ЛАБОРАТОРНАЯ РАБОТА №4

Создание пользовательских элементов управления

1. Цель работы

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

2. Постановка задачи

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.

3. Индивидуальные задания

1. Разработать пользовательский элемент управления, реализующий игру “Бег лошадей по кругу ипподрома”. Скорость движения лошадей должна задаваться случайным образом. При щелчке левой клавишей мыши по изображению лошади должна выводиться информация о ее скорости, при щелчке правой клавишей мыши - информация о ее позиции.

2. Разработать пользовательский элемент управления, реализующий игру “Бег лошадей по прямой”. Скорость движения лошадей должна задаваться случайным образом. При щелчке правой клавиши мыши по изображению лошади должна выводиться информация о ее скорости, при щелчке левой клавишей мыши информация о ее позиции.

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.

4. Разработать пользовательский элемент управления, реализующий игру “Бомбометание с самолета по наземной цели”. С летящего с постоянной скоростью самолета при нажатии кнопки “Пуск” производится бомбометание. Траектория падения бомбы должна соответствовать физическим законам падения тел. Попадание в цель должно сопровождаться видимым взрывом и исчезновением цели. При щелчке левой клавиши мыши по бомбе должна выводиться информация о ее скорости, а при щелчке по самолету - информация о его положении.

5. Разработать пользовательский элемент управления, реализующий игру “Стрельба из пушки”. Пушка должна стрелять через гору по цели. Траектория полета снаряда должна подчиняться физическим законам. Игрок может управлять углом подъема ствола относительно горизонта и начальной скоростью снаряда. При попадании должен происходить видимый взрыв и исчезновение цели. При щелчке левой клавиши мыши по летящему снаряду должна выводиться информация о его положении, а при щелчке по пушке - информация о ее положении.

4. Рекомендации к выполнению задания.

Для каждого объекта игры создайте класс, описывающий основные свойства объекта (например, координаты, скорость и др.). Для движущихся объектов опишите метод, выполняющий одну итерацию перемещения объекта с последующим изменением текущих координат (например, метод Move()).

Если движущихся объектов много, то сохраните объекты в обобщенном списке, например в List<T>. В дальнейшем для выполнения одного перемещения всех объектов достаточно будет обойти коллекцию и вызвать метод Move() у каждого объекта.

Если у разных объектов разные алгоритмы перемещения (как в задании «Бомбометание» - самолет и бомба), то сначала опишите абстрактный класс, явно описывающий общие свойства и методы. Метод перемещения объектов нужно

сделать абстрактным. Производные классы должны будут реализовать этот абстрактный алгоритм. В качестве типа обобщенного списка объектов нужно будет указать базовый абстрактный класс.

Для того, чтобы графические элементы игры изменяли свое положение при изменении свойств координат соответствующих объектов выполните привязку положения элементов (например, свойств Canvas.Top и Canvas.Left) к соответствующим свойствам (X и Y)

4.1. Реализация привязки данных

4.1.1. Способы реализации привязки данных

Для осуществления привязки классы, описывающие объекты, должны либо реализовать интерфейс INotifyPropertyChanged, либо наследоваться от класса DependencyObject и реализовать свойства координат как свойства зависимости (Dependency Property).

4.1.2. Интерфейс INotifyPropertyChanged (пример)

```
class Plane : INotifyPropertyChanged
{
        double x, y;
     // Событие - свойство изменилось
     public event PropertyChangedEventHandler PropertyChanged;

        public double X
        {
            get { return x; }
            set
            {
                x = value;
                OnPropertyChanged("X");
            }
        }

        public double Y
        {
            get { return y; }
            set
            {
                y = value;
                OnPropertyChanged("Y");
            }
        }
```

```
        private void OnPropertyChanged(string v)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler!=null)
            {
                handler(this, new PropertyChangedEventArgs(v));
            }
        }

        public void Move()
        {
           . . .
        }
}
```

4.1.3. Dependency Property (пример)

```
abstract class Unit:DependencyObject
{
      public double X
         {
             get { return (double)GetValue(XProperty); }
             set { SetValue(XProperty, value); }
         }

         // Использование DependencyProperty для X.
         public static readonly DependencyProperty XProperty =
             DependencyProperty.Register("X", typeof(double),
                                 typeof(Unit), new PropertyMetadata());

         public double Y
         {
             get { return (double)GetValue(YProperty); }
             set { SetValue(YProperty, value); }
         }

         // Использование DependencyProperty для Y.
         public static readonly DependencyProperty YProperty =
             DependencyProperty.Register("Y", typeof(double),
```

```
                                    typeof(Unit), new PropertyMetadata(0));

        public abstract void Move();
    }
```

Для создания свойства зависимости можно воспользоваться автоматической генерацией кода Visual Studio. Для этого наберите “propdp” и нажмите клавишу табуляции (см. Рисунок 1).

