**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по объектно-ориентированному программированию

«Разработка графического редактора на языке C++ с использованием механизмов ООП»

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Седых О.М.

Группа ПМ-21-1

Руководитель

Доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кургасов В.В.

Липецк 2022 г.

Оглавление

[Цель работы 3](#_Toc122339716)

[Задание кафедры 3](#_Toc122339717)

[Код программы 4](#_Toc122339718)

[Примеры работы программы 5](#_Toc122339719)

[Выводы 12](#_Toc122339720)

[Ответы на контрольные вопросы 13](#_Toc122339721)

**Цель работы**

Закрепить навыки использования механизмов ООП на примере реализации графического редактора.

**Задание кафедры**

Реализовать на языке C++ редактор графической схемы. В ходе выполнения работы обязательно применение объектно-ориентированных возможностей языка C++: наследования и динамического полиморфизма. Каждый тип элемента схемы должен быть представлен в программе в виде отдельного класса, который наследован от базового класса «графический элемент» (имеющего чисто виртуальную функцию прорисовки). Также необходим один класс «поле рисования», который содержит все графические элементы и отвечает за вызов функций прорисовки. Хранение графических элементов осуществляется с использованием контейнеров стандартной библиотеки C++.

Векторные редакторы

|  |  |
| --- | --- |
| Вариант | Редактор |
| 13 | Позиции игроков на хоккейной площадке |

**Код программы**

Программа написана на языке C++ с подключением фреймворка C++/CLI.

Файл Hockey.h:

#pragma once

#include <string>

#include <vector>

namespace Hockey {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace std;

//создание константы для хранения размера игрока

const float SIZE\_PLAYER = 15.0F;

//класс Игрок

class Player {

private:

//его поля

//координата x

float x;

//координата y

float y;

//булевая переменная для хранения типа команды

bool team;

public:

//чисто виртуальный метод прорисовки

virtual void draw(Image ^image) = 0;

//чисто виртуальный метод, возвращающий тип игрока в виде строки

virtual string get\_type\_player() = 0;

//метод, возвращающий координату x

float get\_x() {

return x;

}

//метод, возвращающий координату y

float get\_y() {

return y;

}

//метод для присваивания координатам текущего игрока новых значений

void set\_coordinates(float x, float y) {

this->x = x;

this->y = y;

}

//метод, возвращающий тип команды

bool get\_team() {

return team;

}

//метод для присваивания типу команды текущего игрока нового типа

void set\_team(bool team) {

this->team = team;

}

};

//класс Вратарь - наследник класса Игрок

class Goalkeeper : public Player {

//реализация виртуального метода, возвращающего тип игрока

virtual string get\_type\_player() {

return "goalkeeper";

}

//реализация виртуального метода прорисовки

virtual void draw(Image ^image) {

//получение указателя на объект класса Graphics (статический метод FromImage()

//класса Graphics возвращает указатель на объект класса, который позволяет

//рисовать на указанном в качестве параметра изображении)

Graphics^ g = Graphics::FromImage(image);

//если команда красная (то есть функция get\_team() имеет значени true), рисуем

//красный прямоугольник, обозначающий вратаря

if (get\_team()) {

g->FillRectangle(Brushes::Red, get\_x() - SIZE\_PLAYER / 2, get\_y() - SIZE\_PLAYER / 2, SIZE\_PLAYER, SIZE\_PLAYER);

}

//если команда синяя (то есть функция get\_team() имеет значени false), рисуем

//синий прямоугольник, обозначающий вратаря

else {

g->FillRectangle(Brushes::Blue, get\_x() - SIZE\_PLAYER / 2, get\_y() - SIZE\_PLAYER / 2, SIZE\_PLAYER, SIZE\_PLAYER);

}

}

};

//класс Защитник - наследник класса Игрок

class Defender : public Player {

//реализация виртуального метода, возвращающего тип игрока

virtual string get\_type\_player() {

return "defender";

}

//реализация виртуального метода прорисовки

virtual void draw(Image ^image) {

//получение указателя на объект класса Graphics (статический метод FromImage()

//класса Graphics возвращает указатель на объект класса, который позволяет

//рисовать на указанном в качестве параметра изображении)

Graphics^ g = Graphics::FromImage(image);

//создание указателя на массив вершин многоугольника (в данном случае

//ромба, которым обозначается защитник)

cli::array<PointF> ^points = gcnew cli::array<PointF>(4);

//присваивание координтам вершинам соответствующих значений

points[0] = PointF(get\_x(), get\_y() - SIZE\_PLAYER / 2);

points[1] = PointF(get\_x() - SIZE\_PLAYER / 2, get\_y());

points[2] = PointF(get\_x(), get\_y() + SIZE\_PLAYER / 2);

points[3] = PointF(get\_x() + SIZE\_PLAYER / 2, get\_y());

//если команда красная (то есть функция get\_team() имеет значени true), рисуем

//красный ромб, обозначающий защитника

if (get\_team()) {

g->FillPolygon(Brushes::Red, points);

}

//если команда синяя (то есть функция get\_team() имеет значени false), рисуем

//синий ромб, обозначающий защитника

else {

g->FillPolygon(Brushes::Blue, points);

}

}

};

//класс Нападающий - наследник класса Игрок

class Offensive : public Player {

//реализация виртуального метода, возвращающего тип игрока

virtual string get\_type\_player() {

return "offensive";

}

//реализация виртуального метода прорисовки

virtual void draw(Image ^image) {

//получение указателя на объект класса Graphics (статический метод FromImage()

//класса Graphics возвращает указатель на объект класса, который позволяет

//рисовать на указанном в качестве параметра изображении)

Graphics^ g = Graphics::FromImage(image);

//создание указателя на массив вершин многоугольника (в данном случае

//треугольника, которым обозначается защитник)

cli::array<PointF> ^points = gcnew cli::array<PointF>(3);

//присваивание координтам вершинам соответствующих значений

points[0] = PointF(get\_x() - SIZE\_PLAYER / 2, get\_y() - SIZE\_PLAYER / 2);

points[1] = PointF(get\_x() - SIZE\_PLAYER / 2, get\_y() + SIZE\_PLAYER / 2);

points[2] = PointF(get\_x() + SIZE\_PLAYER / 2, get\_y());

//если команда красная (то есть функция get\_team() имеет значени true), рисуем

//красный треугольник, обозначающий нападающего

if (get\_team()) {

g->FillPolygon(Brushes::Red, points);

}

//если команда синяя (то есть функция get\_team() имеет значени false), рисуем

//синий треугольник, обозначающий нападающего

else {

g->FillPolygon(Brushes::Blue, points);

}

}

};

//класс Поле рисования

class PlayingField {

private:

//поля класса

//вектор указателей на объекты класса Игрок

vector<Player\*> players;

//переменная для хранения индекса перемещаемого по полю игрока

int index\_moved\_player;

public:

//в конструкторе перменной для хранения индекса перемещаемого по полю игрока

//присваивается -1 (начальное значение)

PlayingField() {

index\_moved\_player = -1;

}

//в декструкторе очищается память из-под каждого элемента вектора

~PlayingField() {

for (int i = 0; i < players.size(); i++) {

delete players[i];

}

}

//метод прорисовки всего поля (параметр - указатель на изображение, то есть на объект

