**ИИТ БГУИР**

**Факультет повышения квалификации и переподготовки**   
  
**Дисциплина: Средства визуального программирования приложений**

**Лабораторная работа №4**

Создание пользовательских элементов управления

Вариант 3

**Выполнил слушатель гр. 40322-1** Проверила

**Гончаров Максим Евгеньевич Желакович Ирина Миролюбовна**

Минск 2025

*Цели лабораторной работы*:

Освоить методику создания пользовательских элементов управления. Изучить способы использования делегатов и событий.

*Выполнение работы:*

a) Создать проект типа WPF ControlLibrary.

b) Запрограммировать пользовательский элемент управления.

c) Добавить в элемент нужные свойства и события.

d) Написать приложение типа WPF, использующее созданный элемент. Приложение должно иметь кнопки Старт/Пауза (для запуска и приостановки игры) и Сброс (для возврата игры в начальное состояние), а также поля для вывода информации об объектах игры согласно заданию.

Вариант 3. Разработать пользовательский элемент управления, реализующий игру “Стрельба из подводной лодки по кораблю из перископа”. На заднем плане должен периодически проплывать кораблик с постоянной поперечной скоростью. С помощью кнопок “влево” и “вправо” следует менять вид в перископе (точка прицеливания). Кнопка “Пуск” должна запускать торпеду. В перископе должна отображаться траектория торпеды с уменьшением скорости движения при приближении к кораблю. Попадание должно сопровождаться взрывом и исчезновением корабля. При щелчке левой клавиши мыши по торпеде должна выводиться информация о ее скорости, а при щелчке по кораблю - информация о его положении.

1. В лабораторной работе используется единый класс ShipTorpedoGame, который содержит все свойства и логику:

// Генератор случайных чисел для позиционирования корабля

private Random random = new Random();

// Флаг состояния игры (запущена/на паузе)

private bool isGameRunning = false;

// Координаты подводной лодки (центр перископа)

private double submarineX = 200;

private double submarineY = 300;

// Скорость торпеды (уменьшается со временем)

private double torpedoSpeed = 5;

// Флаг, указывающий запущена ли торпеда

private bool torpedoLaunched = false;

// Координаты торпеды

private double torpedoX, torpedoY;

// Координаты корабля

private double shipX = -100, shipY = 50;

// Скорость движения корабля

private double shipSpeed = 3;

// Флаг видимости корабля (исчезает при попадании)

private bool shipVisible = true;

// Координаты взрыва (в реальных координатах)

private double explosionX, explosionY;

// Флаг видимости взрыва

private bool explosionVisible = false;

…

// Движение корабля по игровому полю

private void MoveShip()

// Движение торпеды по прямой траектории (без смещения вправо)

private void MoveTorpedo()

1. Реализация интерфейса INotifyPropertyChanged:

public partial class ShipTorpedoGame : UserControl, INotifyPropertyChanged

{

// Событие для уведомления об изменении свойств (для привязки данных)

public event PropertyChangedEventHandler PropertyChanged;

// Метод для уведомления об изменении свойств (реализация INotifyPropertyChanged)

protected virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

1. Свойства с уведомлением об изменении:

public bool ShipVisible

{

get => shipVisible;

set

{

shipVisible = value;

// Уведомляем об изменении всех связанных свойств

OnPropertyChanged(nameof(ShipVisible));

OnPropertyChanged(nameof(ShipPeriscopeX));

OnPropertyChanged(nameof(ShipPeriscopeY));

}

}

// Состояние торпеды (свойство для привязки данных)

public bool TorpedoLaunched

{

get => torpedoLaunched;

set

{

torpedoLaunched = value;

// Уведомляем об изменении всех связанных свойств

OnPropertyChanged(nameof(TorpedoLaunched));

OnPropertyChanged(nameof(TorpedoPeriscopeX));

OnPropertyChanged(nameof(TorpedoPeriscopeY));

}

}

1. Привязка координат к элементам управления:

<!-- Корабль в перископе -->

<Path x:Name="shipElement" Data="M0,0 L15,0 L12.5,-5 L2.5,-5 Z" Fill="White"

Canvas.Left="{Binding ShipPeriscopeX}" Canvas.Top="{Binding ShipPeriscopeY}">

<Path.Style>

<Style TargetType="Path">

<Setter Property="Visibility" Value="Visible"/>

<Style.Triggers>

<DataTrigger Binding="{Binding ShipVisible}" Value="False">

<Setter Property="Visibility" Value="Collapsed"/>

</DataTrigger>

</Style.Triggers>

</Style>

</Path.Style>

<Path.RenderTransform>

<TranslateTransform X="-7.5" Y="2.5"/>

</Path.RenderTransform>

</Path>

<!-- Торпеда в перископе -->

<Ellipse x:Name="torpedoElement" Width="5" Height="2.5" Fill="White"

Canvas.Left="{Binding TorpedoPeriscopeX}" Canvas.Top="{Binding TorpedoPeriscopeY}">

