МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение

высшего образования

**«Северный (Арктический) федеральный университет имени М.В. Ломоносова»**

Высшая школа информационных технологий и автоматизированных систем

(наименование высшей школы / филиала / института / колледжа)

привет от автора)

**РАСЧЕТНО-ГРАФИЧЕСКАЯ РАБОТА**

|  |  |  |
| --- | --- | --- |
| По дисциплине | | Технология разработки программного обеспечения |
|  | |  |
|  | |  |
| По теме | Разработка скринмейта | |
|  | | |

|  |
| --- |
| Выполнил обучающийся:  Макаров Артем Александрович |
| (Ф.И.О.) |
| Направление подготовки:  09.03.02 Информационные системы и технологии |
| (код и наименование) |
| Курс: 3 |
| Группа: 351818 |
| Руководитель:  Маломан Ю.С., ст. преподаватель |
| (Ф.И.О. руководителя, должность / уч. степень / звание) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Отметка о зачете |  |  |  |  |
|  |  | (отметка прописью) |  | (дата) |
| Руководитель |  |  |  | Ю.С. Маломан |
|  |  | (подпись руководителя) |  | (инициалы, фамилия) |

Архангельск 2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ федеральное государственное автономное образовательное учреждение высшего образования   
**«Северный (Арктический) федеральный университет имени М.В. Ломоносова»**

Высшая школа информационных технологий и автоматизированных систем

**ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ РАСЧЕТНО-ГРАФИЧЕСКОЙ РАБОТЫ**

**По дисциплине Технология разработки программного обеспечения**

Студенту высшей школы информационных технологий и автоматизированных систем 3 курса 351818 группы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Макарову Артему Александровичу \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О. студента)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_09.03.02 Информационные системы и технологии\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(код и наименование направления подготовки)

**На тему: Разработка скринмета**

**Задание**

1. По выбранной теме разработать приложение, провести комментирование, форматирование и рефакторинг кода
2. Оформить пояснительную записку по стандарту СТО.

**Структура пояснительной записки**Титульный лист   
Задание   
Лист для замечаний   
СодержаниеВведение

Сбор и анализ требований

Проектирование

Разработка

Тестирование и отладка

Документирование

Заключение

Приложение (при необходимости)

**Срок выполнения:** с 1 ноября 20 20 г. по 1 декабря 20 20 г.

**Руководитель работы** ст.преподаватель Ю.С. Маломан

Архангельск 2020

## лист замечаний

## содержание

[лист замечаний 2](#_Toc57584684)

[содержание 4](#_Toc57584685)

[Введение 5](#_Toc57584686)

[1 Сбор и анализ требований 6](#_Toc57584687)

[2 проектирование 7](#_Toc57584688)

[3 Разработка 8](#_Toc57584689)

[3.1 Инициализация переменных и конструктора 8](#_Toc57584690)

[3.2 Методы режимов 10](#_Toc57584691)

[3.3 Вспомогательные методы 11](#_Toc57584692)

[3.4 Метод смены режима 13](#_Toc57584693)

[4 Тестирование и отладка 14](#_Toc57584694)

[приложение а (обязательное) Листинг режима мяча 16](#_Toc57584695)

[приложение Б (обязательное) Листинг режима преследователя 17](#_Toc57584696)

[приложение В (обязательное) Листинг режима воронки 18](#_Toc57584697)

[приложение Г (обязательное) Листинг метода смены режима 19](#_Toc57584698)

## Введение

В настоящее время человечество переживает не лучшие времена: пандемия, переход на дистанционную работу, документы, отчеты… Люди постепенно выгорают, перестают получать эмоции, становятся грустными, ленивыми, Перестают видеть что-либо вокруг себя.

Но даже в такой ситуации встречаются люди, которые другим людям не нравятся по той или иной причине. И таким людям хочется навредить, напакостить. Или же наоборот, в компании друзей и в качестве отвлечения от повседневности хочется воспользоваться каким-нибудь интересным приспособлением для расслабления и отвлечения от забот.

Поскольку темой данной работы выступает разработка скринмейта (от англ. screen mate – экранный товарищ), возникла идея объединить необычное с зловредным, для веселья и отдыха и создать компьютерного «вредителя».

# 1 Сбор и анализ требований

Данное решение будет предназначено для людей, кто хотел бы каким-либо образом некритично навредить другим людям в компьютере. Таким образом, продукт будет предназначен для офисных работников любого уровня, студентов, школьников.

