ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Доцент факультета компьютерных наук, заместитель декана по учебно-методической работе, канд. социол. наук

_____ И. Ю. Самоненко
«_____» _____ 2020 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

_____ В. В. Шилов
«_____» _____ 2020 г.

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.13-01 ТП 01-1				

Приложение для визуализации метода рекурсивного спуска

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.13-01 ТП 01-1-ЛУ

Исполнитель:
студент группы БПИ 199
_____ К. Н. Борисов
«_____» _____ 2020 г.

Москва 2020

УТВЕРЖДЕН
RU.17701729.04.13-01 ТП 01-1-ЛУ

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.13-01 ТП 01-1				

Приложение для визуализации метода рекурсивного спуска

Текст программы

RU.17701729.04.13-01 ТП 01-1

Листов 30

Москва 2020

Содержание

1 ParserApp

1.1 ParserApp.csproj

```
<Project Sdk="Microsoft.NET.Sdk.WindowsDesktop">

  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net45</TargetFramework>
    <UseWPF>true</UseWPF>
    <ApplicationIcon>logo.ico</ApplicationIcon>
  </PropertyGroup>

  <ItemGroup>
    <Content Include="tutorials\*.*">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
    </Content>
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Emoji.Wpf" Version="0.0.19-experimental" />
    <PackageReference Include="Newtonsoft.Json" Version="12.0.3" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\ParserLib\ParserLib.csproj" />
  </ItemGroup>

  <ItemGroup>
    <Reference Include="System.Windows.Forms" />
  </ItemGroup>

  <Target Name="PreBuild" BeforeTargets="PreBuildEvent">
    <Exec Command="copy /Y ..\pre-commit ..\.git\hooks" />
    <!-- magick convert -density 384 -background transparent logo.svg -define
    icon:auto-resize -colors 256 logo.ico  -->
  </Target>

</Project>
```

1.2 App.xaml

```
<Application x:Class="ParserApp.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:ParserApp"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
```

```
    </Application.Resources>
</Application>
```

1.3 App.xaml.cs

```
using System.Windows;

namespace ParserApp {
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application {
    }
}
```

1.4 AssemblyInfo.cs

```
using System.Windows;

[assembly: ThemeInfo(
    ResourceDictionaryLocation.None, //where theme specific resource dictionaries
    are located
                                     //(used if a resource is not found in the
                                     page,
                                     // or application resource dictionaries)
    ResourceDictionaryLocation.SourceAssembly //where the generic resource
    dictionary is located
                                               //(used if a resource is not found
                                               in the page,
                                               // app, or any theme specific
                                               resource dictionaries)
)]
```