//класса Image)

void draw\_field(Image^ image) {

//в цикле для каждого игрока (каждого элемента вектора players) вызывается

//метод прорисовки

for (int i = 0; i < players.size(); i++) {

players[i]->draw(image);

}

}

//метод подсчёта игроков каждого вида

//(параметры - тип игрока и тип команды)

int counter(string type\_player, bool type\_team) {

int count = 0;

//в цикле считается количество всех игроков данной команды и данного типа

for (int i = 0; i < players.size(); i++) {

if ((players[i]->get\_type\_player() == type\_player) && (players[i]->get\_team() == type\_team)) {

count++;

}

}

return count;

}

//метод поиска игрока с заданными координатами на поле

int find(float x, float y) {

//в цикле осуществляется проверка для каждого элемента вектора: попадает ли игрок с

//заданными координатами в область, накрывающую какого-либо игрока из вектора

//если такой игрок найден, возвращается его индекс

for (int i = 0; i < players.size(); i++) {

if ((abs(players[i]->get\_x() - x) <= SIZE\_PLAYER / 2) && (abs(players[i]->get\_y() - y) <= SIZE\_PLAYER / 2)) {

return i;

}

}

//если такой игрок не был найден, возвращается -1

return -1;

}

//метод добавления нового игрока

//(параметры: тип игрока, тип команды, координаты точки, в которую нужно добавить игрока)

bool add(string type\_player, bool type\_team, float x, float y) {

//дествия в случае добавления вратаря

if (type\_player == "goalkeeper") {

//больше одного вратаря в команде быть не может, поэтому делаем проверку с помощью

//метода counter(), что вратарей < 1 (тогда можно добавлять)

if (counter(type\_player, type\_team) < 1) {

//создание указателя на объект класса Вратарь

Goalkeeper\* goalkeeper = new Goalkeeper();

//присваивание новому объекту переданных в функцию координат

goalkeeper->set\_coordinates(x, y);

//присваивание новому объекту переданного в функцию

//типа команды

goalkeeper->set\_team(type\_team);

//добавление нового игрока в вектор

players.push\_back(goalkeeper);

//если игрок добавлен, функция возвращает true

return true;

}

//если игрок не добавлен, функция возвращает false

else {

return false;

}

}

//дествия в случае добавления защитника

else if (type\_player == "defender") {

//больше двух защитников в команде быть не может, поэтому делаем проверку с помощью

//метода counter(), что защитников < 2 (тогда можно добавлять)

if (counter(type\_player, type\_team) < 2) {

//создание указателя на объект класса Защитник

Defender\* defender = new Defender();

//присваивание новому объекту переданных в функцию координат

defender->set\_coordinates(x, y);

//присваивание новому объекту переданного в функцию

//типа команды

defender->set\_team(type\_team);

//добавление нового игрока в вектор

players.push\_back(defender);

//если игрок добавлен, функция возвращает true

return true;

}

//если игрок не добавлен, функция возвращает false

else {

return false;

}

}

//действия в случае добавления нападающего

else if (type\_player == "offensive") {

//больше трёх нападающих в команде быть не может, поэтому делаем проверку с помощью

//метода counter(), что нападающих < 3 (тогда можно добавлять)

if (counter(type\_player, type\_team) < 3) {

//создание указателя на объект класса Нападающий

Offensive\* offensive = new Offensive();

//присваивание новому объекту переданных в функцию координат

offensive->set\_coordinates(x, y);

//присваивание новому объекту переданного в функцию

//типа команды

offensive->set\_team(type\_team);

//добавление нового игрока в вектор

players.push\_back(offensive);

//если игрок добавлен, функция возвращает true

return true;

}

//если игрок не добавлен, функция возвращает false

else {

return false;

}

}

}

//функция удаления игрока

bool del(float x, float y) {

//записываем в переменную index значение функции find(x, y)

int index = find(x, y);

//если игрок найден, функция удаляет игрока; удаляет указатель на этого игрока из

//вектора и возвращает true

if (index != -1) {

delete players[index];

players.erase(players.begin() + index);

return true;

}

//если игрок не найден, функция возвращает false

else {

return false;

}

}

//метод начала перемещения игрока по полю

void start\_moving(float x, float y) {

//переменной index\_moved\_player присваивается значение индекса перемещаемого

//игрока

index\_moved\_player = find(x, y);

}

//метод перемещения игрока

bool move(float x, float y) {

//если индекс игрока -1, то есть игрок с переданными в функцию координатами не найден,

//функция возвращает false

if (index\_moved\_player == -1) {

return false;

}

//если игрок имеет индекс, отличный от -1, координатам игрока с данным индексом

//присваиваются значения переданных в функцию координат, и функция

//возвращает true

else {

players[index\_moved\_player]->set\_coordinates(x, y);

return true;

}

}

//метод окончания перемещения

void end\_moving() {

//переменной для хранения индекса игрока присваивается значение -1, так как

//перемещение завершено

index\_moved\_player = -1;

}

};

//создание глобальной переменной - объекта класса Игровое поле

PlayingField playingField;

}

Файл MyForm.h:

#pragma once

#include "Hockey.h"

namespace Hockey {

/// <summary>

/// Сводка для MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: добавьте код конструктора

//

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::PictureBox^ pictureBox1;

private: System::Windows::Forms::Button^ add\_goalkeeper\_left;

protected:

protected:

private: System::Windows::Forms::Button^ add\_defender\_left;

private: System::Windows::Forms::Button^ add\_offensive\_left;

private: System::Windows::Forms::Button^ add\_goalkeeper\_right;

private: System::Windows::Forms::Button^ add\_depender\_right;

private: System::Windows::Forms::Button^ add\_offensive\_right;

protected:

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

//метод перерисовки поля

void redraw\_field() {

//создаётся объект класса Bitmap с изображением пустого поля

Bitmap^ image = gcnew Bitmap("field.png");

//вызывается метод рисования игроков

playingField.draw\_field(image);

//полю Image объекта класса pictureBox присваивается изменённое изображение

pictureBox1->Image = image;

}

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(MyForm::typeid));

this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());

this->add\_goalkeeper\_left = (gcnew System::Windows::Forms::Button());

this->add\_defender\_left = (gcnew System::Windows::Forms::Button());

this->add\_offensive\_left = (gcnew System::Windows::Forms::Button());

this->add\_goalkeeper\_right = (gcnew System::Windows::Forms::Button());

this->add\_depender\_right = (gcnew System::Windows::Forms::Button());

this->add\_offensive\_right = (gcnew System::Windows::Forms::Button());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->BeginInit();

this->SuspendLayout();

//

// pictureBox1

//

//this->pictureBox1->Image = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"pictureBox1.Image")));

this->pictureBox1->Image = gcnew Bitmap("field.png");

this->pictureBox1->Location = System::Drawing::Point(36, 148);

this->pictureBox1->Name = L"pictureBox1";

this->pictureBox1->Size = System::Drawing::Size(1200, 622);

this->pictureBox1->SizeMode = System::Windows::Forms::PictureBoxSizeMode::StretchImage;

this->pictureBox1->TabIndex = 0;

this->pictureBox1->TabStop = false;

this->pictureBox1->MouseClick += gcnew System::Windows::Forms::MouseEventHandler(this, &MyForm::pictureBox1\_MouseClick);

this->pictureBox1->MouseDown += gcnew System::Windows::Forms::MouseEventHandler(this, &MyForm::pictureBox1\_MouseDown);

this->pictureBox1->MouseMove += gcnew System::Windows::Forms::MouseEventHandler(this, &MyForm::pictureBox1\_MouseMove);

this->pictureBox1->MouseUp += gcnew System::Windows::Forms::MouseEventHandler(this, &MyForm::pictureBox1\_MouseUp);

