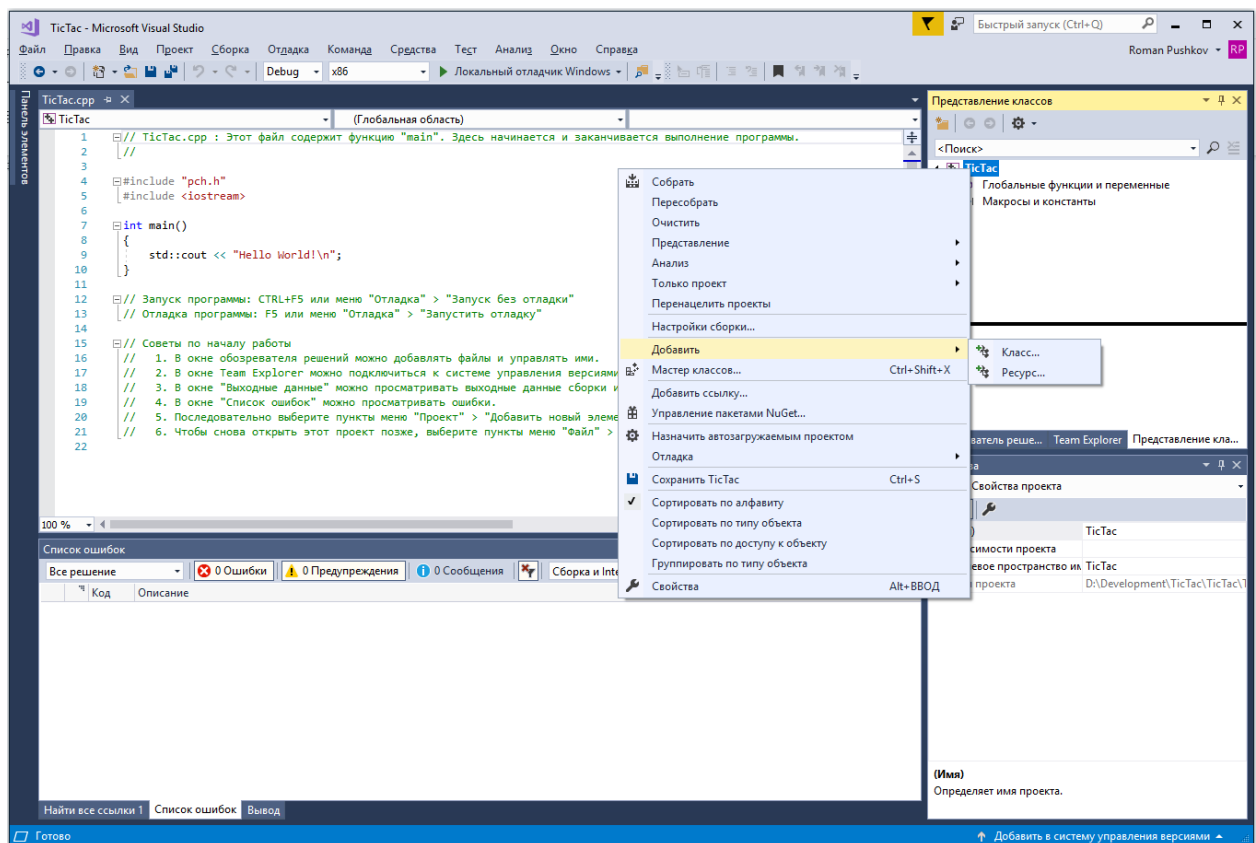
Если отсутствует список классов проекта, то необходимо включить вкладку «Классы».



Вкладка «Представление классов» в структуре проекта.



Добавляем новый класс.



В окне добавляем класс TicTacBoard, который будет представлять поле для игры.

Ставим «виртуальный деструктор», так как применение виртуальных деструкторов рекомендуется в большинстве случаев. Если в нем не будет необходимости, всегда можно убрать его виртуальность вручную.

Добавить класс

Имя класса:

TicTacBoard

Н-файл:

TicTacBoard.h

...

CPP-файл:

TicTacBoard.cpp

...

Базовый класс:

Доступ:

public

Другие параметры:

☒ Виртуальный деструктор

☐ Встроенное

☐ Управляемое

OK

Отмена

Кроме сущности «Игровое поле» напрашивается сущность «Игрок». Заведем класс TicTacPlayer.

Добавить класс

Имя класса:

TicTacPlayer

Н-файл:

TicTacPlayer.h

...

CPP-файл:

TicTacPlayer.cpp

...

Базовый класс:

Доступ:

public

Другие параметры:

☒ Виртуальный деструктор

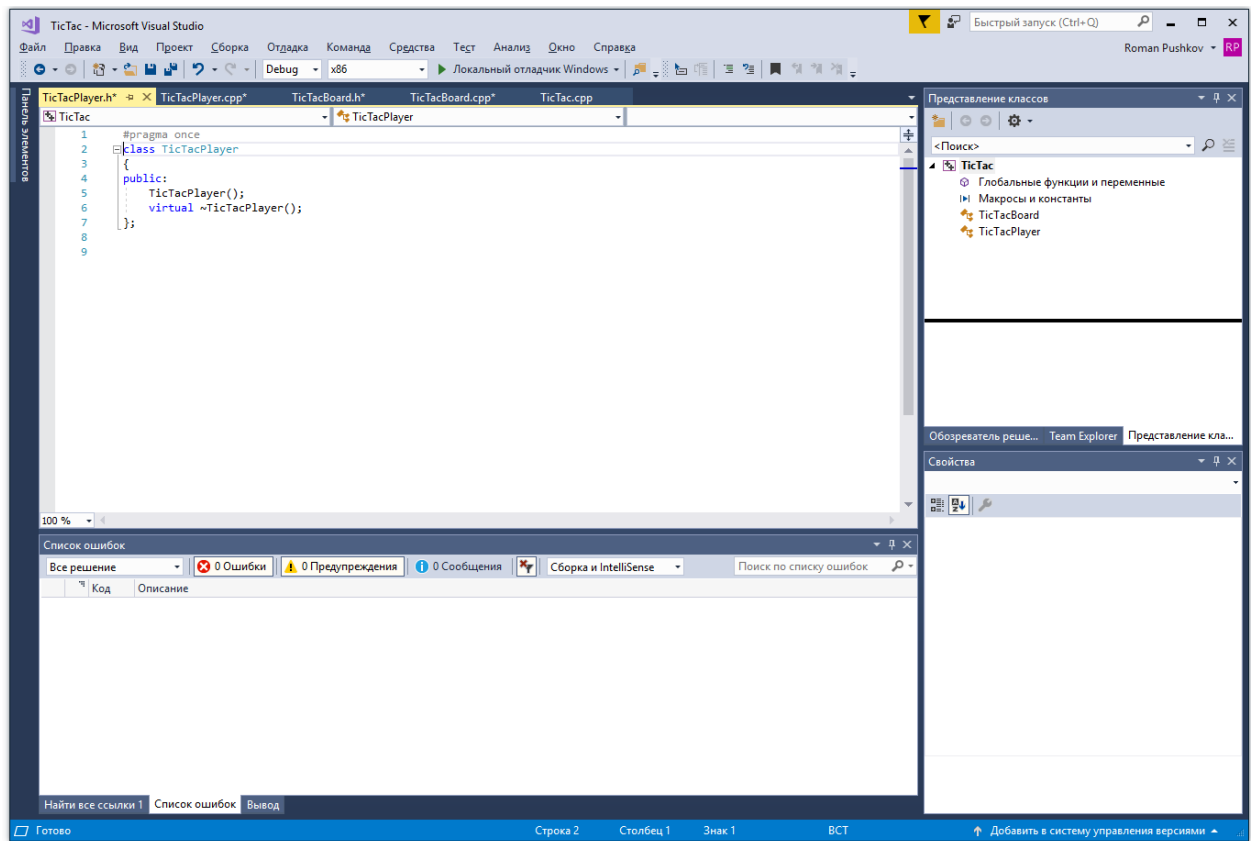
☐ Встроенное

☐ Управляемое

OK

Отмена

Все созданные автоматически классы создаются в виде двух файлов. H-файла с описанием класса и CPP-файла с реализацией. В описание созданных классов уже добавлены конструктор и деструктор, а также их пустые реализации.



