Лабораторная работа №5: Делегаты и события.

Цель работы:

– Знакомство с основными принципами ООП.

Теоретическая информация.

Делегаты - это основной механизм для создания отложенных методов и функций в C#. Они представляют собой ссылки на методы, которые можно вызывать как функции. Формально, делегат является типом данных, который представляет собой ссылку на метод. Он имеет сигнатуру (параметры и возвратные значения), подобную методу:

```
[доступность] delegate [возвращаемый тип] Название([параметры]);
```

Последовательность действий для создания и использования делегата:

1. Объявление делегата
2. Создание переменной делегата
3. Присваивание переменной адреса метода
4. Возов методов делегата

```
using System;

//Делегат без возвращаемых типов данных
//Принимает в сигнатуре две переменные
public delegate void MyDelegate(int a, int b);

class Program
{
    static void Main()
    {
        // создаем экземпляр делегата и присваиваем ему метод
        MyDelegate del = new MyDelegate(Add);

        // вызываем делегат
        del(5, 10);
        //в консоль будет выведено "15"
    }

    //метод передаваемый в делегат
    static void Add(int a, int b)
    {
        Console.WriteLine(a + b);
    }
}
```

Наиболее сильная сторона делегатов состоит в том, что они позволяют делегировать выполнение некоторому коду извне.

События - это один из основных механизмов для реализации взаимодействия между объектами в C#. Они представляют собой способ уведомления о произошедшем событии и позволяют другим объектам реагировать на это событие. Событие является типом данных, который представляет собой

связку из делегата и списка подписчиков. Делегат определяет сигнатуру события, а список подписчиков содержит ссылки на методы, которые хотят услышать о произошедшем событии.

Последовательность действий для создания и использования события:

1. Объявление события
2. Подписка на событие

```
//класс использующий события
public class BankAccount
{
    //событие
    public event EventHandler BalanceChanged;

    private int _balance = 0m;

    public int Balance
    {
        get { return _balance; }
        set
        {
            if (_balance != value)
            {
                _balance = value;
                OnBalanceChanged();
            }
        }
    }
    protected virtual void OnBalanceChanged()
    {
        //Генерация события
        BalanceChanged?.Invoke(this, EventArgs.Empty);
    }
}

class Program
{
    static void Main()
    {
        // создаем экземпляр банковского счета
        var account = new BankAccount();

        // подписываемся на событие изменения баланса
        account.BalanceChanged += delegate { Console.WriteLine("Баланс изменен!"); };

        // делаем несколько операций, которые изменяют баланс счета
        //будет вызван делегат с сообщением в консоль об изменении баланса
        account.Balance = 100;
        account.Balance = 200;
        account.Balance = 300;
    }
}
```

Задание №1. Использование событий для обработки действий игрока.

Выполнить примеры, приведенные в задании.

Функционал программы:

– К основному функционалу программы из лабораторной работы №3 добавляется обработка действий с помощью событий.

Задачи:

– Добавление и удаление объектов из поля реализовано с помощью событий;
– Реализовать добавление сообщения о получении очков в лист с помощью событий;

Для выполнения данного задания используется программа, разработанная в ходе выполнения первого задания лабораторной работы №3.

Для того чтобы создать обработчики событий по добавлению и удалению объектов из окна программы создадим в классе CController делегат:

```
//делегат
public delegate void SceneEvent(object sender, CControllerEventArgs e);

public class CController
{

    //тело класса
}
```

Теперь в основном теле класса можно создать обработчики событий на основе этого класса:

```
public class CController
{

    //ссылка на обработчик события добавления объекта в сцену
    public event SceneEvent addObject;

    //ссылка на обработчики событий удаления объекта из сцены
    public event SceneEvent removeObject;

}
```