//

// add\_goalkeeper\_left

//

this->add\_goalkeeper\_left->Location = System::Drawing::Point(36, 21);

this->add\_goalkeeper\_left->Name = L"add\_goalkeeper\_left";

this->add\_goalkeeper\_left->Size = System::Drawing::Size(120, 105);

this->add\_goalkeeper\_left->TabIndex = 1;

this->add\_goalkeeper\_left->Text = L"Добавить вратаря";

this->add\_goalkeeper\_left->UseVisualStyleBackColor = true;

this->add\_goalkeeper\_left->Click += gcnew System::EventHandler(this, &MyForm::add\_goalkeeper\_left\_Click);

//

// add\_defender\_left

//

this->add\_defender\_left->Location = System::Drawing::Point(177, 21);

this->add\_defender\_left->Name = L"add\_defender\_left";

this->add\_defender\_left->Size = System::Drawing::Size(120, 105);

this->add\_defender\_left->TabIndex = 5;

this->add\_defender\_left->Text = L"Добавить защитника";

this->add\_defender\_left->UseVisualStyleBackColor = true;

this->add\_defender\_left->Click += gcnew System::EventHandler(this, &MyForm::add\_defender\_left\_Click);

//

// add\_offensive\_left

//

this->add\_offensive\_left->Location = System::Drawing::Point(318, 21);

this->add\_offensive\_left->Name = L"add\_offensive\_left";

this->add\_offensive\_left->Size = System::Drawing::Size(137, 105);

this->add\_offensive\_left->TabIndex = 6;

this->add\_offensive\_left->Text = L"Добавить нападающего";

this->add\_offensive\_left->UseVisualStyleBackColor = true;

this->add\_offensive\_left->Click += gcnew System::EventHandler(this, &MyForm::add\_offensive\_left\_Click);

//

// add\_goalkeeper\_right

//

this->add\_goalkeeper\_right->Location = System::Drawing::Point(814, 21);

this->add\_goalkeeper\_right->Name = L"add\_goalkeeper\_right";

this->add\_goalkeeper\_right->Size = System::Drawing::Size(120, 105);

this->add\_goalkeeper\_right->TabIndex = 7;

this->add\_goalkeeper\_right->Text = L"Добавить вратаря";

this->add\_goalkeeper\_right->UseVisualStyleBackColor = true;

this->add\_goalkeeper\_right->Click += gcnew System::EventHandler(this, &MyForm::add\_goalkeeper\_right\_Click);

//

// add\_depender\_right

//

this->add\_depender\_right->Location = System::Drawing::Point(958, 21);

this->add\_depender\_right->Name = L"add\_depender\_right";

this->add\_depender\_right->Size = System::Drawing::Size(120, 105);

this->add\_depender\_right->TabIndex = 8;

this->add\_depender\_right->Text = L"Добавить защитника";

this->add\_depender\_right->UseVisualStyleBackColor = true;

this->add\_depender\_right->Click += gcnew System::EventHandler(this, &MyForm::add\_defender\_right\_Click);

//

// add\_offensive\_right

//

this->add\_offensive\_right->Location = System::Drawing::Point(1099, 21);

this->add\_offensive\_right->Name = L"add\_offensive\_right";

this->add\_offensive\_right->Size = System::Drawing::Size(137, 105);

this->add\_offensive\_right->TabIndex = 9;

this->add\_offensive\_right->Text = L"Добавить нападающего";

this->add\_offensive\_right->UseVisualStyleBackColor = true;

this->add\_offensive\_right->Click += gcnew System::EventHandler(this, &MyForm::add\_offensive\_right\_Click);

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(9, 20);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(1270, 791);

this->Controls->Add(this->add\_offensive\_right);

this->Controls->Add(this->add\_depender\_right);

this->Controls->Add(this->add\_goalkeeper\_right);

this->Controls->Add(this->add\_offensive\_left);

this->Controls->Add(this->add\_defender\_left);

this->Controls->Add(this->add\_goalkeeper\_left);

this->Controls->Add(this->pictureBox1);

this->Name = L"MyForm";

this->Text = L"Window";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->EndInit();

this->ResumeLayout(false);

}

#pragma endregion

//обработчик события нажатия на кнопку добавления синего вратаря

private: System::Void add\_goalkeeper\_left\_Click(System::Object^ sender, System::EventArgs^ e) {

//проверяем, возможно ли добавить вратаря в данную команду

//если да, он добавляется; иначе выводится сообщение об ошибке

if (!playingField.add("goalkeeper", false, pictureBox1->Image->Size.Width / 2, pictureBox1->Image->Size.Height / 2)) {

MessageBox::Show("Нельзя добавить более одного вратаря для одной команды");

}

//вызывается функция перерисовки игрового поля

redraw\_field();

}

//обработчик события нажатия на кнопку добавления синего защитника

private: System::Void add\_defender\_left\_Click(System::Object^ sender, System::EventArgs^ e) {

//проверяем, возможно ли добавить защитника в данную команду

//если да, он добавляется; иначе выводится сообщение об ошибке

if (!playingField.add("defender", false, pictureBox1->Image->Size.Width / 2, pictureBox1->Image->Size.Height / 2)) {

MessageBox::Show("Нельзя добавить более двух защитников для одной команды");

}

//вызывается функция перерисовки игрового поля

redraw\_field();

}

//обработчик события нажатия на кнопку добавления синего нападающего

private: System::Void add\_offensive\_left\_Click(System::Object^ sender, System::EventArgs^ e) {

//проверяем, возможно ли добавить нападающего в данную команду

//если да, он добавляется; иначе выводится сообщение об ошибке

if (!playingField.add("offensive", false, pictureBox1->Image->Size.Width / 2, pictureBox1->Image->Size.Height / 2)) {

MessageBox::Show("Нельзя добавить более трёх нападающих для одной команды");

}

//вызывается функция перерисовки игрового поля

redraw\_field();

}

//обработчик события нажатия на кнопку добавления красного вратаря

private: System::Void add\_goalkeeper\_right\_Click(System::Object^ sender, System::EventArgs^ e) {

//проверяем, возможно ли добавить вратаря в данную команду

//если да, он добавляется; иначе выводится сообщение об ошибке

if (!playingField.add("goalkeeper", true, pictureBox1->Image->Size.Width / 2, pictureBox1->Image->Size.Height / 2)) {

MessageBox::Show("Нельзя добавить более одного вратаря для одной команды");

}

//вызывается функция перерисовки игрового поля

redraw\_field();

}

//обработчик события нажатия на кнопку добавления красного защитника

private: System::Void add\_defender\_right\_Click(System::Object^ sender, System::EventArgs^ e) {

//проверяем, возможно ли добавить защитника в данную команду

//если да, он добавляется; иначе выводится сообщение об ошибке

if (!playingField.add("defender", true, pictureBox1->Image->Size.Width / 2, pictureBox1->Image->Size.Height / 2)) {

MessageBox::Show("Нельзя добавить более двух защитников для одной команды");

}

//вызывается функция перерисовки игрового поля

redraw\_field();