Управлять игрой будет специальный объект – менеджер. Добавляем класс этого объекта.

## Добавить класс

Имя класса:

TicTacManager

Н-файл:

TicTacManager.h

...

CPP-файл:

TicTacManager.cpp

...

Базовый класс:

Доступ:

public

Другие параметры:

☒ Виртуальный деструктор

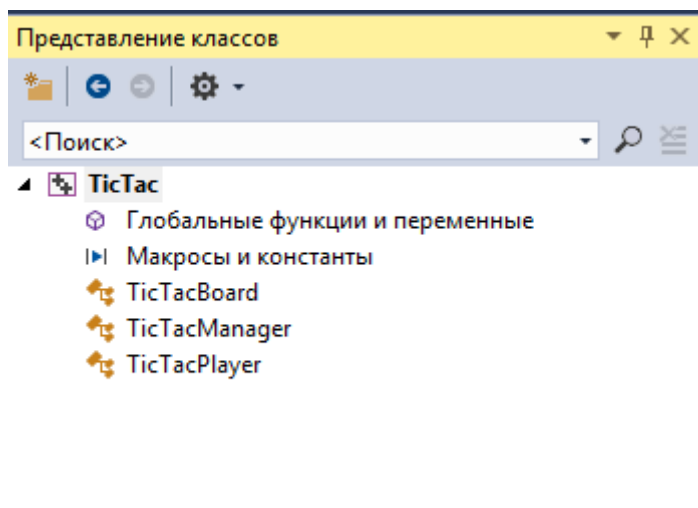
☐ Встроенное

☐ Управляемое

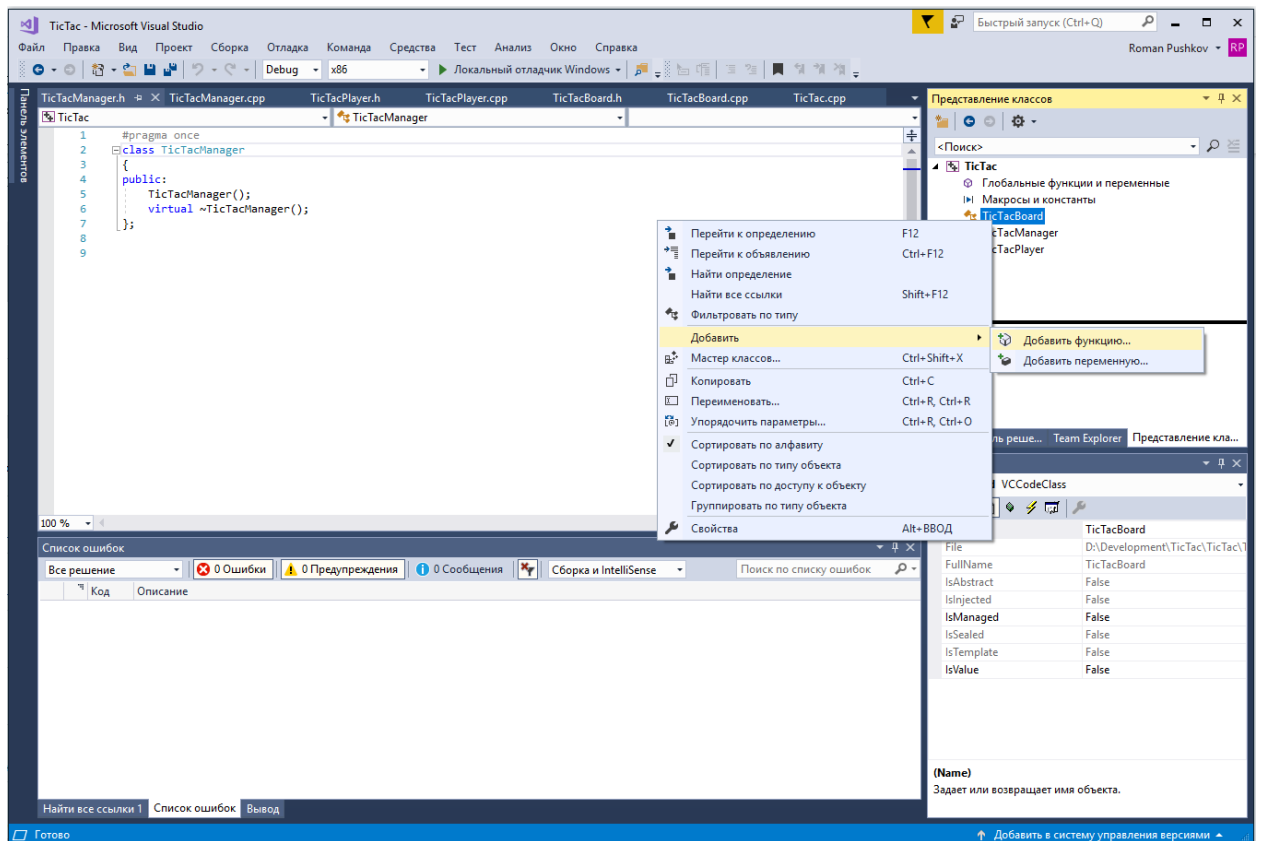
OK

Отмена

Структура классов приложения выглядит следующим образом.



Теперь начнем наполнение. Добавлять функции можно при помощи мастера функций или вручную. Рассмотрим пример добавления при помощи мастера.



Заполняем данные функции.

Добавить функцию

Имя функции:

Show

Тип возвращаемого значения:

void

Доступ:

public

С++-файл:

TicTacBoard.cpp

Комментарий:

Другие параметры:

☐ Встроенное

☐ Статический

☐ Виртуальный

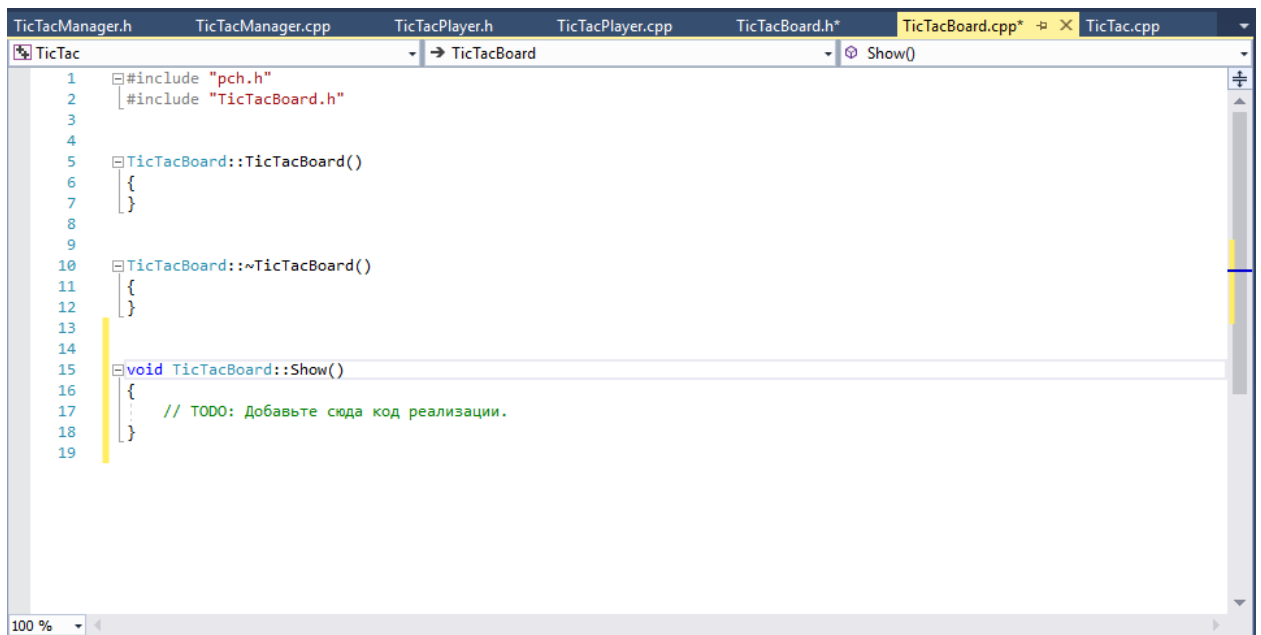
☐ Чистый

OK

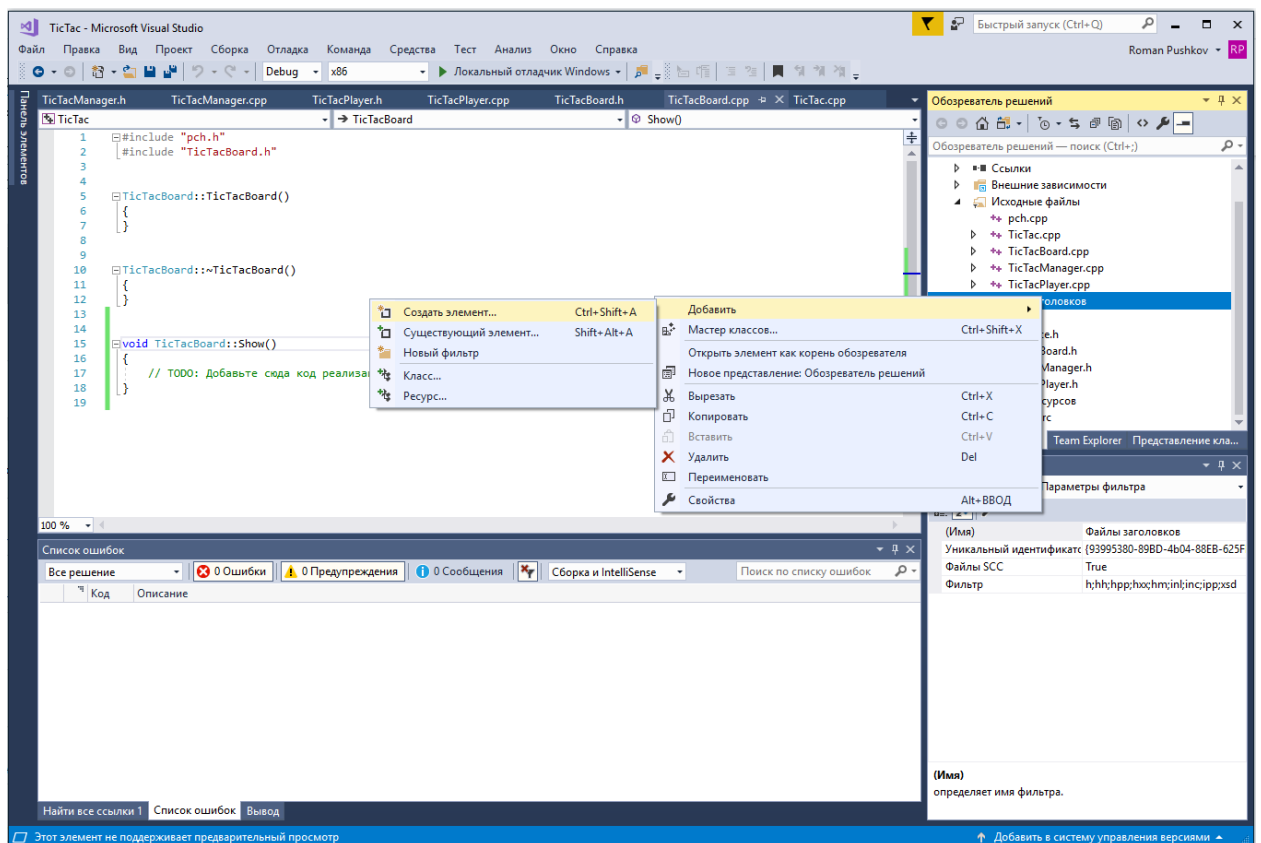
Отмена

Видим добавленную функцию в файле реализации.