Поскольку требуется разработать решение, некритично влияющее на систему, взаимодействие будет ограничено исключительно влиянием на курсор пользователя.

Также требуется при первой необходимости закрыть решение, поэтому блокирования системных комбинаций клавиш произведено не будет.

Поскольку большинство рабочих мест основано на ОС Windows, приложение должно поддерживаться данной системой.

В завершении это должно быть не тяжелое приложение и не сильно ресурсоёмким, поскольку экранный товарищ – дополнение к системе, а не основное рабочее приложение.

Популярных решений подобного плана в мире не существует, поскольку приложения типа screenmate не распространены среди пользователей.

Однако существует решение, именуемое «Компьютерным гусем», у которого такая же цель, что и в обозначенном приложении. Но данное решение считается вирусным приложением, затрагивающее некоторые аспекты hardware. Решение, описанное в данной работе, лишено данного недостатка.

# 2 проектирование

В качестве языка программирования для реализации был выбран C# со средой разработки Visual Studio по причине гораздо более простой работы с API операционной системы Windows.

В качестве системы для построения приложений была выбрана WPF в составе .NET Framework 4.0 для упрощения создания и работы с визуальной составляющей.

Приложение распределяется на два компонента: визуальный и контролирующий. Визуальный элемент представляет из себя XAML-файл, в которую через контролирующий элемент передаются изображения, размеры окна и устанавливаются параметры окна, свойственные для скринмейта, а именно прозрачное окно и небольшие размеры. Контролирующий элемент содержит в себе общую логику приложения и передает значения величин во второй компонент.

Для воздействия на курсор будет реализовано три режима.

Первый режим – режим мяча. Для его реализации потребуется использование имитации физики в двумерном пространстве. Мяч будет перемещаться в сторону курсора и пытаться его коснуться, попутно притягиваться низом экрана к себе. При столкновении курсора с мячом – курсор отскакивает в сторону от центра мяча по вектору. Текстура режима – баскетбольный мяч.

Второй режим – режим преследователя. Для его реализации потребуется механизм перемещения по нормализованному вектору в сторону курсора. В случае соприкосновения курсора с преследователем – курсор отскакивает в сторону от центра. Текстура режима – голова.

Третий режим – режим воронки. Для его реализации потребуется использование механизма перемещения курсора к центру воронки по нормализованному вектору. Текстура режима – черная дыра.

Все три режима заменяют друг друга случайным образом через 25 секунд. Во время смены текстура вращается вокруг собственного центра. После смены режима меняется текстура скринмейта.

После старта скринмейт появляется на втором режиме с левого верхнего угла.

Взаимодействовать со скринмейтом возможно только попыткой перемещения самого персонажа зажатием кнопкой мыши. Если перемещение происходит во время смены режима – смена режима не будет произведена до момента отпуска клавиши мыши.

# 3 Разработка

3.1 Инициализация переменных и конструктора

Первым делом требуется задать параметры основного окна. Окно будет иметь размеры 100 на 67 пикселей (по размеру второй текстуры), имеет прозрачный фон, должно находиться поверх всех окон, иметь максимальный размер такой же какой имеет сейчас, а так же при запуске не быть расширяемым. Внутри окна должна располагаться только картинка. Для дальнейших действий по вращению и изменению требуется учесть размещение атрибутов ScaleTransform и RotateTransform. Код представлен в листинге 1.

Листинг 1 – Код XAML файла

<Window x:Class="ScreenMate.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:ScreenMate"

xmlns:tb="http://www.hardcodet.net/taskbar"

mc:Ignorable="d"

x:Name="Mate"

Title="MainWindow" Height="100" Width="67"

AllowsTransparency="True"

WindowStyle="None"

Background="#00000000"

Topmost="True"

MaxHeight="100" MaxWidth="67"

WindowState="Normal"

>

<Grid>

<Image x:Name="img" Source="image\benis.png" >

<Image.RenderTransform>

<TransformGroup>

<ScaleTransform x:Name="scale" ScaleX="1"></ScaleTransform>

<RotateTransform x:Name="rotate" CenterX="0" CenterY="0" Angle="0" ></RotateTransform>

</TransformGroup>

</Image.RenderTransform>

</Image>

<!--<tb:TaskbarIcon x:Name="toolbarTray" Visibility="Visible" ToolTipText="Screen Fucking Mate"/>-->

</Grid>

</Window>

Перейдем к разработке самой логики скринмейта. Первым делом требуется создать переменные, используемые далее в приложении, а именно: ускорение свободного падения, стартовое время в воздухе, позиции расположения окна и курсора, высоту и ширину окна, разницу для триггера на курсор, переменные для перемещения по осям X и Y окна и курсора, проведение кика, коэффициент перемещения, таймеры для основной логики и смены режима, переменная режима и спрайты. Данный код приведен в листинге 2.