<Ellipse.Style>

<Style TargetType="Ellipse">

<Setter Property="Visibility" Value="Visible"/>

<Style.Triggers>

<DataTrigger Binding="{Binding TorpedoLaunched}" Value="False">

<Setter Property="Visibility" Value="Collapsed"/>

</DataTrigger>

</Style.Triggers>

</Style>

</Ellipse.Style>

<Ellipse.RenderTransform>

<TranslateTransform X="-2.5" Y="-1.25"/>

</Ellipse.RenderTransform>

</Ellipse>

<!-- Взрыв в перископе -->

<Ellipse x:Name="explosion" Width="20" Height="20" Fill="Yellow" Opacity="0"

Canvas.Left="{Binding ExplosionPeriscopeX}" Canvas.Top="{Binding ExplosionPeriscopeY}">

<Ellipse.RenderTransform>

<TranslateTransform X="-10" Y="-10"/>

</Ellipse.RenderTransform>

</Ellipse>

5. Проверка попадания. Метод CheckCollision() для обнаружения столкновений:

private void CheckCollision()

{

if (!TorpedoLaunched || !ShipVisible) return; // Проверяем только если оба объекта активны

// Вычисляем расстояние между торпедой и кораблем по теореме Пифагора

double distance = Math.Sqrt(Math.Pow(torpedoX - shipX, 2) + Math.Pow(torpedoY - shipY, 2));

// Если расстояние меньше 30 пикселей - считаем что попадание

if (distance < 30)

{

Explode(); // Создаем взрыв

ShipVisible = false; // Скрываем корабль

ResetTorpedo(); // Сбрасываем торпеду

GameMessage?.Invoke("Попадание! Корабль уничтожен!"); // Отправляем сообщение

}

}

Метод обработки кликов для показа информации:

private void PeriscopeView\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

// Получаем координаты клика относительно перископа

Point clickPosition = e.GetPosition(periscopeView);

// Проверяем попадание по торпеде (в радиусе 10 пикселей)

double torpedoDistance = Math.Sqrt(

Math.Pow(clickPosition.X - TorpedoPeriscopeX, 2) +

Math.Pow(clickPosition.Y - TorpedoPeriscopeY, 2));

if (TorpedoLaunched && torpedoDistance < 10)

{

GameMessage?.Invoke(TorpedoInfo); // Показываем информацию о торпеде

return;

}

// Проверяем попадание по кораблю (в радиусе 15 пикселей)

double shipDistance = Math.Sqrt(

Math.Pow(clickPosition.X - ShipPeriscopeX, 2) +

Math.Pow(clickPosition.Y - ShipPeriscopeY, 2));

if (ShipVisible && shipDistance < 15)

{

GameMessage?.Invoke(ShipInfo); // Показываем информацию о корабле

return;

}

// Если клик не попал ни в один объект

GameMessage?.Invoke("Цельтесь в корабль или торпеду для информации");

}

// Информация о корабле для отображения (координаты)

public string ShipInfo => $"Корабль: X={shipX:F1}, Y={shipY:F1}, Скорость={shipSpeed:F1}";

// Информация о торпеде для отображения (скорость)

public string TorpedoInfo => $"Торпеда: Скорость={torpedoSpeed:F1}";

Методы управления игрой:

private void BtnStart\_Click(object sender, RoutedEventArgs e)

{

game.StartGame();

txtInfo.Text = "Игра запущена!";

}

private void BtnPause\_Click(object sender, RoutedEventArgs e)

{

game.PauseGame();

txtInfo.Text = "Игра на паузе";

}

private void BtnReset\_Click(object sender, RoutedEventArgs e)

{

game.ResetGame();

txtInfo.Text = "Игра сброшена";

}

private void BtnLeft\_Click(object sender, RoutedEventArgs e)

{

game.MovePeriscopeLeft();

}

private void BtnRight\_Click(object sender, RoutedEventArgs e)

{

game.MovePeriscopeRight();

}

private void BtnFire\_Click(object sender, RoutedEventArgs e)

{

game.LaunchTorpedo();

txtInfo.Text = "Торпеда выпущена!";

}

Изображение выглядит как текст, снимок экрана, диаграмма, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как снимок экрана, текст, программное обеспечение, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, снимок экрана, программное обеспечение, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Код проектов всех лабораторных работ, задания и отчеты находятся в публичном доступе на сайте GitHub: <https://github.com/Maxevgen555/SVPP_LB>. Все проекты находятся в одном решении и выполнены в VS2022. Файл решения SVPP\_LB.sln находится в папке \ЛБ1\LB1\_Calc\.

*Вывод:*

Я ознакомился с методикой создания пользовательских элементов управления. Изучил способы использования делегатов и событий. Создал проект типа WPF ControlLibrary. Запрограммировал пользовательский элемент управления. Добавил в элемент нужные свойства и события. Написал приложение типа WPF, использующее созданный элемент.