}

//обработчик события нажатия на кнопку добавления красного нападающего

private: System::Void add\_offensive\_right\_Click(System::Object^ sender, System::EventArgs^ e) {

//проверяем, возможно ли добавить нападающего в данную команду

//если да, он добавляется; иначе выводится сообщение об ошибке

if (!playingField.add("offensive", true, pictureBox1->Image->Size.Width / 2, pictureBox1->Image->Size.Height / 2)) {

MessageBox::Show("Нельзя добавить более трёх нападающих для одной команды");

}

//вызывается функция перерисовки игрового поля

redraw\_field();

}

//обработчик события нажатия на кнопку мыши

private: System::Void pictureBox1\_MouseClick(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

//проверка, что нажата правая кнопка, по которой происходит удаление игрока

if (e->Button == System::Windows::Forms::MouseButtons::Right) {

//координатам для удаления присваиваются нужные значения

float x = e->X \* pictureBox1->Image->Size.Width / pictureBox1->Size.Width;

float y = e->Y \* pictureBox1->Image->Size.Height / pictureBox1->Size.Height;

//если данные координаты не указывают ни на какого игрока, выводится

//сообщение об ошибке; иначе игрок удаляется с помощью метода удаления

if (!playingField.del(x, y)) {

MessageBox::Show("Неверные координаты игрока");

}

//вызывается функция перерисовки игрового поля

redraw\_field();

}

}

//обработчик события начала нажатия кнопки мыши

private: System::Void pictureBox1\_MouseDown(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

//проверка, что нажата левая кнопка мыши

if (e->Button == System::Windows::Forms::MouseButtons::Left) {

//координатам игрока присваиваются нужные значения

float x = e->X \* pictureBox1->Image->Size.Width / pictureBox1->Size.Width;

float y = e->Y \* pictureBox1->Image->Size.Height / pictureBox1->Size.Height;

//вызывается метод начала перемещения игрока с данными координатами по полю

playingField.start\_moving(x, y);

}

}

//обработчик события движения курсора мыши

private: System::Void pictureBox1\_MouseMove(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

//проверка, что нажата левая кнопка мыши

if (e->Button == System::Windows::Forms::MouseButtons::Left) {

//координатам игрока присваиваются нужные значения

float x = e->X \* pictureBox1->Image->Size.Width / pictureBox1->Size.Width;

float y = e->Y \* pictureBox1->Image->Size.Height / pictureBox1->Size.Height;

//вызывается метод перемещения игрока с данными координатами по полю; если игрока

//можно переместить, поле перерисовывается

if (playingField.move(x, y)) {

redraw\_field();

}

}

}

//обработчик события отпускания кнопки мыши

private: System::Void pictureBox1\_MouseUp(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

//проверка, что отпускается левая кнопка;

//вызов метода окончания перемещения игрока

if (e->Button == System::Windows::Forms::MouseButtons::Left) {

playingField.end\_moving();

}

}

};

}

Файл MyForm.cpp:

#include "MyForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThreadAttribute]

void Main(array<String^>^ args) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Hockey::MyForm form;

Application::Run(%form);

}

**Примеры работы программы**

При запуске графического редактора открывается окно с пустой хоккейной площадкой.

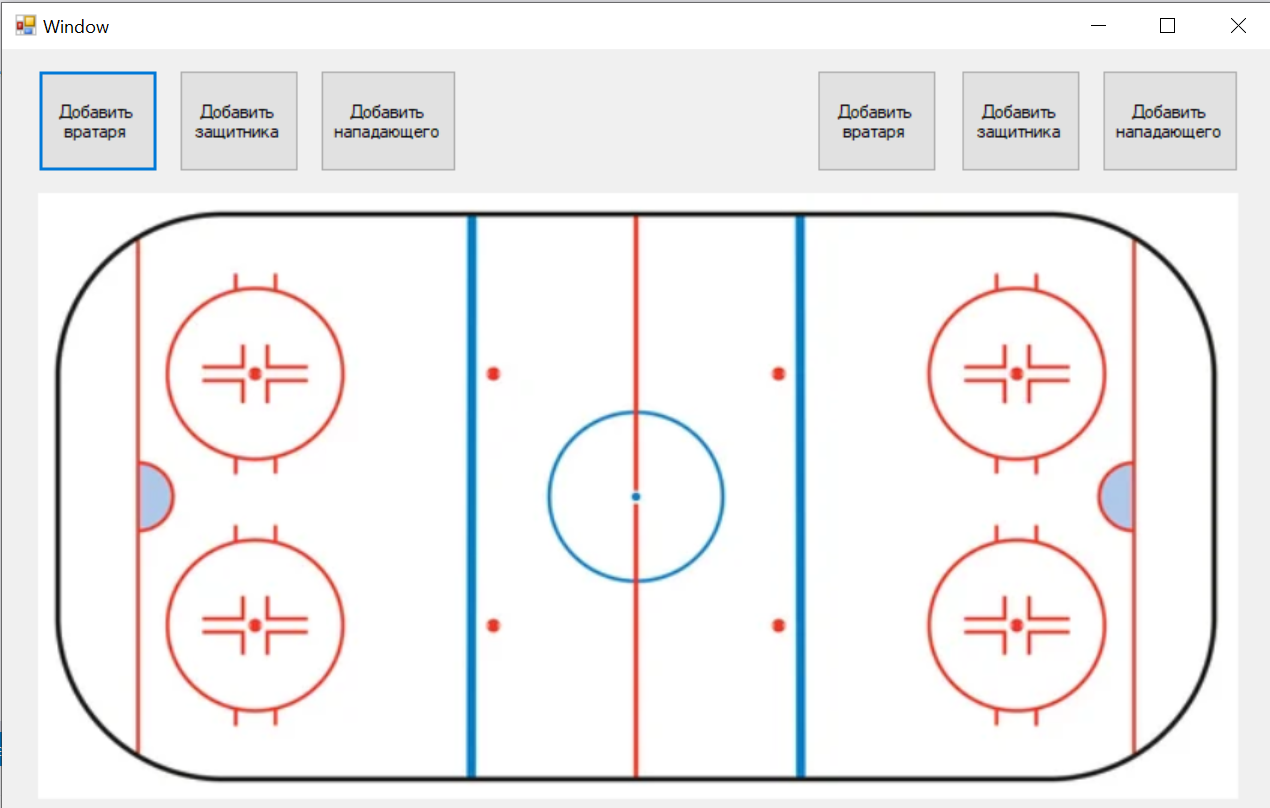


Рисунок 1 – Исходный вил окна

Пользователю доступны следующие действия: добавить вратаря, добавить защитника, добавить нападающего или удалить игрока.

Действия, связанные с добавлением игроков осуществляются после нажатия соответствующей кнопки.

При этом кнопки продублированы для двух сторон поля, то есть пользователь может создать игрока как одной команды, так и другой.

Игроки левой команды обозначены синим цветом, правой – красным.

Типы игроков различаются по форме: вратарь обозначен квадратом, защитник – ромбом, нападающий – треугольником.

Когда пользователь добавляет игрока, он появляется в центре поля. У пользователя есть возможность поставить его в любую точку поля.