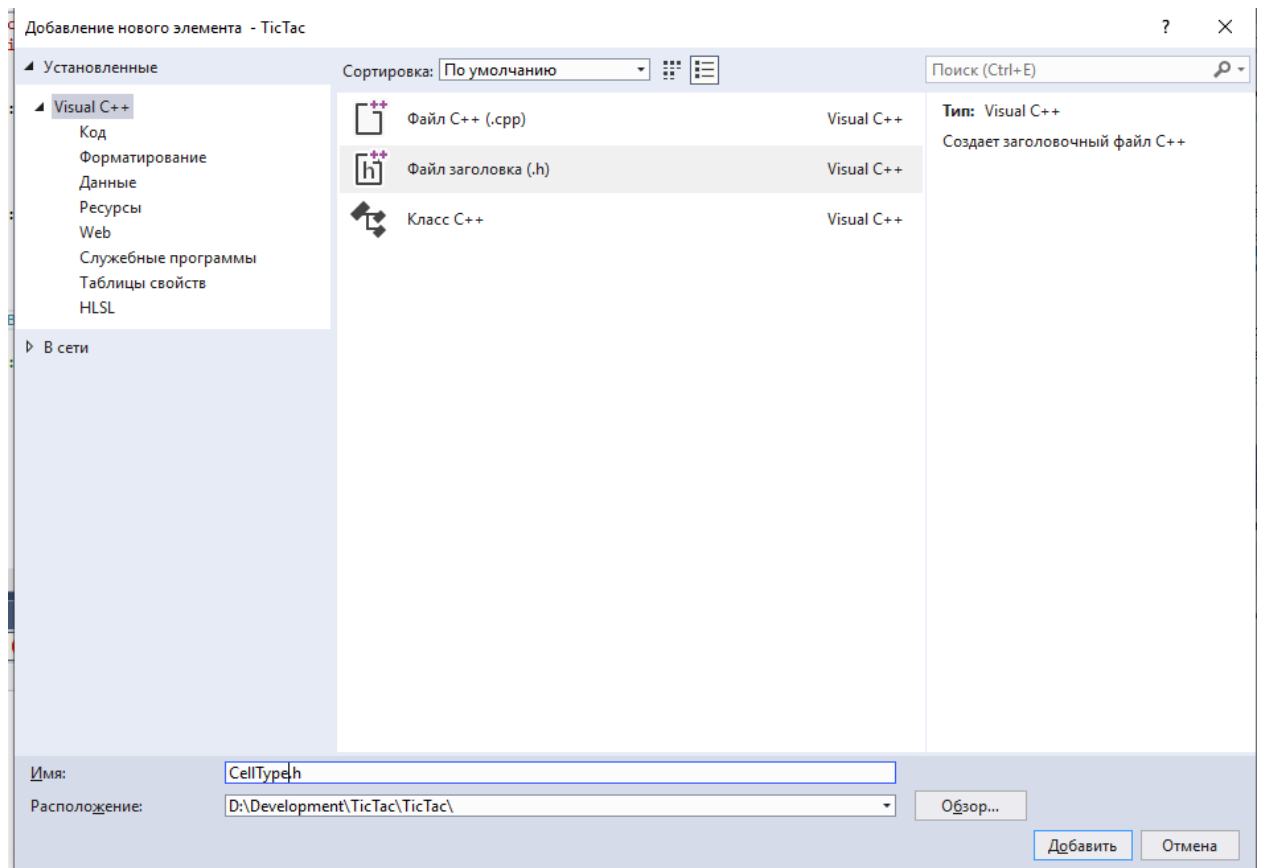




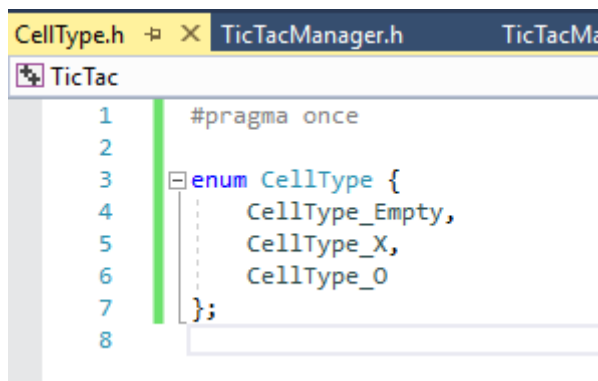
Не все элементы можно добавить с помощью мастера. Добавим в проект перечисление (enum), которое будет принимать значение знака, поставленного в ячейки. Добавляем новый элемент.



Указываем, что это заголовочный файл.



В созданном файле прописываем тип перечисления. Тип знака в ячейке поля может быть «X», «O» или пустая ячейка.



Начинаем наполнять класс поля содержимым.

Он будет содержать размер поля (boardsize) и указатель для динамического двумерного массива, определяющего знаки в позициях.

```
CellType.h    TicTacManager.h    TicTacManager.cpp
TicTac (Глоб
1      #pragma once
2      class TicTacBoard
3      {
4      private:
5          unsigned int boardsize;
6          CellType** cells;
7      public:
8          TicTacBoard();
9          virtual ~TicTacBoard();
10         void Show();
11     };
12
13
```

CellType сейчас не виден. Для видимости подключаем наш заголовочный файл «CellType.h».

```
CellType.h    TicTacManager.h    TicTacManager.cpp
TicTac
1      #pragma once
2      #include "CellType.h"
3
4      class TicTacBoard
5      {
6      private:
7          unsigned int boardsize;
8          CellType** cells;
9      public:
10         TicTacBoard();
11         virtual ~TicTacBoard();
12         void Show();
13     };
14
15
```

Изменяем конструктор класса, чтобы можно было задавать размер поля при создании.

```
TicTac (Глобальная обл
1      #pragma once
2      #include "CellType.h"
3
4      class TicTacBoard
5      {
6      private:
7          unsigned int boardsize;
8          CellType** cells;
9      public:
10         TicTacBoard(unsigned int size);
11         virtual ~TicTacBoard();
12         void Show();
13     };
14
15
```

Реализация конструктора стала ошибочной, так как описание изменилось.

```
TicTacManager.h  TicTacManager.cpp  TicTacBoard.h  TicTacBoard.cpp
TicTacBoard
1  #include "pch.h"
2  #include "TicTacBoard.h"
3
4
5  TicTacBoard::TicTacBoard()
6  {
7  }
8
9
10 TicTacBoard::~TicTacBoard()
11 {
12 }
13
14
15 void TicTacBoard::Show()
16 {
17     // TODO: Добавьте сюда код реализации.
18 }
19
```

Наполняем конструктор. Создаем динамический двумерный массив.

```
Type.h  TicTacManager.h  TicTacManager.cpp  TicTacBoard.h  TicTacBoard.cpp
TicTacBoard
1  #include "pch.h"
2  #include "TicTacBoard.h"
3
4
5  TicTacBoard::TicTacBoard(unsigned int size)
6  {
7      this->boardsize = size;
8      cells = new CellType*[size];
9      for (unsigned int i = 0; i < size; i++)
10         cells[i] = new CellType[size];
11 }
12
```

Добавляем заполнение ячеек поля пустыми знаками.

```

ac
1  #include "pch.h"
2  #include "TicTacBoard.h"
3
4
5  TicTacBoard::TicTacBoard(unsigned int size)
6  {
7      this->boardsize = size;
8      cells = new CellType*[size];
9      for (unsigned int i = 0; i < size; i++)
10         cells[i] = new CellType[size];
11     for (unsigned int i = 0; i < size; i++)
12         for (unsigned int j = 0; j < size; j++)
13             cells[i][j] = CellType_Empty;
14 }
15

```

Не забываем сразу же сделать освобождение памяти в деструкторе.

```

TicTacBoard::~TicTacBoard()
{
    for (unsigned int i = 0; i < boardsize; i++)
        delete[] cells[i];
    delete[] cells;
}

```

Наполняем функцию Show, которая отображает на экране игровое поле.

```

void TicTacBoard::Show()
{
    for (unsigned int i = 0; i < boardsize; i++)
    {
        for (unsigned int j = 0; j < boardsize; j++)
        {
            switch (cells[i][j])
            {
                case CellType_X:
                    cout << "X";
                    break;
                case CellType_O:
                    cout << "O";
                    break;
                case CellType_Empty:
                    cout << "-";
                    break;
            }
            cout << " ";
        }
        cout << endl;
    }
}

```

Для того, чтобы работать с cout необходимо подключить библиотеку iostream. Подключаем ее в файле pch.h, куда необходимо подключать все общие заголовки.

```

5      // 4. В окне "Список ошибок" можно просматривать ошибки.
6      // 5. Последовательно выберите пункты меню "Проект" > "Добавить новый проект"
7      // 6. Чтобы снова открыть этот проект позже, выберите пункты меню "Файл" > "Открыть проект"
8
9      #ifndef PCH_H
10     #define PCH_H
11
12     // TODO: add headers that you want to pre-compile here
13     #include <iostream>
14     using namespace std;
15
16     #include <conio.h>
17     #endif //PCH_H
18

```

Для установки знака в определенную позицию добавляем к классу игрового поля метод SetCell, который принимает координаты поля и символ, который необходимо записать.

```

TicTacBoard (Глобальная область)
1      #pragma once
2      #include "CellType.h"
3
4      class TicTacBoard
5      {
6      private:
7          unsigned int boardsize;
8          CellType** cells;
9      public:
10         TicTacBoard(unsigned int size);
11         virtual ~TicTacBoard();
12         void Show();
13         void SetCell(unsigned int xpos, unsigned int ypos, CellType ct);
14     };
15
16