Листинг 2 – Переменные программы

//Ускорение свободного падения

protected double a = 0.5;

//Время в воздухе

protected double flyingTime = 1;

//Точка расположения животного

protected Point location;

//Точка расположения курсора

protected Point cursorLocation;

//Высота животного

public int height = 100;

//Ширина животного

public int width = 67;

//Поле, в котором происходит триггер курсора

double deltaMouse = 25;

//Переменные перемещения позиции окна за один такт

double moveX, moveY;

double moveMouseX, moveMouseY;

bool kick = false;

//Коэффициент перемещения (для определения, вверх или вниз перемещается персонаж)

int koeff;

//Инициализация таймеров

System.Windows.Threading.DispatcherTimer timer = new System.Windows.Threading.DispatcherTimer();

System.Windows.Threading.DispatcherTimer timerChangeMode = new System.Windows.Threading.DispatcherTimer();

//Режим зверька

int mateMode = 0;

//Инициализация спрайтовв

BitmapImage ball = new BitmapImage(new Uri("image/ball.png", UriKind.Relative));

BitmapImage benis = new BitmapImage(new Uri("image/benis.png", UriKind.Relative));

BitmapImage vortex = new BitmapImage(new Uri("image/vortex.png", UriKind.Relative));

Следующим пунктом станет инициализация всех параметров приложения в конструкторе окна, которое вызывается во время запуска приложения. В конструкторе мы задаем размеры окна, максимальные размеры окна, стартовую текстуру, настраиваем таймеры и проводим подписку на событие завершения времени таймеров (для первого таймера – метод определенного режима, каждые 0.01 секунду, для второго – метод проведения замены режима, каждые 25 секунд). Присвоение переменных происходит по именам определенных пунктов XAML файла, в котором ранее мы указали имена тех или иных атрибутов. Код конструктора представлен на листинге 3.

Листинг 3 – Конструктор приложения

public MainWindow()

{

InitializeComponent();

//Установка размеров окна

Mate.Height = height + 20;

Mate.Width = width + 20;

Mate.MaxHeight = height + 20;

Mate.MaxWidth = width + 20;

//Стартовый зверь

img.Source = benis;

//Установка интервала действия персонажа

timer.Interval = TimeSpan.FromSeconds(0.01);

timer.Start();

timer.Tick += TickerTimerBenis;

timer.Tick += ControlWindowSize;

//Таймер на смену режима

timerChangeMode.Interval = TimeSpan.FromSeconds(25);

timerChangeMode.Start();

timerChangeMode.Tick += TimerChangeMode;

}

3.2 Методы режимов

Далее требуется создать по методу, контролирующему каждый из трех режимов.

Первый метод будет обрабатывать режим мяча. Данный метод разработан в соответствии с планом реализации. Метод Kicking будет описан в разделе 3.3. Код метода приведен в приложении А.

Стоит обратить внимание на следующие параметры:

- moveX и moveY расчитываются из положения курсора и положения персонажа с поправкой на размер окна;

- Left и Top представляют из себя расстояния от левого и верхнего края экрана, при этом значение Left изменяется в зависимости от положения мыши с помощью проекции нормализованного вектора на ось X, помноженного на 5, а значение Top изменяется по формуле равноускоренного движения, в зависимости от коэффициента полета и времени полета.

Второй метод будет обрабатывать режим преследователя. Данный метод разработан в соответствии с планом реализации. Код приведен в приложении Б.

Следует обратить внимание на корневую механику, а именно перемещение происходит по пятикратно увеличенному нормализованному вектору от позиции скринмейта до позиции курсора (изменяются параметры Left и Top).

Третий метод будет обрабатывать режим воронки. Данный метод разработан в соответствии с планом реализации. Метод MoveMouse будет описан в разделе 3.3. Код приведен в приложении В.

Следует обратить внимание на механику перемещения курсора к воронке: она идентична механике преследователя, только теперь курсор перемещается к объекту, а не объект к курсору.

3.3 Вспомогательные методы

В качестве вспомогательных методов выступают методы перемещений, обработка событий и контроллер расширения экрана.