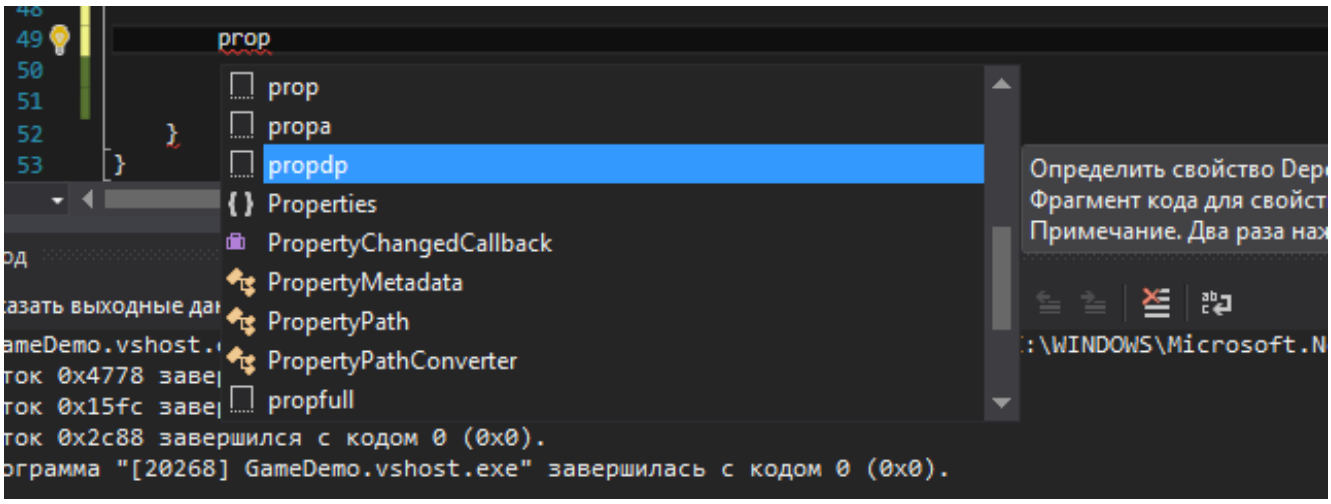

Рисунок 1. Добавление DependencyProperty

Вам останется только указать тип и имя свойства, а также имя класса-собственника (обычно имя своего же класса) (см. Рисунок 2).

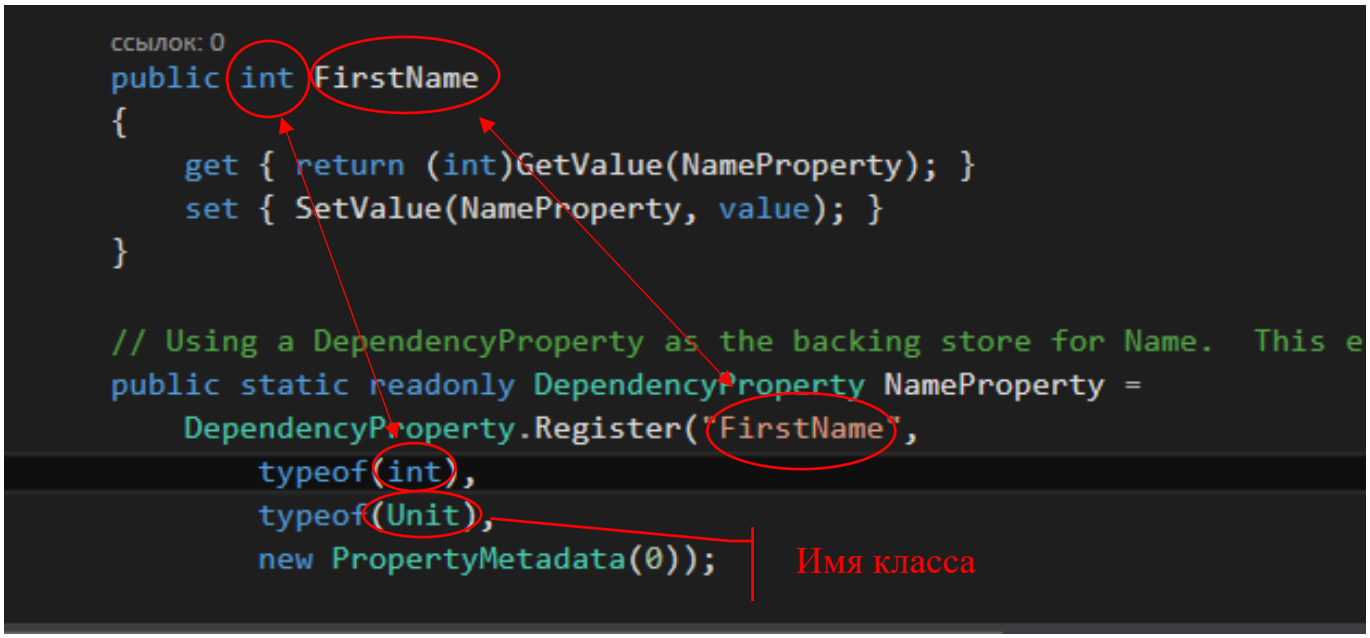

Рисунок 2. Настройка Dependency Property

4.1.4. Привязка объекта к элементу (пример)

```
var plane = new Plane();
Ellipse el = new Ellipse();
el.Width = 20;
el.Height = 20;
el.Stroke = new SolidColorBrush(Colors.Black);
el.DataContext = plane;
el.SetBinding(Canvas.TopProperty, new Binding("Y"));
el.SetBinding(Canvas.LeftProperty, new Binding("X"));
gameArea.Children.Add(el);
```

4.2. Проверка попадания точки в элемент на игровом поле

Для проверки можно использовать метод VisualTreeHelper.HitTest.

Например:

```
HitTestResult result = VisualTreeHelper
                       .HitTest(gameArea, new Point(plane.X, plane.Y));
if (result!=null)
{
     var element = result.VisualHit as FrameworkElement;
     if (element.Name == "ball")
     {
     . . .
      }
}
```

В данном примере gameArea – это имя контейнера Canvas