```

Реализуем.

```

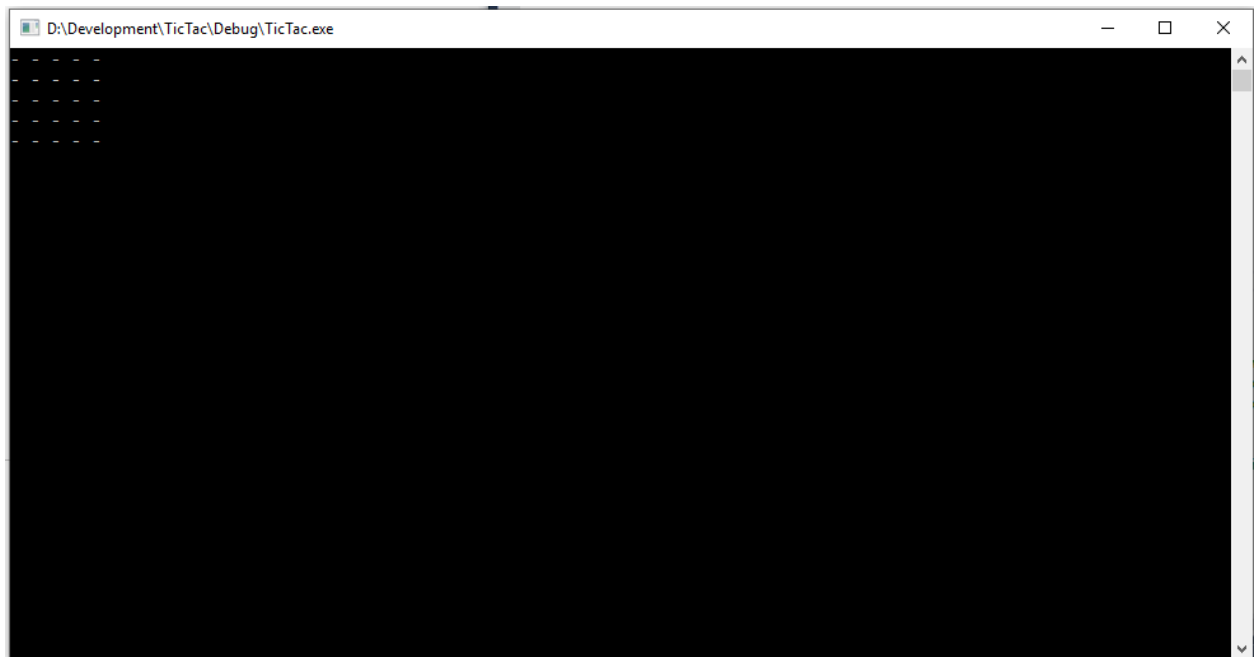
1      }
2
3      void TicTacBoard::SetCell(unsigned int xpos, unsigned int ypos, CellType ct)
4      {
5          cells[ypos][xpos] = ct;
6      }
7

```

Для тестирования создаем в основной программе игровое поле, выводим его на экран.

```
TicTac (Глобальная...)
1  // TicTac.cpp : Этот файл содержит функцию "main".
2  //
3
4  #include "pch.h"
5  #include <iostream>
6  #include "TicTacBoard.h"
7
8  int main()
9  {
10     TicTacBoard* board;
11     board = new TicTacBoard(5);
12     board->Show();
13     _getch();
14     delete board;
15 }
16
17 // Запуск программы: CTRL+F5 или меню "Отладка" > "Отладка программы..."
18 // Отладка программы: F5 или меню "Отладка" > "Отладка программы..."
```

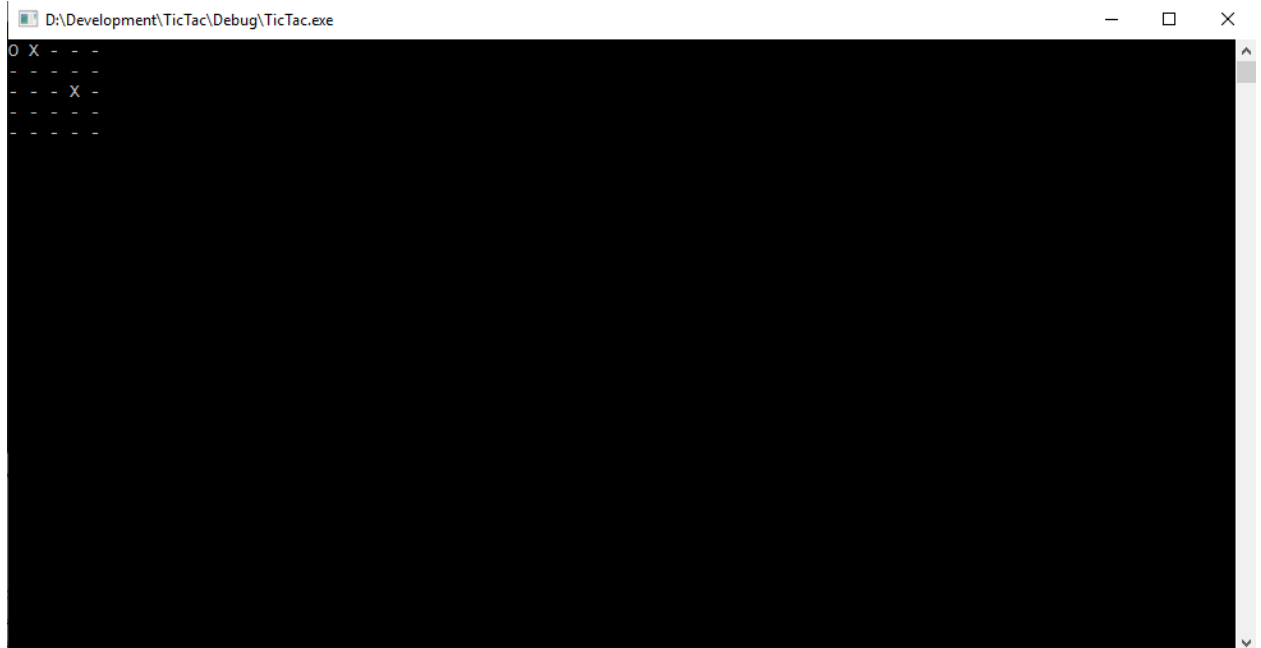
Запускаем программу.



Проверим вывод с установленными знаками.

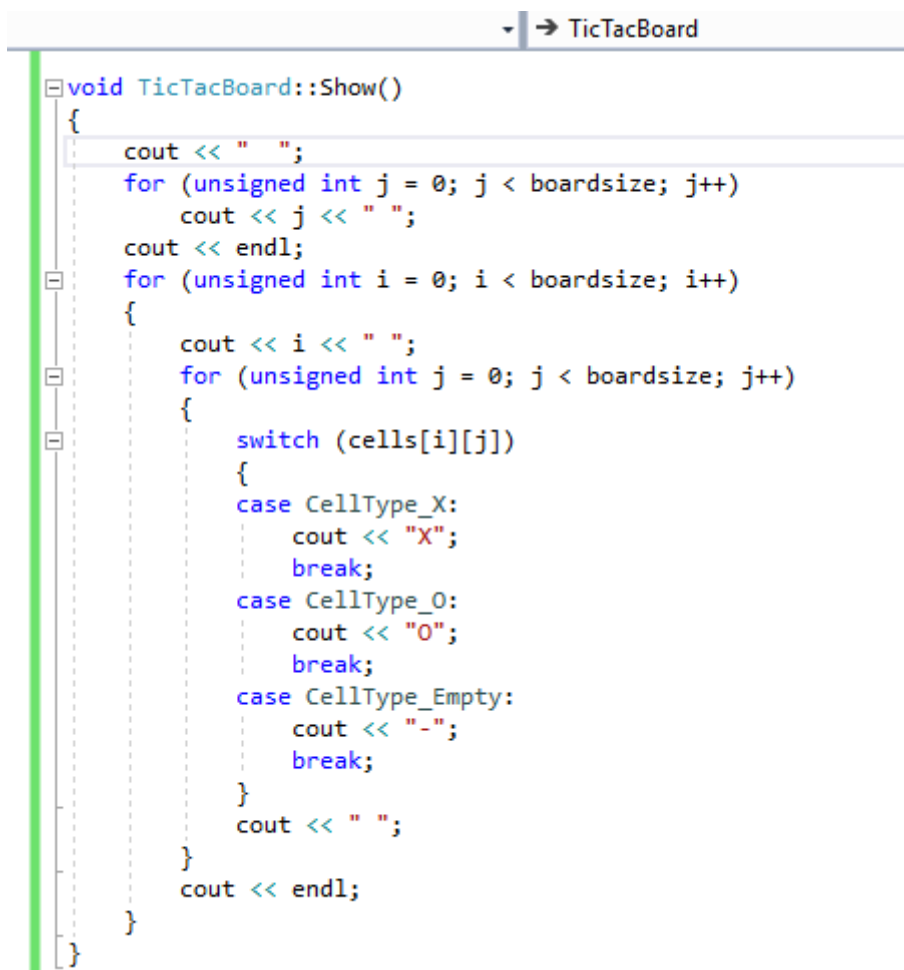
```
6  #include "TicTacBoard.h"
7
8  int main()
9  {
10     TicTacBoard* board;
11     board = new TicTacBoard(5);
12     board->SetCell(0, 0, CellType_0);
13     board->SetCell(1, 0, CellType_X);
14     board->SetCell(3, 2, CellType_X);
15     board->Show();
16     _getch();
17     delete board;
18 }
19
20 // Запуск программы: CTRL+F5 или меню "Отладка" > "Отладка программы..."
```

