**Silverlight**

Jaagup Kippar

Sisukord

[Sissejuhatus 2](#_Toc326663949)

[Käivitamine 2](#_Toc326663950)

[Ülesandeid 3](#_Toc326663951)

[Hiirevajutusel tekkivad ruudud 3](#_Toc326663952)

[Ülesandeid 5](#_Toc326663953)

[Ülesandeid 7](#_Toc326663954)

[Näidislahendus 7](#_Toc326663955)

[Liikuv ruut 9](#_Toc326663956)

[Ülesandeid 10](#_Toc326663957)

[Liigutatav ruut 10](#_Toc326663958)

[Ülesandeid 12](#_Toc326663959)

[Soovitud arv ruute 12](#_Toc326663960)

[Ülesandeid 13](#_Toc326663961)

[Soovitud arv omaloodud komponente 14](#_Toc326663962)

[Ülesandeid 15](#_Toc326663963)

[Kujundite joonistamine 16](#_Toc326663964)

[Ülesandeid 17](#_Toc326663965)

[Hiiretundlikud kujundid 17](#_Toc326663966)

[Ülesandeid 19](#_Toc326663967)

[Jututahvli abil liigutamine 19](#_Toc326663968)

[Ülesandeid 20](#_Toc326663969)

[Jututahvli abil keeramine 20](#_Toc326663970)

[Ülesandeid 21](#_Toc326663971)

[Piltide vahetus 21](#_Toc326663972)

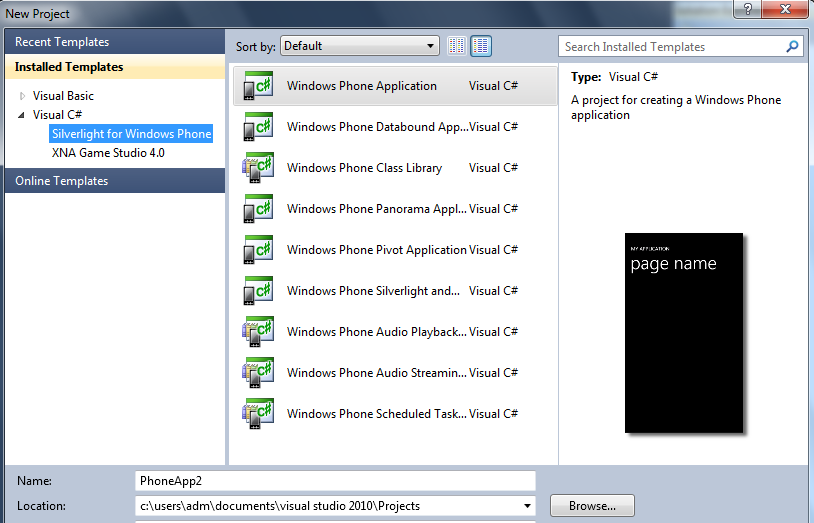
[Ülesandeid 24](#_Toc326663973)

## Sissejuhatus

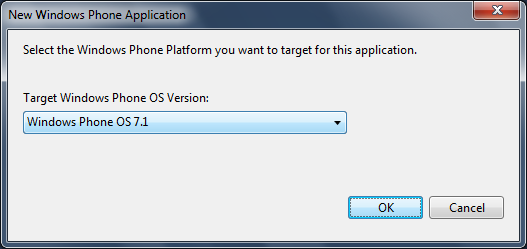
Järgnevalt tutvustatakse, kuidas programmeerimisvahendite abil Silverlightiga sagedamini ette tulevaid toimetusi tehakse.