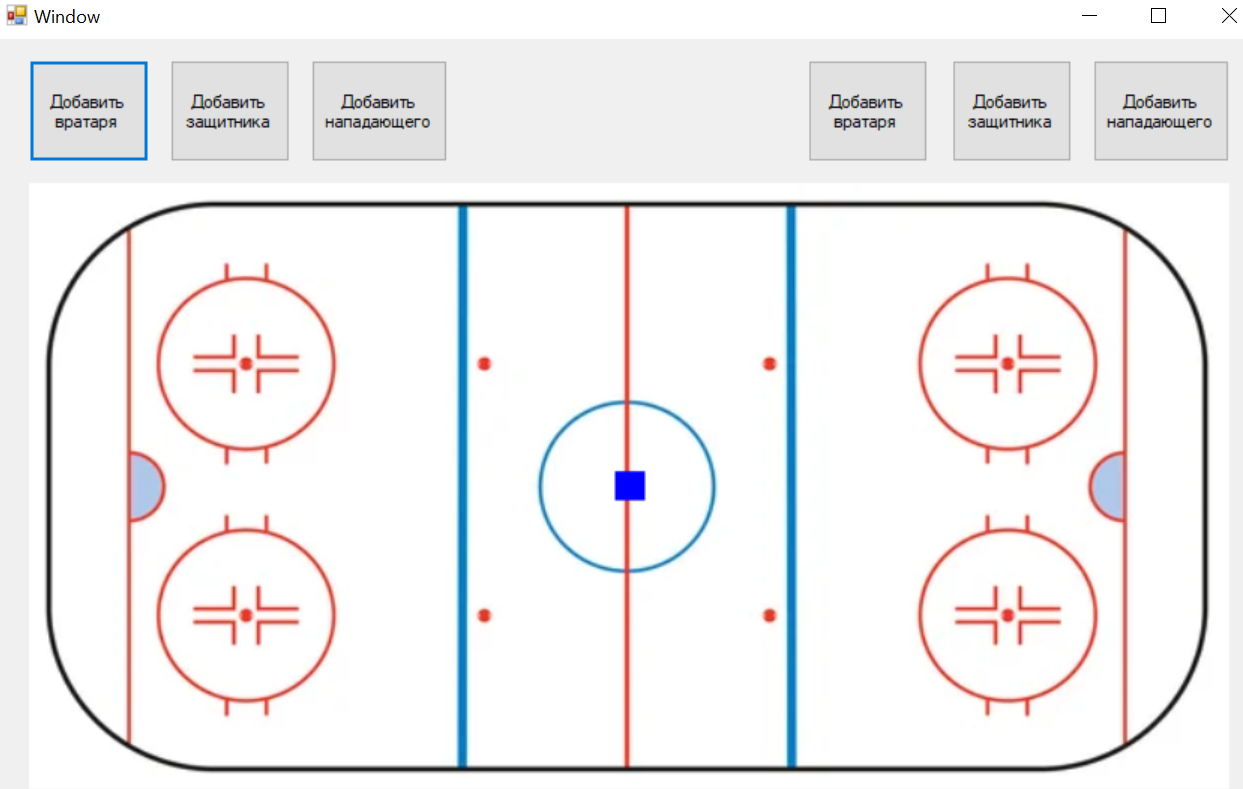


Рисунок 2 – Пример добавления синего вратаря

На рисунке выше приведён пример добавления вратаря левой (синей) команды. Теперь его можно переместить на любую позицию.

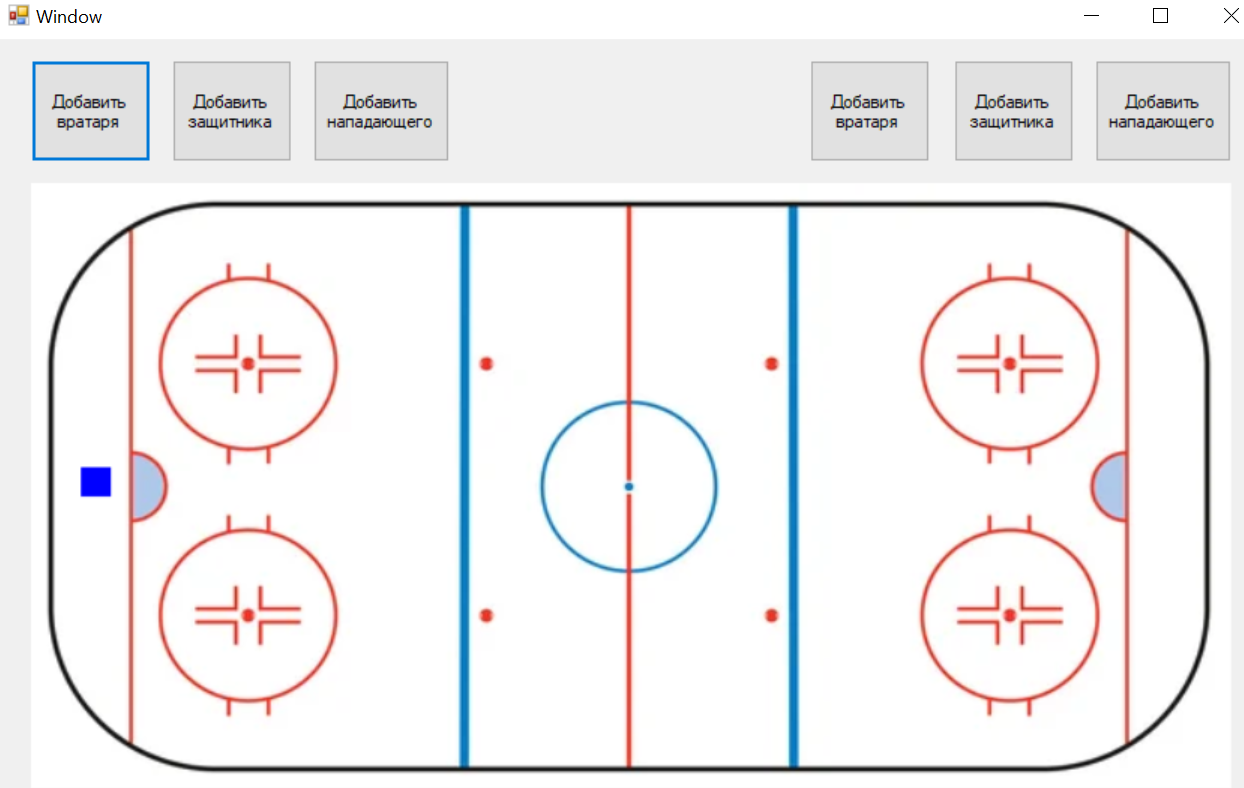


Рисунок 3 – Пример перемещения вратаря

Для примера добавим ещё несколько игроков и разместим их на поле.