Запускаем программу.



```
0 X - - -
- - - - -
- - X - -
- - - - -
```

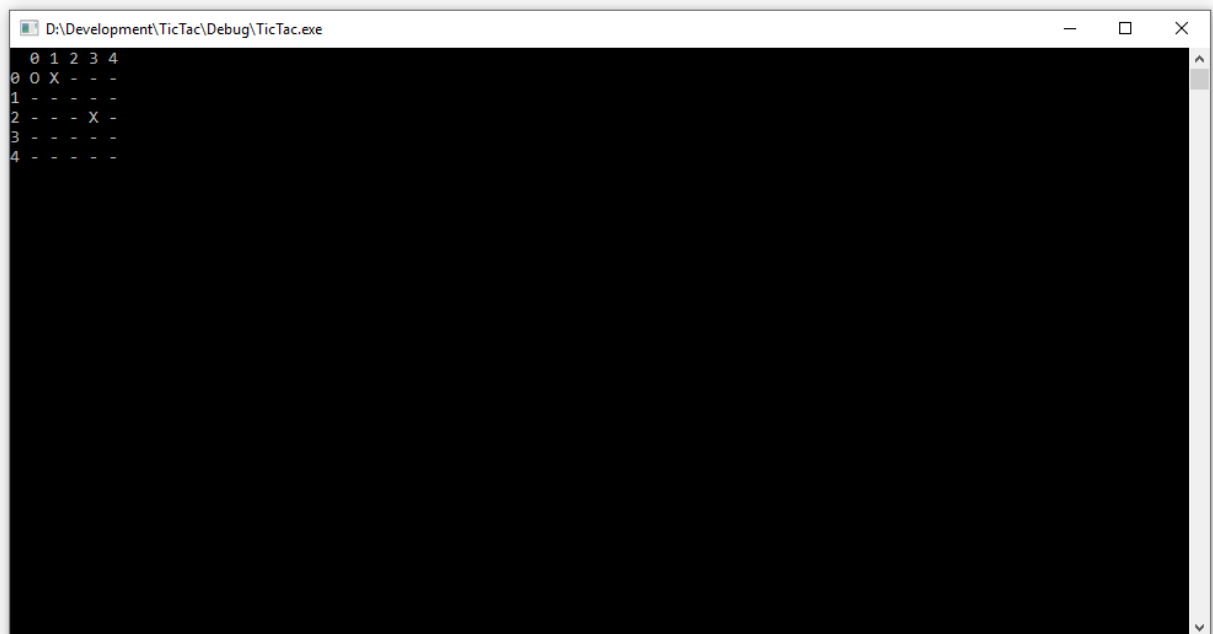
Вывод работает, но как обратиться к конкретному полю из пользовательского интерфейса – неочевидно. Модифицируем метод `TicTacBoard::Show` для показа номеров строк и столбцов.



```
void TicTacBoard::Show()
{
    cout << " ";
    for (unsigned int j = 0; j < boardsize; j++)
        cout << j << " ";
    cout << endl;
    for (unsigned int i = 0; i < boardsize; i++)
    {
        cout << i << " ";
        for (unsigned int j = 0; j < boardsize; j++)
        {
            switch (cells[i][j])
            {
                case CellType_X:
                    cout << "X";
                    break;
                case CellType_0:
                    cout << "0";
                    break;
                case CellType_Empty:
                    cout << "-";
                    break;
            }
            cout << " ";
        }
        cout << endl;
    }
}
```

Проверяем.

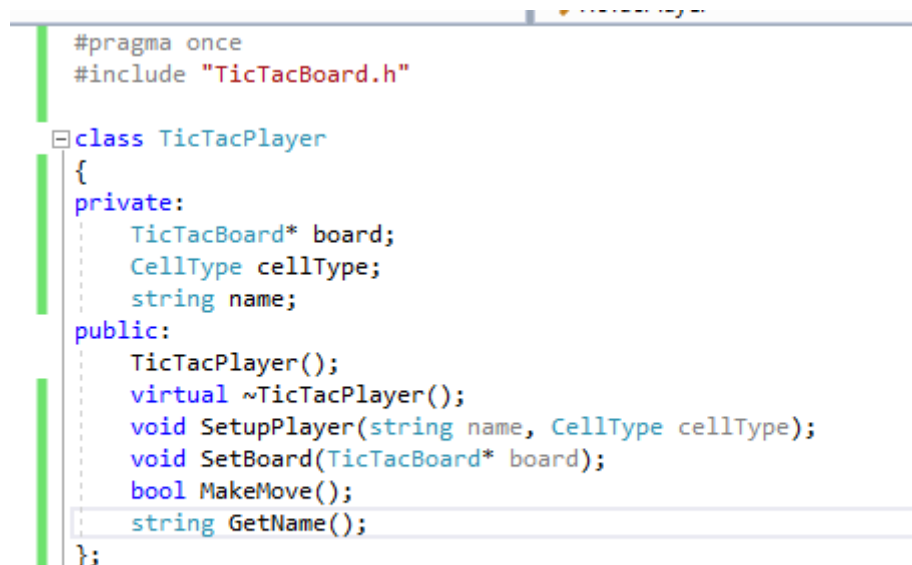




```
D:\Development\TicTac\Debug\TicTac.exe
0 1 2 3 4
0 0 X - -
1 - - - -
2 - - X -
3 - - - -
4 - - - -
```

Наполняем класс игрока.

Игрок должен иметь поля, отвечающие за его характеристики и видеть игровое поле, на котором он играет. В описание добавляем методы установки данных игрока (SetupPlayer), установки игрового поля (SetBoard), получения имени (GetName) и выполнения хода (MakeMove). Функция выполнения хода возвращает true, если ход выполнен и false, если ход был недопустим.



```
#pragma once
#include "TicTacBoard.h"

class TicTacPlayer
{
private:
    TicTacBoard* board;
    CellType cellType;
    string name;
public:
    TicTacPlayer();
    virtual ~TicTacPlayer();
    void SetupPlayer(string name, CellType cellType);
    void SetBoard(TicTacBoard* board);
    bool MakeMove();
    string GetName();
};
```

Для поддержки работы с типом string из библиотеки std добавляем в pch.h необходимые заголовочные файлы. На всякий случай, добавим и conio.h.

```

5      // 4. В окне "Список ошибок" мож
6      // 5. Последовательно выберите п
7      // 6. Чтобы снова открыть этот п
8
9      #ifndef PCH_H
10     #define PCH_H
11
12     // TODO: add headers that you want
13     #include <iostream>
14     using namespace std;
15
16     #include <conio.h>
17     #include <string>
18     #endif //PCH_H
19

```

Реализуем функции игрока.

```

}
1 void TicTacPlayer::SetupPlayer(string name, CellType cellType)
2 {
3     this->name = name;
4     this->cellType = cellType;
5 }
6
7 void TicTacPlayer::SetBoard(TicTacBoard* board)
8 {
9     this->board = board;
10 }
11
12 bool TicTacPlayer::MakeMove()
13 {
14     return true;
15 }
16

```

```

17 string TicTacPlayer::GetName()
18 {
19     return this->name;
20 }
21

```

Все функции реализованы, кроме MakeMove, вместо которой пока установлена заглушка.