## Käivitamine

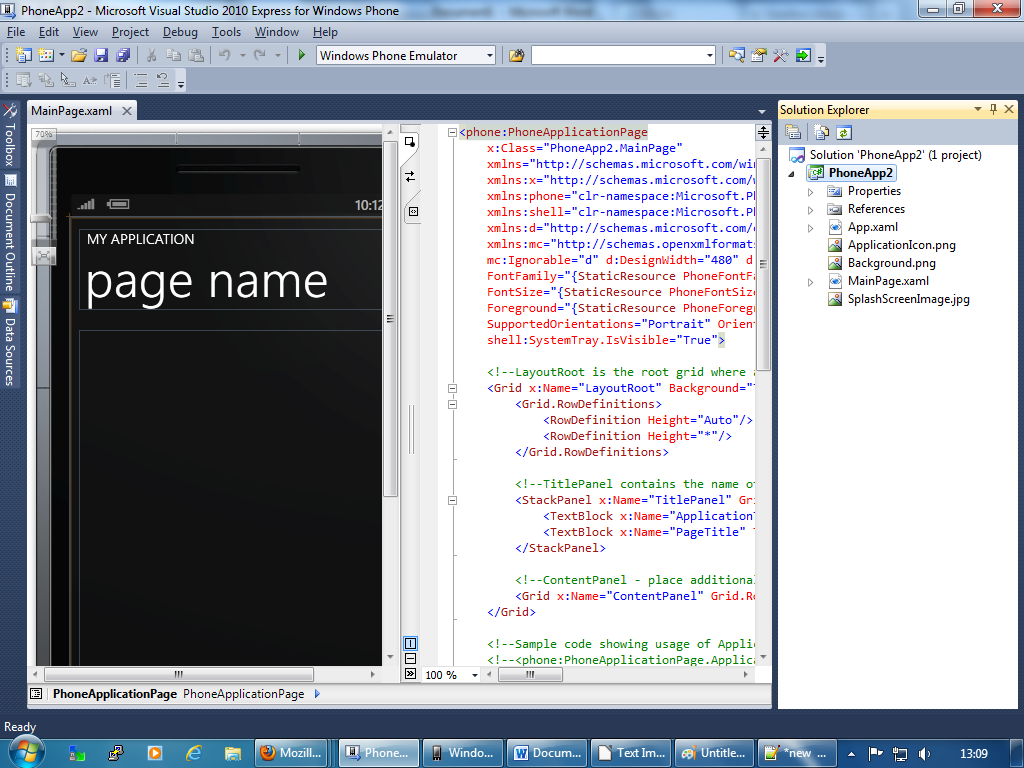
Mobiiliarenduse tarbeks loodud express-arendusvahendis on selle tarbeks mitmeid algmalle välja pakutud. Kõige lihtsamaks ja „puhtamaks“ neist on Windows Phone Application.



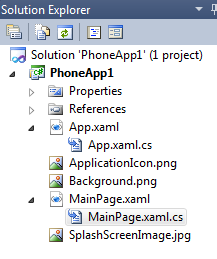
Rakenduse loomise juures palutakse valida ka telefoni operatsioonisüsteemi täpsem versioon.



Avanevas aknas ongi juba kasutada käivituva mobiilirakenduse kood. Käivituva rakenduse võimalik ekraanipilt sealsamas kõrval nähtav. Välimine element näitab, et tegemist mobiilis käivituva koodiga. Edasised elemendid on aga juba samasugused sõltumata lahenduse käivituskohast. Vaikimisi paigutushalduriks on Grid, kus võimalik elemente tabelina paigutada.



Rakenduse struktuuri poolest esialgu põhilise tähtsusega kood on failis MainPage.xaml. Sinna juurde kuulovad C# koodilõigud paigutuvad MainPage.xaml.cs-faili. Rakenduse suhtes üldised käsklused on App.xaml-i juures. Pildid ja muud staatiliselt kaasas olevad lisandfailid asuvad samuti rakenduse puus.



### Ülesandeid

* Loo lihtne rakendus
* Käivita rakendus emulaatoril, veendu selle toimimises.
* Muuda ära etteantud pealkiri

## Hiirevajutusel tekkivad ruudud

Ekraanil millegi joonistamiseks ning selle asukoha määramiseks pikslite kaupa sobib lõuend ehk Canvas. Selle poole programmikoodis pöördumiseks on kasulik lõuendile panna nimi (x:Name), siin näiteks on nimeks enamasti juur. Ka on lõuendile antud taustavärv – süsteemi omapärana on vaid sel juhul võimalik lõuendi abil mitmesuguseid sündmusi ning hiire/näpuvajutusi kinni püüda. Omadus MouseLeftButtonDown määrab, millise nimega funktsioon xaml-i juurde kuuluvas cs-failis käivitatakse. Siin puhul siis LayoutRoot\_MouseLeftButtonDown. Automaatselt genereerides küll püütakse nimed ja seosed enamvähem arusaadavateks jätta. Kuid iseenesest on tegemist vaid nimedega ning neid saab vajadusel muuta, samuti näiteks mõnda funktsiooni mitme eri sündmuse peale välja kutsuda.

<phone:PhoneApplicationPage

x:Class="PhoneApp1.MainPage"

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

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

xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"

xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"

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

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

mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"

FontFamily="{StaticResource PhoneFontFamilyNormal}"

FontSize="{StaticResource PhoneFontSizeNormal}"

Foreground="{StaticResource PhoneForegroundBrush}"

SupportedOrientations="Portrait" Orientation="Portrait"

shell:SystemTray.IsVisible="True">

<!--LayoutRoot is the root grid where all page content is placed-->

<Canvas x:Name="juur" Background="LightGray" MouseLeftButtonDown="LayoutRoot\_MouseLeftButtonDown" Height="auto" Width="auto">

</Canvas>

</phone:PhoneApplicationPage>

Programmikoodi cs-failis kirjeldatakse siis lähemalt ära mitmesugused lehe juures vajalikud toimetused. Automaatselt tõmmatakse sinna sisse hulgem nimeruume, mis rakenduse loomise juures võiksid kasulikuks osutuda. Hiljem saab seda loetelu vajadusel täiendada või puhastada. Konstruktoris ehk klassiga samanimelises käskluses olev InitializeComponent laseb xaml-failis paikneva paigutuse paika sättida. Kõiksugu muud käsklused on juba lahenduse enese sobivamaks muutmiseks kasutatavad.