В основном теле программы создадим две функции для добавления и удаления объектов из сцены:

```
//обработчик события добавление собираемого объекта в сцену
public void addObjectInScene(object sender, CControllerEventArgs e)
{
    scene.Children.Add(e.sprite);
}

//обработчик события удаления собираемого объекта из сцены
public void removeObjectFromScene(object sender, CControllerEventArgs e)
{
    scene.Children.Remove(e.sprite);
}
```

Данные функции требуется назначить обработчикам событий:

```
public MainWindow()
        {
            InitializeComponent();

            controller = new CController(0.2, 0, new Size(scene.Width, scene.Height));

            //назначение обработчика события добавления объекта в сцену
            controller.addObject += addObjectInScene;

            //при удалении объекта срабатывает обработчик удаления
            controller.removeObject += removeObjectFromScene;

         }
```

Как можно было заметить функции имеют два параметра – sender и некие аргументы. В качестве отправителя в данном случае будет выступать сам класс, а аргументы, отправляемые в функцию, требуется так же реализовать отдельно.

Аргументами будет класс, имеющий в себе информацию о спрайте, который потребуется создать в сцене:

```
//класс, описывающий аргументы событий класса контроллера
public class CControllerEventArgs : EventArgs
{
        //ссылка на визуальное представление собираемого объекта
        public Ellipse sprite;

        public CControllerEventArgs(Ellipse sprite)
        {
            this.sprite = sprite;
        }
    }
```

Теперь создадим функции создания и удаления объектов из сцены создадим функции, которые будут вызывать реализованные ранее события:

```
//создание нового объекта
private void spawnObject()
{
            Point position = new Point(rng.NextDouble() * (sceneSize.Width - maxSpriteSize) ,
rng.NextDouble() * (sceneSize.Height - maxSpriteSize));
            double size = (rng.NextDouble() * (maxSpriteSize - minSpriteSize)) +
minSpriteSize;
            double lifetime = (rng.NextDouble() * (maxLifetime - minLifetime)) + minLifetime;

            CObject newObject = new CObject(position, size, lifetime);
            objects.Add(newObject);

            //вызов события добавления обьекта
            addObject?.Invoke(this, new CControllerEventArgs(newObject.getSprite()));
}


//удаление объекта из списка и сцены
private void destroyObject(CObject o)
{
            CControllerEventArgs e = new CControllerEventArgs(o.getSprite());
```

```
            //вызов события удаления обьекта
            removeObject?.Invoke(this, e);
            objects.Remove(o);
}
```

Для добавления сообщения в листбокс так же требуется реализовать свое событие внутри класса. Для этого можно добавить еще одно действие в событие удаления обьекта. Прежде всего требуется реализовать функцию:

```
//обработчик события добавление сообщения в список
public void logMessage(object sender, CControllerEventArgs e)
        {
            message_log.Items.Insert(0, e.msg);
        }
```

И подписать событие на еще одну функцию

```
controller.removeObject += logMessage;
```

Модифицируем класс аргументов для того чтобы оно содержало сообщение, которое будет добавляться в список:

```
public class CControllerEventArgs : EventArgs
    {
        public Ellipse sprite;

        //сообщение связанное с действием
        public string msg = "";

        public CControllerEventArgs(Ellipse sprite)
        {
            this.sprite = sprite;
        }
    }
```

Так же требуется модифицировать и функцию вызывающую данное событие:

```
private void destroyObject(CObject o, string message)
        {
            CControllerEventArgs e = new CControllerEventArgs(o.getSprite());

            //установка сообщения связанного с действием
            e.msg = message;
            removeObject?.Invoke(this, e);

            objects.Remove(o);
        }
```

Задание №2. Использование событий для обработки действий в кликкере.

Используя примеры из предыдущего задания модифицируйте программу из лабораторной работы №4 для использования событий.

Задачи:

- Добавление и замена противника в основном окне программы реализовано с помощью событий;
- Добавление и удаление бонусных сфер в основном окне программы реализовано с помощью событий;
- Получение урона противником реализовано с помощью событий;