Возвращаемся к игровому полю. Добавляем метод CheckLegal, который проверяет возможность хода в заданной позиции.

```

TicTacManager.cpp  TicTacPlayer.cpp  TicTacBoard.h  TicTacBoard.cpp
: Tac
1  #pragma once
2  #include "CellType.h"
3
4  class TicTacBoard
5  {
6  private:
7      unsigned int boardsize;
8      CellType** cells;
9  public:
10     TicTacBoard(unsigned int size);
11     virtual ~TicTacBoard();
12     void Show();
13     void SetCell(unsigned int xpos, unsigned int ypos, CellType ct);
14     bool CheckLegal(unsigned int xpos, unsigned int ypos);
15 };
16
17

```

Согласно логике, ход невозможен, если он за пределами поля или если ячейка не пустая.

```

bool TicTacBoard::CheckLegal(unsigned int xpos, unsigned int ypos)
{
    if ((xpos < 0) || (xpos > boardsize - 1) || (ypos < 0) || (ypos > boardsize - 1))
        return false;
    return (cells[ypos][xpos] == CellType_Empty);
}

```

Наполняем менеджер игры. В игре участвуют поле (board), игроки (player1 и player2). Также менеджер должен знать чей сейчас ход (currentPlayer).

```

anager.cpp  TicTacPlayer.cpp  TicTacBoard.h
c
1  #pragma once
2  #include "TicTacBoard.h"
3  #include "TicTacPlayer.h"
4
5  class TicTacManager
6  {
7  private:
8      TicTacBoard* board;
9      TicTacPlayer* player1;
10     TicTacPlayer* player2;
11     TicTacPlayer* currentPlayer;
12  public:
13     TicTacManager();
14     virtual ~TicTacManager();
15 };

```

Инициализацию можно делать в конструкторе, однако, если предусмотрена логика, которая требует взаимодействия с пользователем и инициализация может быть отменена, то рекомендуется выносить инициализацию в отдельный метод (Init). В нашем методе Init создаются поле и игроки. Необходимо обратить внимание на особенности, связанные с работой с типом string и строку cin.ignore. Что будет, если не использовать cin.ignore?

```

bool TicTacManager::Init()
{
    unsigned int boardsize;
    string playerName;
    cout << "Введите число клеток поля (3-6): ";
    cin >> boardsize;
    if ((boardsize < 3) || (boardsize > 6))
    {
        cout << "Неверное число клеток поля" << endl;
        return false;
    }
    this->board = new TicTacBoard(boardsize);
    this->player1 = new TicTacPlayer();
    this->player2 = new TicTacPlayer();
    cin.ignore();
    cout << "Введите имя игрока, играющего X: ";
    getline(cin, playerName);
    player1->SetupPlayer(playerName, CellType_X);
    cout << "Введите имя игрока, играющего O: ";
    getline(cin, playerName);
    player2->SetupPlayer(playerName, CellType_O);
    player1->SetBoard(this->board);
    player2->SetBoard(this->board);
    currentPlayer = player1;

    return true;
}

```

Очищаем данные в деструкторе.

```

TicTacManager::~TicTacManager()
{
    delete this->player2;
    delete this->player1;
    delete this->board;
}

```

Добавляем в описание класса метод Init, а также метод ShowBoard.

```

#pragma once
#include "TicTacBoard.h"
#include "TicTacPlayer.h"

class TicTacManager
{
private:
    TicTacBoard* board;
    TicTacPlayer* player1;
    TicTacPlayer* player2;
    TicTacPlayer* currentPlayer;
public:
    TicTacManager();
    virtual ~TicTacManager();
    bool Init();
    void ShowBoard();
};

```

Метод ShowBoard вызывает у поля его показ на экране.

```
void TicTacManager::ShowBoard()
{
    this->board->Show();
}
```

Добавляем описание метода MakeMove для выполнения игрового хода.

```
#pragma once
#include "TicTacBoard.h"
#include "TicTacPlayer.h"

class TicTacManager
{
private:
    TicTacBoard* board;
    TicTacPlayer* player1;
    TicTacPlayer* player2;
    TicTacPlayer* currentPlayer;
public:
    TicTacManager();
    virtual ~TicTacManager();
    bool Init();
    void ShowBoard();
    void MakeMove();
};
```

Делаем первую версию реализации. Если вызов MakeMove у текущего игрока вернул false, т.е. ход был неверным, то сообщаем, что ход недопустимый и запрашиваем ход снова.

Если все в порядке, меняем текущего игрока.

```
void TicTacManager::MakeMove()
{
    ShowBoard();
    while (!currentPlayer->MakeMove())
    {
        cout << "Недопустимый ход, попробуйте еще раз" << endl;
        ShowBoard();
    }
    currentPlayer = (currentPlayer == player1) ? player2 : player1;
}
```

Наполняем метод MakeMove для игрока. Для введенных координат проверяется верность хода в рамках правил и, если он верен, возвращается true.

```

bool TicTacPlayer::MakeMove()
{
    unsigned int row, col;
    cout << "Игрок " << name << ", ваш ход..." << endl;
    cout << "Введите строку: ";
    cin >> row;
    cout << "Введите столбец: ";
    cin >> col;

    if (this->board->CheckLegal(col, row))
    {
        this->board->SetCell(col, row, this->cellType);
        return true;
    }

    return false;
}

```

Переписываем основную функцию. Устанавливаем русский язык, создаем и инициализируем менеджер, запускаем бесконечный цикл ходов.

Не рекомендуется вводить имена игроков русскими буквами. Установка русского языка работает только на вывод данных на экран!

```

#include "pch.h"
#include <iostream>
#include "TicTacManager.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    TicTacManager manager;
    if (!manager.Init())
    {
        cout << "Неверные данные, выходим...";
        _getch();
        return 0;
    }
    while (true)
    {
        manager.MakeMove();
    }
}

```

Запускаем приложение. Тестируем, пробуем вводить допустимые и недопустимые ходы и проверяем результат.

```
D:\Development\TicTac\Debug\TicTac.exe
Введите число клеток поля (3-6): 4
Введите имя игрока, играющего X: Ivan
Введите имя игрока, играющего O: Petr
  0 1 2 3
0 - - - -
1 - - - -
2 - - - -
3 - - - -
Игрок Ivan, ваш ход...
Введите строку: 0
Введите столбец: 0
  0 1 2 3
0 X - - -
1 - - - -
2 - - - -
3 - - - -
Игрок Petr, ваш ход...
Введите строку: 2
Введите столбец: 3
  0 1 2 3
0 X - - -
1 - - - -
2 - - - O
3 - - - -
Игрок Ivan, ваш ход...
Введите строку:
```

```
D:\Development\TicTac\Debug\TicTac.exe
1 - - - -
2 - - - -
3 - - - -
Игрок Ivan, ваш ход...
Введите строку: 0
Введите столбец: 0
  0 1 2 3
0 X - - -
1 - - - -
2 - - - -
3 - - - -
Игрок Petr, ваш ход...
Введите строку: 2
Введите столбец: 3
  0 1 2 3
0 X - - -
1 - - - -
2 - - - O
3 - - - -
Игрок Ivan, ваш ход...
Введите строку: 6
Введите столбец: 2
Недопустимый ход, попробуйте еще раз
  0 1 2 3
0 X - - -
1 - - - -
2 - - - O
3 - - - -
Игрок Ivan, ваш ход...
Введите строку:
```

Приложение работает, но пока не определяется конечная ситуация (ничья или победа). Займемся ее добавлением.