Siin näites siis lisati funktsioon LayoutRoot\_MouseLeftButtonDown, mille esimene parameeter (sender) määrab sündmuse allika/alustaja ning teine võimalikud lisaparameetrid – näiteks, et kas tegemist ühekordse või topeltklõpsuga. Allikas näitab enamasti, et millist lehel olevat elementi tabagi – vajalik näiteks juhul, kui vaja eristada, millist lehel olevat nuppu vajutati.

Siin näites luuakse iga vajutuse peale ristkülik – class Rectangle on kättesaadav System.Windows.Shapes nimeruumist. Hetkel tehakse ta seest punane ning laiuse ja kõrgusega 20 ekraanipunkti. Ekraanil paigutamiseks on käsud Canvas.setLeft ning Canvas.setTop – erinevalt mõnest muust programmeerimisvahendist on siin paigutuse andmed mitte elemendi enese, vaid teda ümbritseva lõuendi hallata. Iga elemendi loomisel tuleb hoolitseda, et see ka rakenduse kujundusosa külge liidetakse – siin siis juur.Children.Add-käsklus selle tarbeks.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

using Microsoft.Phone.Controls;

namespace PhoneApp1

{

public partial class MainPage : PhoneApplicationPage

{

// Constructor

public MainPage()

{

InitializeComponent();

}

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

{

Rectangle r = new Rectangle();

r.Fill = new SolidColorBrush(Colors.Red);

r.Width = 20;

r.Height = 20;

Canvas.SetLeft(r, e.GetPosition(juur).X);

Canvas.SetTop(r, e.GetPosition(juur).Y);

juur.Children.Add(r);

}

}

}

Tulemuseks rakendus, kus vajutuste kohale tekivad ruudud.

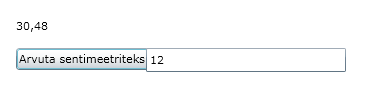


### Ülesandeid

* Muuda iga järgnev ruut eelmisest veidi väiksemaks
* Ruutude asemele tekivad ellipsid/ringid (klass Ellipse samade parameetritega kui Rectangle)
* Vajutuste tulemusena tekib kindla kujuga pilt (nt. klotsidest robot), iga vajutusega lisandub üks klots.

Andmete sisestus

Järgnev kood seletab, kuidas sisestatud tollide arvust sentimeetrid leida – ehk siis universaalne näide andmete sisestamiseks, arvutamiseks ning tulemuste väljastamiseks. Lõpptulemus pildina järgmine:



Vastuse näitamiseks kasutatakse TextBlock-i, sisestamiseks TextBox-i, koodi käivitamiseks nupp. Kui tahta elementi pikslite kaupa paigutada, siis aitavad selleks parameetrid Canvas.Top ja Canvas.Left – sarnaselt kui ennist programmikoodi abil asju paika sätiti. Ning nagu aimata võib, siis nupp1\_Click viitab koodifailis samanimelisele alamprogrammile. Pikast elemendi koodist siia vaid sisemine lõuendi osa välja toodud.

xaml:

<Canvas>

<TextBlock x:Name="plokk1" Canvas.Top="10" Canvas.Left="20"

Width="200" Height="50" >

</TextBlock>

<Button x:Name="nupp1" Content="Arvuta sentimeetriteks"

Canvas.Top="40" Canvas.Left="20" Click="nupp1\_Click" ></Button>

<TextBox x:Name="txttollid" Width="200"

Canvas.Top="40" Canvas.Left="150" />

</Canvas>

Arvuti jaoks on tekstid ja arvud erinevad. Seetõttu tuleb siin hoolitseda, et sisend kõigepealt tekstist arvuks muudetaks ning pärast kuvamiseks tulemus taas tagasi tekstiks tehtaks. Andmete muundamiseks suhteliselt universaalne klass on Convert. Kui midagi on vaja aga tekstiks ehk stringiks teha, siis selleks aitab üldjuhul iga objekti küljes kasutada olev käsklus ToString().

cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class NupudLouendil : UserControl

{

public NupudLouendil()

{

InitializeComponent();

}

private void nupp1\_Click(object sender, RoutedEventArgs e)

{

double cm = Convert.ToDouble(txttollid.Text) \* 2.54;

plokk1.Text = cm.ToString();

}

}

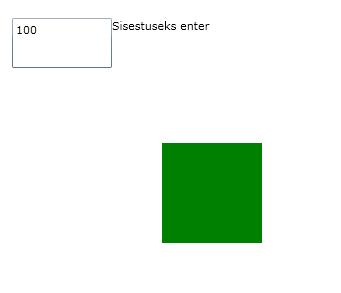
}

### Ülesandeid

* Väljasta kahe sisestatud arvu korrutis
* Võimalda kasutajal arvu abil määrata ekraanil oleva ruudu külje pikkus

### Näidislahendus

Ruudus suuruse muutmise näide.