1.5 MainWindow.xaml

```
<Window x:Class="ParserApp.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:ParserApp"
        xmlns:emoji="clr-namespace:Emoji.Wpf;assembly=Emoji.Wpf"
        mc:Ignorable="d"
        Title="Визуализация парсеров" Height="450" Width="800" AllowDrop="true">

    <Window.Resources>
        <RoutedUICommand x:Key="TogglePauseCommand"/>
        <RoutedUICommand x:Key="NextFrameCommand"/>
        <RoutedUICommand x:Key="PrevFrameCommand"/>
```

```
        <RoutedUICommand x:Key="FirstFrameCommand"/>
        <RoutedUICommand x:Key="LastFrameCommand"/>
        <RoutedUICommand x:Key="ReverseCommand"/>
        <RoutedUICommand x:Key="SaveCommand"/>
        <RoutedUICommand x:Key="LoadCommand"/>
        <RoutedUICommand x:Key="NextTutorialCommand"/>
        <RoutedUICommand x:Key="PrevTutorialCommand"/>

        <Style x:Key="commonButtonStyle" TargetType="{x:Type Control}">
            <Setter Property="Margin" Value="7,0,0,0"/>
            <Setter Property="FontFamily" Value="Segoe UI Symbol"/>
            <Setter Property="Height" Value="20"/>
            <Setter Property="Width" Value="20"/>
        </Style>
    </Window.Resources>

    <Window.CommandBindings>
        <CommandBinding Command="{StaticResource TogglePauseCommand}"
        Executed="TogglePauseEvent"/>
        <CommandBinding Command="{StaticResource NextFrameCommand}"
        Executed="NextFrameEvent"/>
        <CommandBinding Command="{StaticResource PrevFrameCommand}"
        Executed="PrevFrameEvent"/>
        <CommandBinding Command="{StaticResource FirstFrameCommand}"
        Executed="FirstFrameEvent"/>
        <CommandBinding Command="{StaticResource LastFrameCommand}"
        Executed="LastFrameEvent"/>
        <CommandBinding Command="{StaticResource ReverseCommand}"
        Executed="ReverseEvent"/>
        <CommandBinding Command="{StaticResource SaveCommand}"
        Executed="SaveEvent"/>
        <CommandBinding Command="{StaticResource LoadCommand}"
        Executed="LoadEvent"/>
        <CommandBinding Command="{StaticResource NextTutorialCommand}"
        Executed="NextTutorialEvent"/>
        <CommandBinding Command="{StaticResource PrevTutorialCommand}"
        Executed="PrevTutorialEvent"/>
    </Window.CommandBindings>

    <Window.InputBindings>
        <KeyBinding Key="Space" Command="{StaticResource TogglePauseCommand}"/>
        <KeyBinding Key="N" Command="{StaticResource NextFrameCommand}"/>
        <KeyBinding Key="P" Command="{StaticResource PrevFrameCommand}"/>
        <KeyBinding Key="Home" Command="{StaticResource FirstFrameCommand}"/>
        <KeyBinding Key="End" Command="{StaticResource LastFrameCommand}"/>
        <KeyBinding Key="R" Command="{StaticResource ReverseCommand}"/>
        <KeyBinding Key="S" Command="{StaticResource SaveCommand}"/>
        <KeyBinding Key="S" Modifiers="Ctrl" Command="{StaticResource
        SaveCommand}"/>
```

```
    <KeyBinding Key="O" Command="{StaticResource LoadCommand}"/>
    <KeyBinding Key="O" Modifiers="Ctrl" Command="{StaticResource
    LoadCommand}"/>

    <KeyBinding Key="L" Command="{StaticResource NextFrameCommand}"/>
    <KeyBinding Key="K" Command="{StaticResource TogglePauseCommand}"/>
    <KeyBinding Key="J" Command="{StaticResource PrevFrameCommand}"/>

    <!-- незя тк у нас textbox -->
    <!-- <KeyBinding Key="OemPeriod" Command="{StaticResource
    NextFrameCommand}"/> -->
    <!-- <KeyBinding Key="OemComma" Command="{StaticResource
    PrevFrameCommand}"/> -->
</Window.InputBindings>

<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="120"/>
        <RowDefinition />
        <RowDefinition Height="30"/>
        <RowDefinition Height="20"/>
        <RowDefinition Height="30"/> <!-- 20+30 = 50 -->
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition Width="400"/>
    </Grid.ColumnDefinitions>

    <RichTextBox x:Name="richTextBox" Grid.Column="1" IsReadOnly="True"
    Panel.ZIndex="1"/>
    <emoji:RichTextBox x:Name="tutorialBox" Grid.Column="1" IsReadOnly="True"
    Panel.ZIndex="1" Grid.Row="1" Grid.RowSpan="3"
    VerticalScrollBarVisibility="Auto" BorderBrush="#FFABADB3">
         <emoji:RichTextBox.Resources>
            <Style TargetType="Hyperlink">
                <Setter Property="Cursor" Value="Hand" />
                <EventSetter Event="MouseLeftButtonDown"
                Handler="Hyperlink_MouseLeftButtonDown" />
            </Style>
        </emoji:RichTextBox.Resources>
    </emoji:RichTextBox>

    <Border Grid.Column="1" Grid.Row="4" Panel.ZIndex="2"
    BorderBrush="#ABADB3" BorderThickness="1,0,0,0">
        <DockPanel HorizontalAlignment="Left">
            <DockPanel.Resources>
                <Style TargetType="{x:Type Button}" BasedOn="{StaticResource
                commonButtonStyle}"/>
```

```
            <Style TargetType="{x:Type ToggleButton}"
            BasedOn="{StaticResource commonButtonStyle}"/>
        </DockPanel.Resources>

        <Button x:Name="prevTutorialButton" ToolTip="Предыдущий туториал"
        Content="&#x25c0;" Command="{StaticResource
        PrevTutorialCommand}"/>
        <Button x:Name="nextTutorialButton" ToolTip="Следующий туториал"
        Content="&#x25b6;" Command="{StaticResource
        NextTutorialCommand}"/>
        <ToggleButton x:Name="trimTreeButton" ToolTip="Подровнять дерево"
        Content="&#x0001f334;"/>
        <ToggleButton x:Name="oriTreeButton" ToolTip="Поменять ориентацию
        дерева" Content="&#x0001f332;" RenderTransformOrigin="0.5,0.5">
            <ToggleButton.RenderTransform>
                <RotateTransform Angle="-180"/>
            </ToggleButton.RenderTransform>
        </ToggleButton>
        <ToggleButton x:Name="gravTreeButton" ToolTip="Поменять гравитацию
        дерева" Content="&#x0001f30c;"/>
        <ToggleButton x:Name="helpTreeButton" ToolTip="Режим новичка"
        Content="&#x0001f4d6;"/>
    </DockPanel>
</Border>

<local:TreeCanvas x:Name="canvas" Grid.RowSpan="2">
    <TextBox x:Name="inputBox" Opacity="0" Panel.ZIndex="1"/>
</local:TreeCanvas>

<Slider x:Name="mainSlider" Margin="42,0,10,0" VerticalAlignment="Center"
Grid.Row="3" IsSnapToTickEnabled="True" TickFrequency="1" Minimum="0"
Maximum="1" Grid.RowSpan="2"/>
<Button
  x:Name="playButton" ToolTip="Воспроизведение" Content="&#x25b6;"
  Command="{StaticResource TogglePauseCommand}" Margin="7,0,0,0"
  HorizontalAlignment="Left" Grid.Row="3" VerticalAlignment="Center"
  Height="30" Width="30" FontFamily="Segoe UI Symbol" Grid.RowSpan="2"/>

<DockPanel Grid.Row="2" VerticalAlignment="Center">
    <DockPanel.Resources>
        <Style TargetType="{x:Type ToggleButton}" BasedOn="{StaticResource
        commonButtonStyle}"/>
        <Style TargetType="{x:Type Button}" BasedOn="{StaticResource
        commonButtonStyle}"/>
    </DockPanel.Resources>
```

```
            <!-- &#x0001f4be;&#x0001f5ab;
            &#x0001f331;&#x0001f332;&#x0001f333;&#x0001f334;
            &#x0001f335;&#x0001f33e;&#x0001f33f;&#x2618;&#x0001f340; &#x0001f4c2;
            -->
            <Button x:Name="loadButton" ToolTip="Загрузить сохранение"
            Content="&#x0001f4c2;" Command="{StaticResource LoadCommand}"/>
            <Button x:Name="saveButton" ToolTip="Сохранить" Content="&#x0001f4be;"
            Command="{StaticResource SaveCommand}"/>
            <ToggleButton x:Name="reverseButton" ToolTip="Прокрутка назад"
            Content="&#x25c0;" Command="{StaticResource ReverseCommand}"/>
            <Button x:Name="firstButton" ToolTip="Первый кадр" Content="&#x23ee;"
            Command="{StaticResource FirstFrameCommand}"/>
            <Button x:Name="prevButton" ToolTip="Предыдущий кадр"
            Content="&#x23ea;" Command="{StaticResource PrevFrameCommand}"/>
            <Button x:Name="nextButton" ToolTip="Следующий кадр"
            Content="&#x23e9;" Command="{StaticResource NextFrameCommand}"/>
            <Button x:Name="lastButton" ToolTip="Последний кадр"
            Content="&#x23ed;" Command="{StaticResource LastFrameCommand}"/>

            <TextBlock Text="Скорость:" ToolTip="(кадров в секунду)"
            Margin="7,0,0,0" VerticalAlignment="Center"/>
            <TextBox x:Name="speedBox" Width="40" Margin="7,0,0,0"/>
            <Slider x:Name="speedSlider" Maximum="60" Margin="7,0,10,0"
            VerticalAlignment="Center"/>
        </DockPanel>
    </Grid>
</Window>
```