Рассмотрим ситуации, когда игра заканчивается:

1. Одна из строк заполнена одинаковыми знаками.
2. Один из столбцов заполнен одинаковыми знаками.
3. Одна из диагоналей заполнена одинаковыми знаками.
4. Все поле заполнено.

Легко заметить, что в случаях 1-3 игра заканчивается победой одного из игроков. В случае 4 будет либо победа (если выполнится одно из условий 1-3), либо ничья. Для случая 4 исключить рассмотрение отдельного варианта победы возможно, рассмотрев изначально условия 1-3, тогда, если они не выполнились, то исход – ничья.

Введем два метода.

CheckEndCondition – будет возвращать true, если на поле ситуаций, при которой игра закончена.

IsVictory – будет возвращать true, если возникло состояние победы одного из игроков. Для поддержки этого запроса введем поле bVictory, которое будет установлено в состоянии победы.

Для упрощения задачу разобьем на несколько (согласно перечисленным условиям). Введем методы проверки строк, столбцов, диагоналей и заполнения поля.

```
#pragma once
#include "CellType.h"

class TicTacBoard
{
private:
    unsigned int boardsize;
    CellType** cells;
    bool bVictory = false;
    bool IsRowMade(unsigned int row);
    bool IsColumnMade(unsigned int col);
    bool IsDiagMade();
    bool IsBoardFull();
public:
    TicTacBoard(unsigned int size);
    virtual ~TicTacBoard();
    void Show();
    void SetCell(unsigned int xpos, unsigned int ypos, CellType ct);
    bool CheckLegal(unsigned int xpos, unsigned int ypos);
    bool CheckEndCondition();
    bool IsVictory();
};
```

Реализуем их.

```
bool TicTacBoard::IsRowMade(unsigned int row)
{
    int numX = 0, numO = 0;
    for (unsigned int i = 0; i < boardsize; i++)
    {
        if (cells[row][i] == CellType_O)
            numO++;
        if (cells[row][i] == CellType_X)
            numX++;
    }

    if ((numX == boardsize) || (numO == boardsize))
    {
        bVictory = true;
        return true;
    }

    return false;
}
```



```

bool TicTacBoard::IsColumnMade(unsigned int col)
{
    int numX = 0, numO = 0;
    for (unsigned int i = 0; i < boardsize; i++)
    {
        if (cells[i][col] == CellType_O)
            numO++;
        if (cells[i][col] == CellType_X)
            numX++;
    }

    if ((numX == boardsize) || (numO == boardsize))
    {
        bVictory = true;
        return true;
    }

    return false;
}

```

```

bool TicTacBoard::IsDiagMade()
{
    int numX = 0, numO = 0;
    for (unsigned int i = 0; i < boardsize; i++)
    {
        if (cells[i][i] == CellType_O)
            numO++;
        if (cells[i][i] == CellType_X)
            numX++;
    }

    if ((numX == boardsize) || (numO == boardsize))
    {
        bVictory = true;
        return true;
    }

    numX = numO = 0;
    for (unsigned int i = 0; i < boardsize; i++)
    {
        if (cells[i][boardsize - i - 1] == CellType_O)
            numO++;
        if (cells[i][boardsize - i - 1] == CellType_X)
            numX++;
    }

    if ((numX == boardsize) || (numO == boardsize))
    {
        bVictory = true;
        return true;
    }

    return false;
}

```

```

bool TicTacBoard::IsBoardFull()
{
    int numX = 0, numO = 0;
    for (unsigned int i = 0; i < boardsize; i++)
    {
        for (unsigned int j = 0; j < boardsize; j++)
        {
            if (cells[i][j] == CellType_O)
                numO++;
            if (cells[i][j] == CellType_X)
                numX++;
        }
    }
    if ((numX + numO) == (boardsize * boardsize))
        return true;

    return false;
}

```

Используя приведенные выше методы реализуем CheckEndCondition.

```

bool TicTacBoard::CheckEndCondition()
{
    for (unsigned int i = 0; i < boardsize; i++)
    {
        if (IsRowMade(i) || IsColumnMade(i))
            return true;
    }
    if (IsDiagMade() || IsBoardFull())
        return true;

    return false;
}

```

Простая реализация IsVictory.

```

bool TicTacBoard::IsVictory()
{
    return bVictory;
}

```

Расширяем менеджер, добавляя проверку, что игра завершена.

Метод IsGameFinished возвращает true, если игра завершилась. Для работы этого метода вводится поле bGameFinished.

```

#pragma once
#include "TicTacBoard.h"
#include "TicTacPlayer.h"

class TicTacManager
{
private:
    TicTacBoard* board;
    TicTacPlayer* player1;
    TicTacPlayer* player2;
    TicTacPlayer* currentPlayer;
    bool bGameFinished = false;
public:
    TicTacManager();
    virtual ~TicTacManager();
    bool Init();
    void ShowBoard();
    void MakeMove();
    bool IsGameFinished();
};

bool TicTacManager::IsGameFinished()
{
    return bGameFinished;
}

```

Дорабатываем MakeMove, вводя проверку на окончание партии и вывод информации о победе или ничьей.

```

void TicTacManager::MakeMove()
{
    ShowBoard();
    while (!currentPlayer->MakeMove())
    {
        cout << "Недопустимый ход, попробуйте еще раз" << endl;
        ShowBoard();
    }
    if (this->board->CheckEndCondition())
    {
        if (this->board->IsVictory())
            cout << "Игрок " << currentPlayer->GetName() << " победил!" << endl;
        else
            cout << "Ничья!" << endl;
        this->bGameFinished = true;
        ShowBoard();
        return;
    }
    currentPlayer = (currentPlayer == player1) ? player2 : player1;
}

```

Дорабатываем основную программу, вводим в качестве условия выхода из приложения значение, возвращаемое IsGameFinished.

```

#include "pch.h"
#include <iostream>
#include "TicTacManager.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    TicTacManager manager;
    if (!manager.Init())
    {
        cout << "Неверные данные, выходим...";
        _getch();
        return 0;
    }
    while (!manager.IsGameFinished())
    {
        manager.MakeMove();
    }
    _getch();
}

```

Проводим тестирование разных игровых ситуаций и концовок и убеждаемся, что приложение работает и удовлетворяет всем нашим условиям.

```

D:\Development\TicTac\Debug\TicTac.exe
0 X X - -
1 - 0 - -
2 - - X 0
3 0 - - -
Игрок Ivan, ваш ход...
Введите строку: 0
Введите столбец: 2
  0 1 2 3
0 X X X -
1 - 0 - -
2 - - X 0
3 0 - - -
Игрок Petr, ваш ход...
Введите строку: 3
Введите столбец: 1
  0 1 2 3
0 X X X -
1 - 0 - -
2 - - X 0
3 0 0 - -
Игрок Ivan, ваш ход...
Введите строку: 0
Введите столбец: 3
Игрок Ivan победил!
  0 1 2 3
0 X X X X
1 - 0 - -
2 - - X 0
3 0 0 - -

```

```
D:\Development\TicTac\Debug\TicTac.exe
Игрок Petr, ваш ход...
Введите строку: 0
Введите столбец: 2
  0 1 2
0 X X 0
1 0 0 -
2 X - -
Игрок Ivan, ваш ход...
Введите строку: 1
Введите столбец: 2
  0 1 2
0 X X 0
1 0 0 X
2 X - -
Игрок Petr, ваш ход...
Введите строку: 2
Введите столбец: 1
  0 1 2
0 X X 0
1 0 0 X
2 X 0 -
Игрок Ivan, ваш ход...
Введите строку: 2
Введите столбец: 2
Ничья!
  0 1 2
0 X X 0
1 0 0 X
2 X 0 X
```