<Grid x:Name="juur" Background="White">

<Grid.RowDefinitions>

<RowDefinition Height="50" />

<RowDefinition />

</Grid.RowDefinitions>

<StackPanel Grid.Row="0" Orientation="Horizontal">

<TextBox x:Name="txtSuurus" Text="100" Width="100" KeyDown="txtSuurus\_KeyDown"></TextBox>

<TextBlock>Sisestuseks enter</TextBlock>

</StackPanel>

<Rectangle x:Name="ruut1" Width="100" Height="100" Fill="Green" Grid.Row="1"></Rectangle>

</Grid>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class RuuduSuurus : UserControl

{

public RuuduSuurus()

{

InitializeComponent();

}

private void txtSuurus\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Enter) {

int kyljepikkus = Convert.ToInt32(txtSuurus.Text);

ruut1.Width = kyljepikkus;

ruut1.Height = kyljepikkus;

}

}

}

}

Ruudu suurust on võimalik ka otse tekstivälja väärtusega siduda. Siin näites siis ruudu omadus Width loetakse otse tekstikastist nimega txtSuurus.

<Grid x:Name="juur" Background="White">

<Grid.RowDefinitions>

<RowDefinition Height="50" />

<RowDefinition />

</Grid.RowDefinitions>

<StackPanel Grid.Row="0" Orientation="Horizontal" HorizontalAlignment="Center">

<TextBox x:Name="txtSuurus" Text="100" Width="100" ></TextBox>

</StackPanel>

<Rectangle x:Name="ruut1" Width="{Binding Text, ElementName=txtSuurus}" Height="100" Fill="Green" Grid.Row="1"

></Rectangle>

</Grid>

## Liikuv ruut

Pildina näha ruut kord ühes kohas



mõnigase ajaga aga sõidab teine sujuvalt natukese maad edasi.



Ruudu ekraanile kuvamiseks piisab vaid lõuendi sisse vastavast Rectangle-käsust.

<Canvas x:Name="juur">

<Rectangle x:Name="ruut1" Fill="Green" Width="100" Height="100" />

</Canvas>

Kella järgi millegi iseeneslikult liikuma panekuks sobib DispatcherTimer. Tuleb talle teada anda, kui sageli miskit funktsiooni käivitama asuda. Ning edasi selle sisse saab juba koodi kirjutada, milliseid elemente ja kuhu liigutada soovitakse. Taimeri käivitamiseks vajalikud käsud tuleb InitializeComponent-käskluse järele konstruktorisse – nõnda saab kindel olla, et need rakenduse juures vaid ühe korra käivituvad ning tegemist on meil üheaegselt ühe taimeriga. Praegusel juhul iga tiksu puhul suurendatakse ruudu x-koordinaati ning tulemuseks võibki näha vaikselt paremale liikuvat ruutu.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class RuuduLiikumine : UserControl

{

int x = 0;

public RuuduLiikumine()

{

InitializeComponent();

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

kell.Interval = new TimeSpan(0, 0, 0, 0, 100); // 100 Milliseconds

kell.Tick += new EventHandler(tiks);

kell.Start();

}

public void tiks(object o, EventArgs saatja)

{

x++;

Canvas.SetLeft(ruut1, x);

}

}

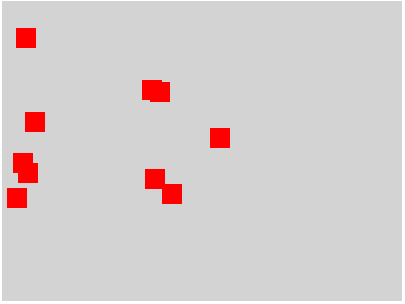
}

### Ülesandeid

* Pane ruut liikuma ülevalt alla
* Pane ekraanile korraga liikuma kaks ruutu – üks alla, teine üles
* Pane ruut liikuma mööda ruudukujulist trajektoori – alla, vasakule, paremale, üles
* Loo illusioon ruudu lähenemisest ja kaugenemisest: alla, väiksemaks (kaugemale), üles, suuremaks (lähemale). Kaugusefekti saavutamiseks võib kauguse koordinaatide keskpunktist läbi jagada kujundi kaugusega Z-teljel.
* Katseta ringjoonelist liikumist (x=raadius\*sin(nurk), y=raadius\*cos(nurk)).

## Liigutatav ruut

Järgnevas näites saab ekraanil olevaid ruute hiirega liigutada



Xaml-fail püsib lihtsana. Vaid lõuend ning käsklus seal peal hiire liikumise jälgimiseks.