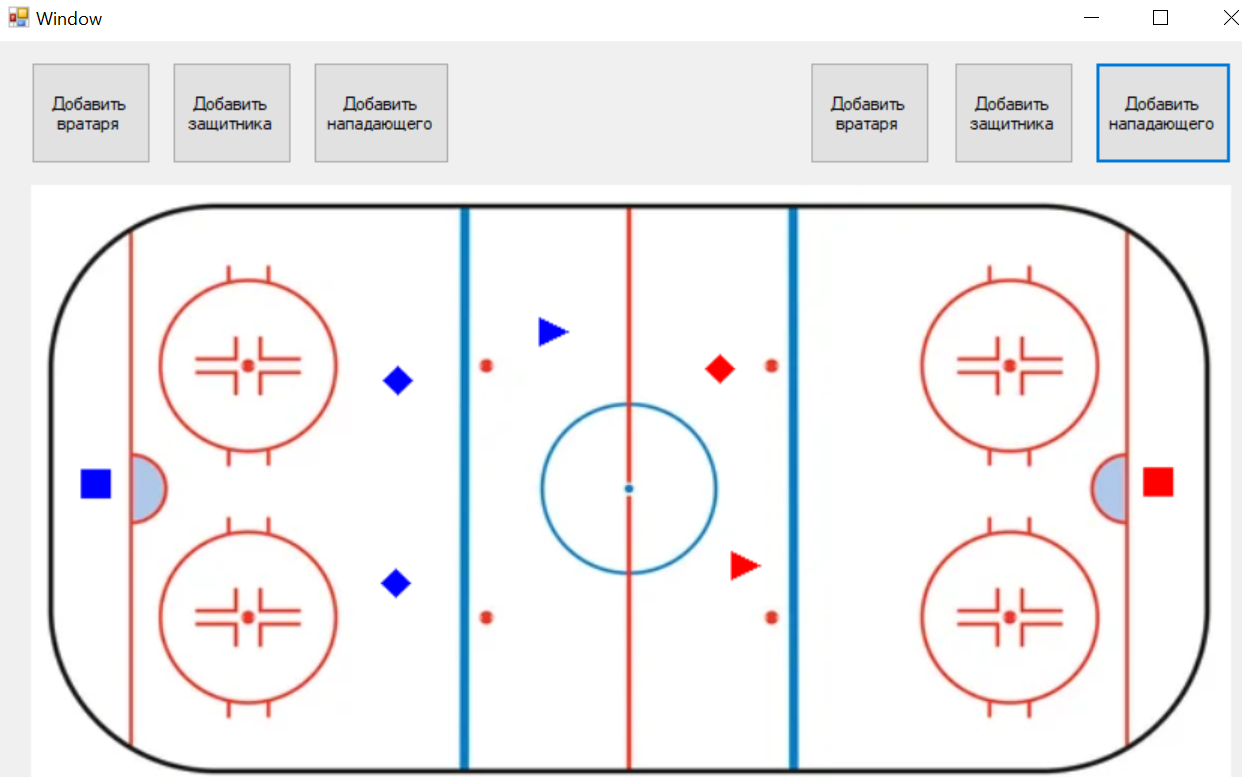


Рисунок 4 – Пример размещения игроков

В каждой хоккейной команде может быть не более 6 человек, причём среди них 1 вратарь, 2 защитника, 3 нападающих.

При попытке добавить лишнего игрока программа выводит сообщение об ошибке.

Попробуем добавить ещё одного красного вратаря.

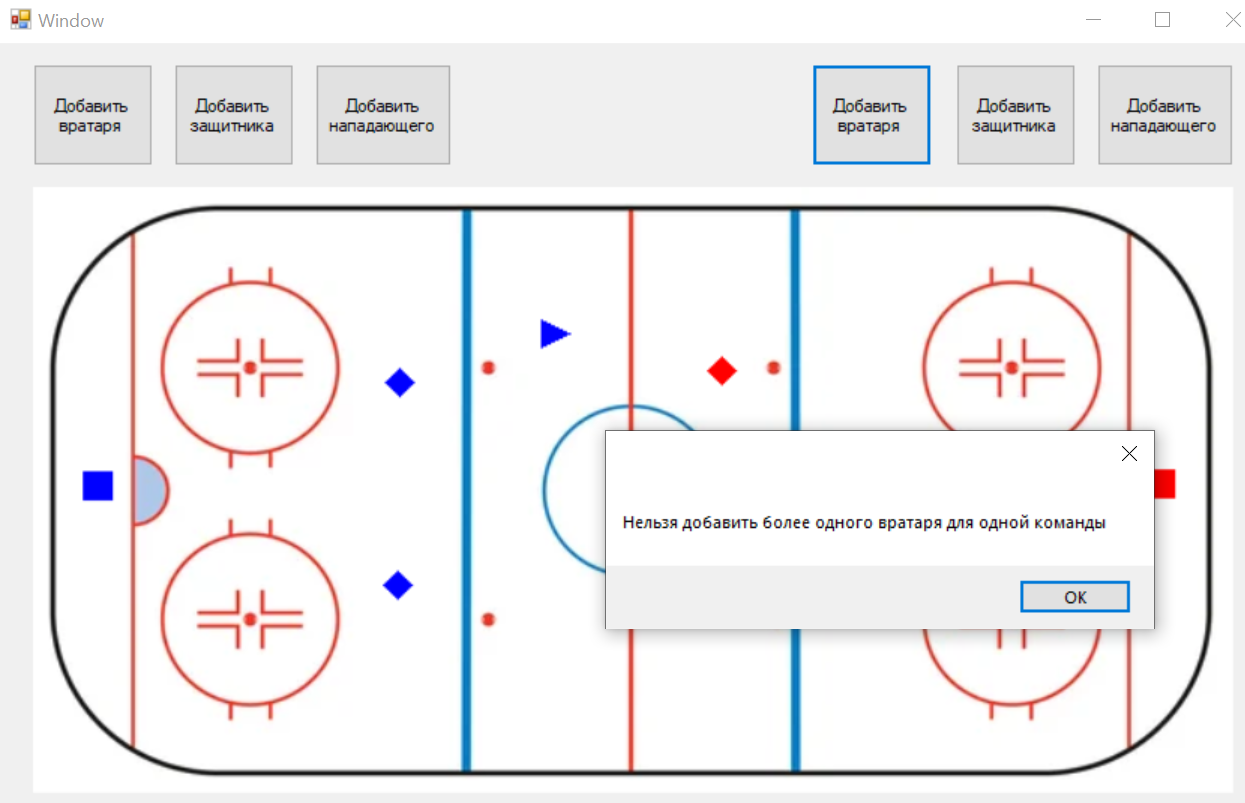


Рисунок 5 – Попытка добавить лишнего игрока

Действительно, это действие невозможно.

Полностью собранные две команды выглядят так:

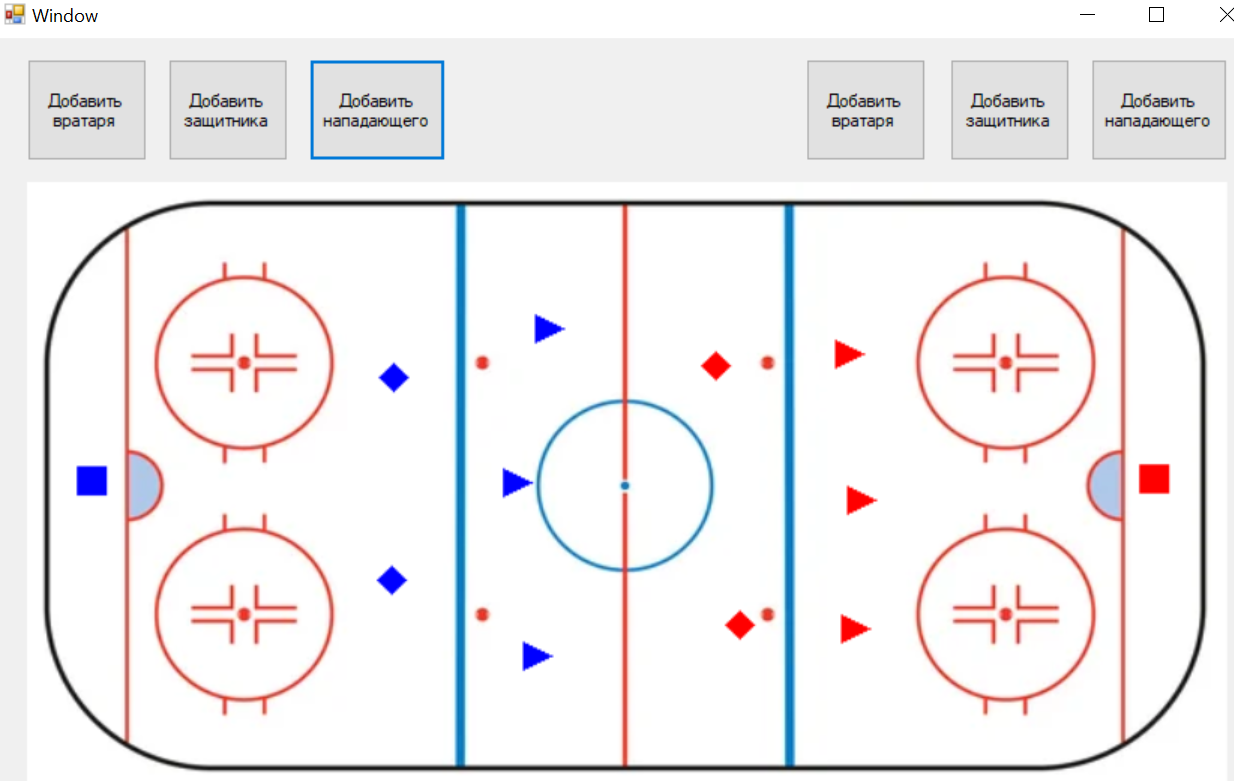


Рисунок 6 – Полностью собранные команды

Для удаления игрока с поля, пользователю необходимо нажать на соответствующего игрока правой кнопкой мыши.

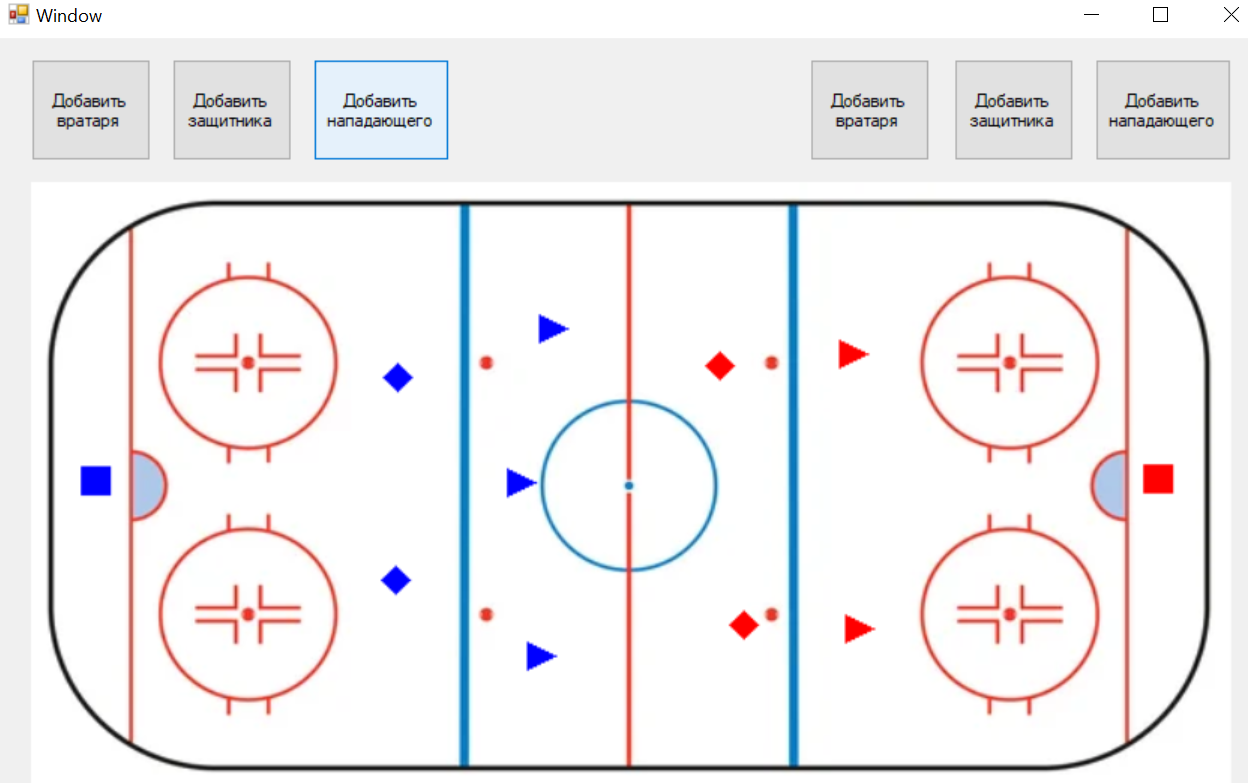


Рисунок 7 – Пример удаления игрока

На рисунке удалили одного красного нападающего. Пользователь может добавить его снова:

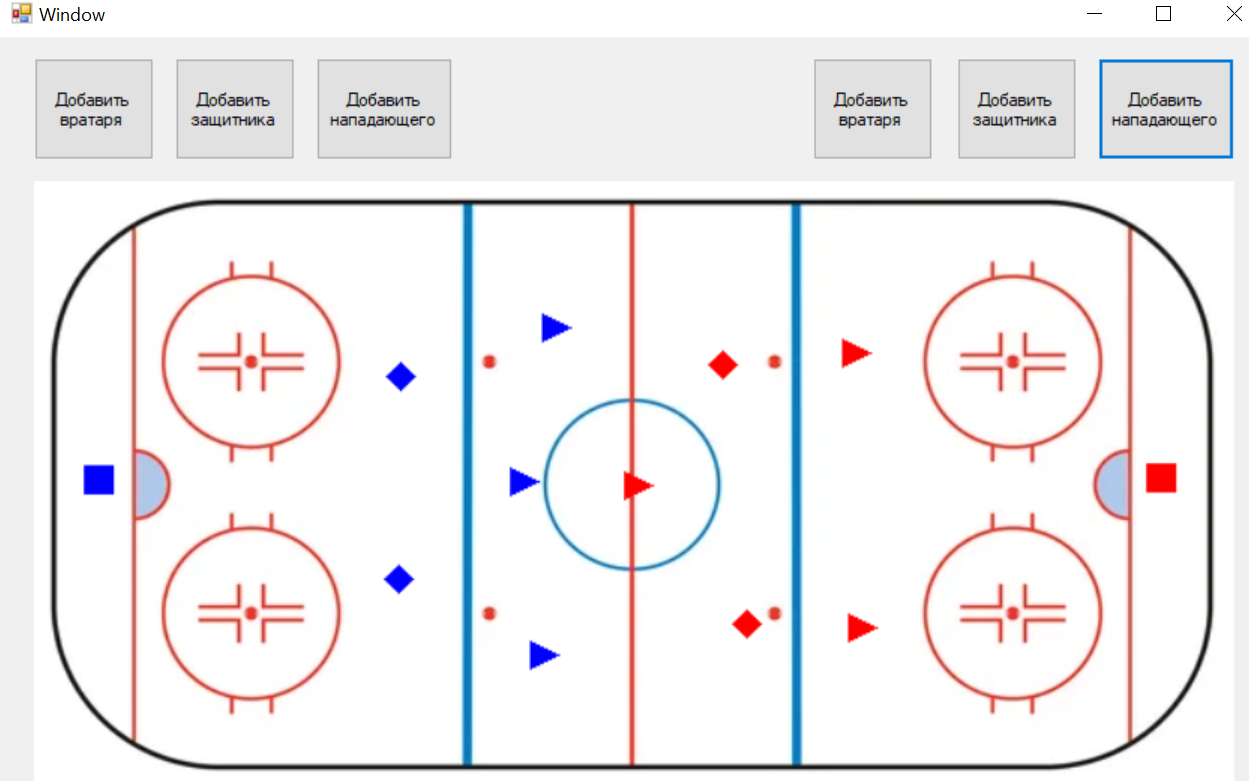


Рисунок 8 – Добавление удалённого игрока

Если поле пустое, или пользователь для удаления нажимает мимо игрока (то есть координаты нажатия не соответствуют координатам какого-либо игрока), выводится сообщение об ошибке.

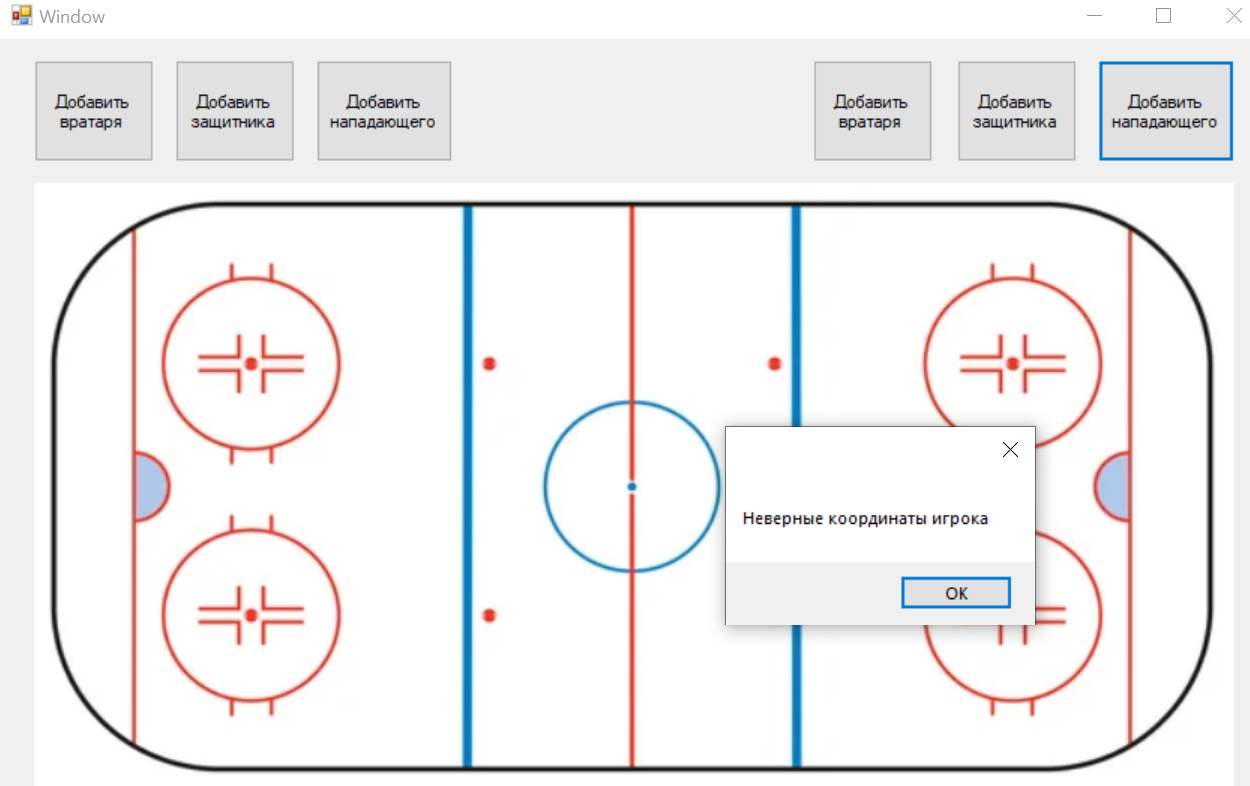


Рисунок 9 – Попытка некорректного удаления игрока

**Выводы**

Закреплены навыки использования механизмов ООП на примере реализации графического редактора.

**Ответы на контрольные вопросы**

1. Зачем нужен перегруженный оператор присваивания?

Чтобы была возможность объекту класса присвоить значение другого объекта этого же класса.

1. Зачем нужен механизм наследования?

Чтобы была возможность использовать свойства и методы объекта-предка в объектах-наследниках без их дублирования.

3. Зачем используются модификаторы при наследовании классов? Какие это модификаторы?

Модификаторы при наследовании классов используются, чтобы определить уровень доступности элементов базового класса для классов-наследников.

Существует три модификатора доступа: public, protected, private.

При использовании модификатора public действуют следующие правила:

* protected и public-члены базового класса доступны из методов унаследованного класса;
* private-члены базового класса недоступны из методов унаследованного класса;
* public-члены базового класса доступны из экземпляров унаследованного класса;
* private и protected члены базового класса недоступны из экземпляров унаследованного класса.

При использовании модификатора protected:

* все protected и public-элементы базового класса доступны из методов унаследованного класса;
* все protected и public-элементы базового класса доступны из методов класса, унаследованного от другого класса-наследника;
* нет доступа к любому элементу базового класса из экземпляров производных классов.

При использовании модификатора private:

* все protected и public-элементы базового класса видимы в классе-наследнике;
* все private-элементы базового класса недоступны в классе-наследнике;
* все классы, унаследованные от класса-наследника, не имеют доступа к элементам базового класса;
* из экземпляра базового класса доступны только public-элементы (члены данных, методы);
* из экземпляра класса-наследника не доступны элементы базового класса.

4. Зачем нужен механизм полиморфизма?

Для того, чтобы реализовать одни и те же методы (с одинаковыми именами) различным способом для разных классов (с учётом особенностей этих классов).

5. Что понимается под динамическим полиморфизмом?

Это полиморфизм во время выполнения программы (в отличие от статического полиморфизма — полиморфизма во время компиляции).

То есть код вызова нужной функции формируется только при выполнении программы.

6. Что такое интерфейс класса?

Интерфейс — это класс, который не имеет переменных-членов, и все методы которого являются виртуальными функциями.

7. Зачем нужен чисто виртуальный метод? Как он выглядит?

Чисто виртуальный метод даёт гарантию, что он будет переопределён в каком-либо из наследников данного класса (иначе код, в котором выполняется попытка создания объекта такого класса, не скомпилируется).

Такие методы объявляются с использованием конструкции «= 0».

(Например: virtual int count\_parameters() = 0;)

8. Какой класс называется абстрактным?

Класс, содержащий хотя бы один чисто виртуальный метод.

Абстрактный класс определяет интерфейс для переопределения производными классами. Он может использоваться только в качестве базового для других классов.

Объекты абстрактного класса создавать нельзя, так как вызов чисто виртуального метода приводит к ошибке при выполнении.