Первый вспомогательный метод – метод перемещения курсора на определенное расстояние от нынешнего местоположения. Данный метод имеет обработчик исключений: с помощью тестирования было выявлено, что если курсор должен переместиться на бесконечно малое расстояние – появляется исключение переполнение ячейки объекта. Поэтому в случае появления исключения метод насильно вытолкнет курсор на 100 пикселей влево и вверх. Код метода представлен в листинге 4.

Листинг 4 – Метод перемещения курсора мыши

private void MoveMouse(double dx, double dy)

{

try

{

System.Windows.Forms.Cursor.Position = new System.Drawing.Point(Convert.ToInt32(System.Windows.Forms.Cursor.Position.X + dx), Convert.ToInt32(System.Windows.Forms.Cursor.Position.Y + dy));

}

catch

{

System.Windows.Forms.Cursor.Position = new System.Drawing.Point(Convert.ToInt32(System.Windows.Forms.Cursor.Position.X - 100), Convert.ToInt32(System.Windows.Forms.Cursor.Position.Y - 100));

}

}

Второй метод – метод пинка, представляющий из себя последовательное перемещение мыши на определенное расстояние в зависимости от времени полета. Когда происходит столкновение с скринмейтом в первом или втором режимах, курсор должен отлетать. Для этого используется ранее описанный метод MoveMouse, но расстояние, на которое требуется переместить курсор постоянно уменьшается в 2 раза, пока расстояние не будет меньше либо равно единице, для остановки выполнения перемещения. Данный метод показан в листинге 5.

Листинг 5 – Метод перемещения курсора во время столкновения

private void Kicking()

{

if (kick)

{

MoveMouse(moveMouseX, moveMouseY);

moveMouseX /= 2;

moveMouseY /= 2;

if (moveMouseX <= 1 && moveMouseY <= 1)

{

kick = false;

}

}

if ((location.X + width / 2 + deltaMouse > cursorLocation.X) && (location.X + width / 2 - deltaMouse - 35 < cursorLocation.X) && (location.Y + height / 2 - 20 + deltaMouse + 10 > cursorLocation.Y) && (location.Y + height / 2 - 50 - deltaMouse < cursorLocation.Y))

{

kick = true;

moveMouseX = 105 \* (moveX) / Math.Sqrt(Math.Pow(moveX, 2) + Math.Pow(moveY, 2));

moveMouseY = 105 \* (moveY) / Math.Sqrt(Math.Pow(moveX, 2) + Math.Pow(moveY, 2));

}

}

Третий метод – метод перемещения скринмейта при помощи зажатой левой клавиши мыши. Код приведен в листинге 6.

Листинг 6 – Метод перемещения окна зажатой клавишей мыши

protected override void OnMouseLeftButtonDown(MouseButtonEventArgs e)

{

base.OnMouseLeftButtonDown(e);

this.DragMove();

}

И последний метод – метод контроля запрета расширения и сворачивания окна. При помощи тестирования выявлена ошибка, при которой пользователь может убрать или расширить приложение. Данный метод, привязываемый к основному таймеру, позволяет следить изменение расширения окна и предотвращает это действие. Код представлен в листинге 7.

Листинг 7 – Метод запрета расширения окна

private void ControlWindowSize(object sender, EventArgs e)

{

//Контроль расширения (для восстановления)

if (Mate.WindowState != WindowState.Normal)

{

Mate.WindowState = WindowState.Normal;

}

}

3.4 Метод смены режима

Метод смены режима – третий тип методов, подвязываемый к отдельному таймеру с временем реализации 25 секунд. Данный метод призван сменить режим скринмейта, параллельно вращая его.

Для смены режима для начала происходит отписка всех методов от первого таймера и изменение времени тика второго таймера, если изображение скринмейта было повернуто – происходит обратный поворот, после чего происходит начало вращения изображения. Как только угол поворота будет больше 120 – произойдет смены текстуры и подписываемого метода к основному таймеру.

Данный метод описан в приложении Г.

# 4 Тестирование и отладка

Тестирование проходило двумя методами: методом черного ящика и методом белого ящика.

Для тестирования методом черного ящика приложение в исполняемой форме было разослано коллегам с просьбой протестировать, на что было получено определенное количество найденных ошибок, которые в той или иной степени были исправлены.

Для тестирования методом белого ящика выполнялись функции, которые были расписаны в проектировании, что так же нашло несколько ошибок, которые были исправлены впоследствии.