<Canvas x:Name="juur" Background="LightGray" Height="auto" Width="auto" MouseMove="juur\_MouseMove">

</Canvas>

Programmikood aga mõnevõrra pikem. Abivahenditeks juhuarvude generaator ning meelespeetav ruut. Juhuarvude generaator võimaldab ruudud juhuslikult mööda ekraani paigutada. rnd.Next(200) annab igale loodud ruudule juhusliku asukoha mõlemal teljel nulli ja 200 vahel. Iga ristküliku külge kinnitatakse ka hiirekuular, et oleks võimalik hiirevajutustele reageerida. Operaator += lisab sündmuse vastava elemendi, praegusel juhul loodud ristküliku külge.

Funktsiooni hiir\_alla esimese parameetri tüübiks on object. Et me aga teame, et hiirekuularid on vaid ristkülikutel, siis saame käsuga saatja as Rectangle vastava tüübi ära muuta ning tulemuse Rectangle-tüüpi muutujale omistada. Sealtkaudu võimalik siis hiljem tema asukohta ja omadusi muuta.

Funktsioon hiir\_yles määrab meelespeetava ruudu tühiviidaks (null) ning enam midagi ei liigutata. Kui aga üks ruut on meeles, siis ta liigub vastavalt hiire/näpu liikumisele kaasa.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class RuuduHiir : UserControl

{

Random rnd = new Random();

Rectangle meeles = null;

public RuuduHiir()

{

InitializeComponent();

for (int i = 0; i < 10; i++)

{

Rectangle r = new Rectangle();

r.Fill = new SolidColorBrush(Colors.Red);

r.Width = 20;

r.Height = 20;

Canvas.SetLeft(r, rnd.Next(200));

Canvas.SetTop(r, rnd.Next(200));

juur.Children.Add(r);

r.MouseLeftButtonDown += new MouseButtonEventHandler(hiir\_alla);

r.MouseLeftButtonUp += new MouseButtonEventHandler(hiir\_yles);

}

}

private void hiir\_alla(object saatja, MouseButtonEventArgs e)

{

meeles = saatja as Rectangle;

}

private void hiir\_yles(object saatja, MouseButtonEventArgs e)

{

meeles = null;

}

private void juur\_MouseMove(object sender, MouseEventArgs e)

{

if (meeles != null)

{

Canvas.SetLeft(meeles, e.GetPosition(null).X);

Canvas.SetTop(meeles, e.GetPosition(null).Y);

}

}

}

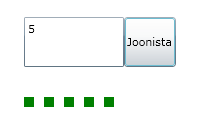
}

### Ülesandeid

* Iga vajutuse peale läheb liigutatav ruut veidi väiksemaks.
* Piisavalt väikeseks muudetud ruut kaob ekraanilt
* Kui vajutatakse ruudust mööda, siis tekib uus ruut ekraanile sinna kohta juurde.

## Soovitud arv ruute

Kui mõnda tegevust tuleb korrata ning eriti, kui selle juures pole lõplik arv ette teada – sellisel juhul on heaks abiliseks tsüklid. Loodava näite juures määrab kasutaja, mitu ruutu ta soovib näha ning programm teeb talle vastava koguse ruute.



Andmete sisestamiseks tekstikast ja nupp, tulemuse joonistamiseks Canvas ehk lõuend.

<Grid x:Name="juur" Background="White">

<Grid.RowDefinitions>

<RowDefinition Height="50" />

<RowDefinition />

</Grid.RowDefinitions>

<StackPanel Grid.Row="0" Orientation="Horizontal">

<TextBox x:Name="txtKogus" Text="5" Width="100"></TextBox>

<Button Content="Joonista" Click="Button\_Click" />

</StackPanel>

<Canvas Grid.Row="1" Height="auto" Width="auto" x:Name="ala1"></Canvas>

</Grid>

Ruutude joonistamiseks nupuvajutuse külge vastav käsklus. Children.Clear() ütleb et lõuendilt tuleb kõik olemasolevad komponendid eemaldada – nõnda saame puhtalt lehelt alustada. Muutuja i loendab, et mitmenda loodava ruudu juures me oleme. Seda tasub kasutada ka ruudu asukoha määramisel, et need üksteisele otsa ei satuks. UpdateLayout pärast lisamisi annab teada, et muutunud elemendid tasub ka uuesti ekraanile kuvada.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class RuutudeArv : UserControl

{

public RuutudeArv()

{

InitializeComponent();

}

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

{

ala1.Children.Clear();

int kogus = Convert.ToInt32(txtKogus.Text);

for (int i = 0; i < kogus; i++) {

Rectangle r = new Rectangle();

r.Width = 10;

r.Height = 10;

r.Fill = new SolidColorBrush(Colors.Green);

Canvas.SetLeft(r, i \* 20);

Canvas.SetTop(r, 30);

ala1.Children.Add(r);

}

ala1.UpdateLayout();

}

}

}

### Ülesandeid

* Joonista tulpdiagramm, kus iga järgmine tulp on eelmisest pikem
* Tulpade pikkused on võrdelised tulba järjekorranumbri ruuduga
* Joonista kasutaja poolt ette antud ridade ja veergudega ruudustik
* Hoolitse, et ruudud oleksid üle ühe erinevat värvi

## Soovitud arv omaloodud komponente

Siiani kasutasime vaid valmiskomponente ehk põhiliselt ristkülikuid oma jooniste koostamisel. Silverlighti üheks võluks aga on panna lahenduse tükid „elama“. Ning seda on võimalik just saavutada, tehes kõigepealt piisavalt targad komponendid eraldi valmis ning siis asuda neid omavahel kombineerima.

Omaloodud komponendiks kõigepealt element nimega Animatsioon1. Tegemist taas lihtsa ruuduga, kus aga hiirega peale liikumisel pannakse jututahvli abil tööle animatsioon – ruudu laius muutub 20st 30ks ja tagasi. Ristküliku küljes olev ruut1\_MouseEnter-funktsioon paneb vastava käsu käima.

Animatsioon1.xaml

<UserControl x:Class="SilverlightNaited.Animatsioon1"

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

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

Width="40" Height="40">

<Grid x:Name="juur" Background="White">

<Grid.Resources>

<Storyboard x:Name="suurendus">

<DoubleAnimation

Storyboard.TargetName="ruut1" Storyboard.TargetProperty="Width"

From="20" To="30" Duration="0:0:1" AutoReverse="True" />

</Storyboard>

</Grid.Resources>

<Rectangle x:Name="ruut1" Fill="Green" Width="20" Height="20" MouseEnter="ruut1\_MouseEnter" ></Rectangle>

</Grid>

</UserControl>

Animatsioon1.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class Animatsioon1 : UserControl

{

public Animatsioon1()

{

InitializeComponent();

}

private void ruut1\_MouseEnter(object sender, MouseEventArgs e)

{

suurendus.Begin();

}

}

}

Põhiprogrammis pole edasi siis enam muud teha, kui et tuleb eelnenud näites olevate ruutude asemele lihtsalt omapoolsed komponendid panna ning andekam tulemus ongi olemas.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class RuutudeArv2 : UserControl

{

public RuutudeArv2()

{

InitializeComponent();

}

private void txtKogus\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Enter) {

ala1.Children.Clear();

int kogus = Convert.ToInt32(txtKogus.Text);

for (int i = 0; i < kogus; i++)

{

Animatsioon1 a = new Animatsioon1();

Canvas.SetLeft(a, i \* 40);

Canvas.SetTop(a, 30);

ala1.Children.Add(a);

}

ala1.UpdateLayout();

}

}

}

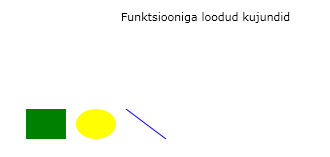
}

### Ülesandeid

* Koosta kasvava pikkusega ristkülikutest tulpdiagramm. Hiirega ristküliku peale liikumisel ta laius korraks kasvab.
* Koosta komponendiks soovitud raadiusega ring, mille keskel on näha ringi raadius arvuna. Hiirega ringi peale vajutamisel on võimalik ringi raadiust eraldi tekstikastis muuta.

## Kujundite joonistamine

Lisaks ristkülikule siis ka ellipsi ning joone loomise näide.



xaml-failis vaid kujundid koos juurdekuuluva tekstiga. Iseenesest piisaks ka vaid lõuendist.

<Grid x:Name="juur" Background="White">

<Grid.RowDefinitions>

<RowDefinition Height="50" />

<RowDefinition />

</Grid.RowDefinitions>

<StackPanel Grid.Row="0" Orientation="Horizontal" HorizontalAlignment="Center">

<TextBlock>Funktsiooniga loodud kujundid</TextBlock>

</StackPanel>

<Canvas Grid.Row="1" Height="auto" Width="auto" x:Name="ala1"></Canvas>

</Grid>

Kujundite loomiseks on eraldi funktsioonid tehtud, et neid hiljem mugav välja kutsuda oleks. Et ristkülikul ja ellipsil enamik käsklusi sarnased, siis saab siinsed koodiasjad ühendada. Joon aga tahab omi andmeid saada otspunktide kaudu – seetõttu joone jaoks ka eraldi alalmprogramm.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class KujunditeJoonistus : UserControl

{

public KujunditeJoonistus()

{

InitializeComponent();

joonista(ala1, new Rectangle(), 20, 50, 40, 30, Colors.Green);

joonista(ala1, new Ellipse(), 70, 50, 40, 30, Colors.Yellow);

joon(ala1, 120, 50, 160, 80, Colors.Blue);