Все тесты, которые проводились с данным приложением, показаны в таблице 1.

Таблица 1 – Тестирование приложения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № теста | Найденная ошибка | Тип теста | Действия | Статус |
| 1 | Расширение скринмейта | «Черный ящик» | 1) Поймать скринмейт  2) Перетянуть на самый верх экрана | Исправлено: добавлено ограничение на максимальный размер |
| 2 | Вылет при невозможности перемещения курсора | «Белый ящик» | 1) Дождаться активации режима воронки  2) Заставить её засосать курсор в самый центр, чтобы отсутствовали колебания курсора | Исправлено: добавлена обработка исключения перемещения |
| 3 | При попытке расширения происходит иммобилизация скринмейта | «Черный ящик» | Попытаться реализовать действия 1 теста | Исправлено: добавлен запрет на расширение |
| 4 | Режим клонирования не функционирует | «Белый ящик» | Попытаться запустить клонирование персонажа | Исправлено: функционал вырезан из-за большой нагрузки на систему |
| 5 | Во время перемещения скринмейта курсор продолжает отталкиваться | «Черный ящик» | 1) Поймать скринмейт  2) Попытаться переместить | Исправлено: поставлен блок на отталкивание если замещен захват персонажа |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 | Во время захвата скринмейта после смены режима приложение начинает глючить | «Черный ящик» | 1) Поймать скринмейт во время смены режима  2) Удерживать захват до окончания превращения | Исправлено: поставлен блок на превращение до окончания захвата |
| 7 | Во время смены режима после режима преследователя некорректно поставлен центр вращения | «Белый ящик» | 1) Дождаться режим преследователя  2) Заставить начать менять режим во время поворота преследователя | Исправлено: добавлено принудительное возвращение изображения в исходную позицию |
| 8 | При попытке потрясти воронку приложение вылетает | «Черный ящик» | 1) Дождаться режим воронки  2) Захватить воронку  3) Начать быстро перетаскивать | Не исправлено: при попытке воспроизвести данную ошибку ошибка не была обнаружена. Попытки были выполнены на 6 разных устройствах, включая устройство тестировщика |

# 5 документирование

5.1 Документирование кода

Код задокументирован в соответствии со стандартами документирования кода на языке C# на платформе Visual Studio[1]. Полный текст программы можно посмотреть на сайте GitHub[2] и BitBucket[3].

5.2 Документация для пользователя

Для конечного пользователя не требуется как таковая техническая документация из-за отсутствия прямого взаимодействия пользователя с приложением.

Для запуска приложения требуется запустить ScreenMate.exe.

Для перетаскивания приложения требуется зажать изображение левой кнопкой мыши и перетащить.

Для завершения приложения требуется нажать комбинацию клавиш Alt+F4 или закрыть приложение через панель приложений.

## Заключение

Выполняя задание, были улучшены навыки программирования и создано забавное приложение, способное поднять настроение или разозлить кого-угодно.

Приложение работоспособное и функционирует в соответствии с проектированием. Все требования выполнены.

## Список использованых источников

1 Документирование кода с помощью XML-комментариев [Электронный ресурс] : [Офиц сайт] / Mic. Docs – Электрон. дан. – [Редмонд] : Microsoft, 1975–2020. – Режим доступа : https://docs.microsoft.com/ru-ru/dotnet/csharp/codedoc, свободный (дата обращения : 29.11.2020). – Загл. с экрана.

2 theneoxs/screenMate [Электронный ресурс] / theneoxs // GitHub – Электрон. дан. – [Редмонд] : Microsoft, 2008–2020. – Режим доступа : https://github.com/theneoxs/screenMate, свободный (дата обращения : 29.11.2020). – Загл. с экрана.

3 РГР-351818-9 [Электронный ресурс] / Артем Макаров // BitBucket – Электрон. дан. – [Сидней] : Atlassian, 2008–2020. – Режим доступа : https://bitbucket.org/theneoxs/rgr-351818-9/src/master/, свободный (дата обращения : 29.11.2020). – Загл. с экрана.

## приложение а (обязательное) Листинг режима мяча

private void TickTimerBall(object sender, EventArgs e)

{

location = PointToScreen(new Point(0, 0));

cursorLocation = new Point(System.Windows.Forms.Control.MousePosition.X - 22, System.Windows.Forms.Control.MousePosition.Y - 38);

//Вектор от мяча до курсора

moveX = cursorLocation.X - (location.X + width / 2 - 14);

moveY = cursorLocation.Y - (location.Y + height / 2 - 25);

//Коэффициент направления перемещения

koeff = 1;

//Если левая клавиша не зажата (т.е. животное не перетаскивается)

if (Mouse.LeftButton != MouseButtonState.Pressed)

{

Kicking();

Left += 5 \* (moveX) / Math.Sqrt(Math.Pow(moveX, 2) + Math.Pow(moveY, 2));

//Проверка, в воздухе ли животное

if (Top < System.Windows.SystemParameters.PrimaryScreenHeight - 140)

{

//Здесь он взлетает или падает, в зависимости от flyingTime

if (flyingTime < 0)

{

koeff = -1;

}

else

{

koeff = 1;

}

Top += koeff \* a \* Math.Pow(flyingTime, 2) / 2;

flyingTime += 0.2;

}

else

{

//Здесь он прыгает как только касается низа (получает импульс)

Top = System.Windows.SystemParameters.PrimaryScreenHeight - 140;

flyingTime = -11;

koeff = -1;

Top += koeff \* a \* Math.Pow(flyingTime, 2) / 2;

}

}

else

{

flyingTime = 1;

}}

## приложение Б (обязательное) Листинг режима преследователя

private void TickerTimerBenis(object sender, EventArgs e)

{

location = PointToScreen(new Point(0, 0));

cursorLocation = new Point(System.Windows.Forms.Control.MousePosition.X - 22, System.Windows.Forms.Control.MousePosition.Y - 38);

//Вектор от бениса до курсора

moveX = cursorLocation.X - (location.X + width / 2 - 14);

moveY = cursorLocation.Y - (location.Y + height / 2 - 25);

if (Mouse.LeftButton != MouseButtonState.Pressed)

{

//Перемещение бениса к курсору

Top += 5 \* (moveY) / Math.Sqrt(Math.Pow(moveX, 2) + Math.Pow(moveY, 2));

Left += 5 \* (moveX) / Math.Sqrt(Math.Pow(moveX, 2) + Math.Pow(moveY, 2));

//Логика поворота

if (moveX < 0)

{

scale.CenterX = width / 2;

scale.ScaleX = -1;

}

else

{

scale.CenterX = 0;

scale.ScaleX = 1;

}

//Отталкивание курсора

Kicking();

}

}

## приложение В (обязательное) Листинг режима воронки

private void TickerTimerVortex(object sender, EventArgs e)

{

location = PointToScreen(new Point(0, 0));

cursorLocation = new Point(System.Windows.Forms.Control.MousePosition.X - 22, System.Windows.Forms.Control.MousePosition.Y - 38);

//Вектор от воронки до курсора

moveX = cursorLocation.X - (location.X + width / 2 - 14);

moveY = cursorLocation.Y - (location.Y + height / 2 - 25);

rotate.Angle += 5;

//Перемещение курсора к воронке

MoveMouse(-5 \* (moveX) / Math.Sqrt(Math.Pow(moveX, 2) + Math.Pow(moveY, 2)), -5 \* (moveY) / Math.Sqrt(Math.Pow(moveX, 2) + Math.Pow(moveY, 2)));

}

## приложение Г (обязательное) Листинг метода смены режима

private void TimerChangeMode(object sender, EventArgs e)

{

timerChangeMode.Interval = TimeSpan.FromSeconds(0.01);

//Снятие всех режимов

timer.Tick -= TickTimerBall;

timer.Tick -= TickerTimerBenis;

timer.Tick -= TickerTimerVortex;

scale.ScaleX = 1;

//Смещение центра (для вращения)

rotate.CenterX = (width + 22) / 2;

rotate.CenterY = (height) / 2;

if (Mouse.LeftButton != MouseButtonState.Pressed)

{

//Если прошло достаточно времени поворота - смена режима

if (flyingTime >= 120)

{

timerChangeMode.Interval = TimeSpan.FromSeconds(25);

flyingTime = 0;

rotate.Angle = 0;

//Выбор режима

mateMode = new Random().Next(0, 3);

if (mateMode == 0)

{

timer.Tick += TickerTimerVortex;

img.Source = vortex;

}

else if (mateMode == 1)

{

timer.Tick += TickTimerBall;

img.Source = ball;

}

else

{

timer.Tick += TickerTimerBenis;

img.Source = benis;

}

}

else

{

flyingTime += 1;

Kicking();

}

}

rotate.Angle += flyingTime;

}