}

Shape joon(Canvas ala, double x1, double y1, double x2, double y2, Color varv) {

Line j=new Line();

j.X1 = x1;

j.Y1 = y1;

j.X2 = x2;

j.Y2 = y2;

j.Stroke = new SolidColorBrush(varv);

ala.Children.Add(j);

return j;

}

Shape joonista(Canvas ala, Shape kujund, double vasak, double yla, double laius, double korgus, Color varv) {

Canvas.SetLeft(kujund, vasak);

Canvas.SetTop(kujund, yla);

kujund.Width = laius;

kujund.Height = korgus;

kujund.Fill = new SolidColorBrush(varv);

ala.Children.Add(kujund);

return kujund;

}

}

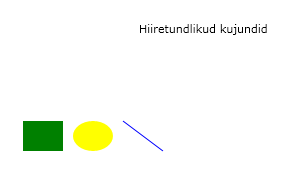
}

### Ülesandeid

* Koosta kriipsujuku
* Loo joonistusvahend, kus kasutajal võimalik valida soovitud kujundit ning see lõuendile paigutada.

## Hiiretundlikud kujundid

Kujunditele hulgem omaduste külge panekuks on nad mugav kõigepealt mõnda kogumisse korjata. Hästi sobib selleks näiteks LinkedList – sealt on neid hiljem võimalik ka suhteliselt vähese vaevaga ükshaaval eemaldada.



<Grid x:Name="juur" Background="White">

<Grid.RowDefinitions>

<RowDefinition Height="50" />

<RowDefinition />

</Grid.RowDefinitions>

<StackPanel Grid.Row="0" Orientation="Horizontal" HorizontalAlignment="Center">

<TextBlock>Hiiretundlikud kujundid</TextBlock>

</StackPanel>

<Canvas Grid.Row="1" Height="auto" Width="auto" x:Name="ala1"></Canvas>

</Grid>

Tsükliga foreach pannakse iga kujundi külge hiire sisenemise ning väljumise funktsioon. Hiljem saab vastavalt hiire sündmustele kujundi läbipaistvust mõjutada ning selle kaudu lahendus kasutaja suhtes „ärksamaks“ muuta.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class KujunditeJoonistus2 : UserControl

{

LinkedList<Shape> kujundid = new LinkedList<Shape>();

public KujunditeJoonistus2()

{

InitializeComponent();

kujundid.AddLast(joonista(ala1, new Rectangle(), 20, 50, 40, 30, Colors.Green));

kujundid.AddLast(joonista(ala1, new Ellipse(), 70, 50, 40, 30, Colors.Yellow));

kujundid.AddLast(joon(ala1, 120, 50, 160, 80, Colors.Blue));

foreach(Shape kujund in kujundid){

kujund.MouseEnter += new MouseEventHandler(hiirSisenes);

kujund.MouseLeave+=new MouseEventHandler(hiirLahkus);

}

}

Shape joon(Canvas ala, double x1, double y1, double x2, double y2, Color varv)

{

Line j = new Line();

j.X1 = x1;

j.Y1 = y1;

j.X2 = x2;

j.Y2 = y2;

j.Stroke = new SolidColorBrush(varv);

ala.Children.Add(j);

return j;

}

void hiirSisenes(object sender, MouseEventArgs e)

{

Shape kujund = sender as Shape;

kujund.Opacity = kujund.Opacity / 2;

}

void hiirLahkus(object sender, MouseEventArgs e)

{

Shape kujund = sender as Shape;

kujund.Opacity = kujund.Opacity \* 2;

}

Shape joonista(Canvas ala, Shape kujund, double vasak, double yla, double laius, double korgus, Color varv)

{

Canvas.SetLeft(kujund, vasak);

Canvas.SetTop(kujund, yla);

kujund.Width = laius;

kujund.Height = korgus;

kujund.Fill = new SolidColorBrush(varv);

ala.Children.Add(kujund);

return kujund;

}

}

}

### Ülesandeid

* Võimalda luua ekraanile mitmesuguseid kujundeid.
* Neid saab hiljem hulgi valida ning neil korraga omadusi muuta – näiteks suurendust ja värvi.

## Jututahvli abil liigutamine

Jututahvel ehk Storyboard on mugav vahend mitmesuguste animatsioonide sidumiseks elementidega. Lihtsaima näitena muudetakse sealtkaudu kujundi suurust.



<Grid x:Name="juur" Background="White">

<Grid.Resources>

<Storyboard x:Name="suurendus">

<DoubleAnimation

Storyboard.TargetName="ruut1" Storyboard.TargetProperty="Width"

From="20" To="30" Duration="0:0:1" AutoReverse="True" />

</Storyboard>

</Grid.Resources>

<Rectangle x:Name="ruut1" Fill="Green" Width="20" Height="20" MouseEnter="ruut1\_MouseEnter" ></Rectangle>

</Grid>

Koodi pool piisab vaid vastava omaduse käivitamisest – praegusel juhul sel nimeks „suurendus“.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class Animatsioon1 : UserControl

{

public Animatsioon1()

{

InitializeComponent();

}

private void ruut1\_MouseEnter(object sender, MouseEventArgs e)

{

suurendus.Begin();

}

}

}

### Ülesandeid

* Katseta mitmesuguste omaduste (asukoht, läbipaistvus) muutmist.
* Katseta mitmesuguseid muutmise kiirusi ning suundi.

## Jututahvli abil keeramine



Keeramise tarbeks peab kujundile eraldi lisama RotateTransformi. Siis on võimalik sealseid väärtusi muutes kujundit omale sobivalt keerama saada. Tuleb sealjuures ära määrata keerlemise keskpunkt ning algnurk.

<Grid x:Name="juur" Background="White">

<Grid.Resources>

<Storyboard x:Name="sb1" Completed="sb1\_Completed">

<DoubleAnimation

Storyboard.TargetName="keeramine" Storyboard.TargetProperty="Angle"

From="30" To="390" Duration="0:0:2" />

</Storyboard>

</Grid.Resources>

<Rectangle x:Name="ruut1" Fill="Green" Width="20" Height="20" MouseEnter="ruut1\_MouseEnter" >

<Rectangle.RenderTransform>

<RotateTransform x:Name="keeramine" CenterX="10" CenterY="10" Angle="30"></RotateTransform>

</Rectangle.RenderTransform>

</Rectangle>

</Grid>

Järgnevas koodis hoolitsetakse, et iga hiire sisenemise peale ei hakataks uuesti otsast peale keerama, vaid et tehtaks üks ring enne lõpuni ning ollakse alles siis valmis uut teadet vastu võtma.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class Animatsioon2 : UserControl

{

bool seisab = true;

public Animatsioon2()

{

InitializeComponent();

}

private void ruut1\_MouseEnter(object sender, MouseEventArgs e)

{

if (seisab)

{

sb1.Begin();

seisab = false;

}

}

private void sb1\_Completed(object sender, EventArgs e)

{

seisab = true;

}

}

}

### Ülesandeid

* Loo osuti, määra talle RotateTransform ning võimalda seda pöörata
* Katsu mitu osutit üheskoos panna kella kombel liikuma.

## Piltide vahetus

Ehk siis veidi pikem näide veebiandmeid kasutava pildijada loomiseks

Avanedes on näha üks pilt.



Pildile vajutades läheneb kaugusest uus pilt samal ajal keereldes.



Mõne aja pärast on ta kohal ning siis võib uut pilti uudistada.



Alustuseks on xaml-failis ees üks pilt. Selle küljes olev PlaneProjection võimaldab kujundit ümber telgede pöörata. Jututahvlil kirjeldus suurendab üheaegselt nii Y-telje suunalist pööret kui ka pildi suurust.

<Canvas Background="White" >

<Canvas.Resources>

<Storyboard x:Name="tahvel1">

<DoubleAnimation From="-180" To="0" Storyboard.TargetName="proj1" Storyboard.TargetProperty="RotationY" />

<DoubleAnimation From="0" To="400" Storyboard.TargetName="pilt1" Storyboard.TargetProperty="Width" />

</Storyboard>

</Canvas.Resources>

<Image Source="http://minitorn.tlu.ee/~jaagup/oma/ansambel/lootspillipoisid.jpg" x:Name="pilt1" Canvas.Left="10" Canvas.Top="30" Width="300" MouseLeftButtonDown="pilt1\_MouseLeftButtonDown">

<Image.Projection>

<PlaneProjection x:Name="proj1" />

</Image.Projection>

</Image>

</Canvas>

Koodifailis võetakse iga nupuvajutuse tulemusena ette uus pilt ning pannakse animatsioon tööle.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media.Imaging;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

namespace SilverlightNaited

{

public partial class PiltideKeeramine : UserControl

{

int koht = 0;

string[] pildiaadressid ={

"http://minitorn.tlu.ee/~jaagup/oma/ansambel/lootspillipoisid.jpg",

"http://minitorn.tlu.ee/~jaagup/oma/ansambel/pildid/kolmmerekividel.jpg",

"http://minitorn.tlu.ee/~jaagup/oma/ansambel/pildid/enrikuksel.jpg",

"http://minitorn.tlu.ee/~jaagup/oma/ansambel/pildid/ainjaagupkannel2.jpg"

};

public PiltideKeeramine()

{

InitializeComponent();

}

private void pilt1\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

koht++;

if (koht >= pildiaadressid.Length) { koht = 0; }

pilt1.Source = new BitmapImage(new Uri(pildiaadressid[koht], UriKind.Absolute));

tahvel1.Begin();

}

}

}

### Ülesandeid

* Muuda koodi nõnda, et pilti keerataks ümber Z-telje
* Tee lahendus nõnda ümber, et kasutajale jääks mulje pildilehtede keeramisest albumis.