1.6 MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Threading;
using Microsoft.Win32;
using Newtonsoft.Json;
using ParserLib;

namespace ParserApp {
```

```
/// <summary>
/// Interaction logic for MainWindow.xaml
/// </summary>
[JsonObject(MemberSerialization.OptIn)]
public partial class MainWindow : Window {
    #region json properties
    [JsonProperty("historyIndex")]
    private int historyIndex = 0;
    [JsonProperty("isPaused")]
    private bool isPaused = true;
    [JsonProperty("isReversed")]
    private bool isReversed = false;
    [JsonProperty("colors")]
    private List<Brush> colors;

    [JsonProperty("speed")]
    private double speed {
        get => speedSlider.Value;
        set => SetSpeed(value);
    }
    [JsonProperty("treeTrim")]
    private bool treeTrim {
        get => (bool)trimTreeButton.IsChecked;
        set => trimTreeButton.IsChecked = value;
    }
    [JsonProperty("treeOrientation")]
    private bool treeOrientation {
        get => (bool)oriTreeButton.IsChecked;
        set => oriTreeButton.IsChecked = value;
    }
    [JsonProperty("treeGravity")]
    private bool treeGravity {
        get => (bool)gravTreeButton.IsChecked;
        set => gravTreeButton.IsChecked = value;
    }
    [JsonProperty("treeHelp")]
    private bool treeHelp {
        get => (bool)helpTreeButton.IsChecked;
        set => helpTreeButton.IsChecked = value;
    }

    [JsonProperty("inputString")]
    private string inputString {
        get => theHistory.InputString;
        set => RunParser(value);
    }
    #endregion

    private int tutorialIndex;
```

```
private DispatcherTimer mainTimer = new DispatcherTimer();
private DispatcherTimer speedBoxTimer = new DispatcherTimer();

private Parser parser = new Parser("simple");
private ParserHistory theHistory;
private const string autosavePath = "./autosave.json"; // todo

public MainWindow() {
    InitializeComponent();
    InitializeEvents();

    speed = 4;
    SelectTutorialPage(0);
}

private void InitializeEvents() {
    Drop += MainWindow_Drop;
    mainSlider.ValueChanged += MainSlider_ValueChanged;
    speedSlider.ValueChanged += SpeedSlider_ValueChanged;

    speedBoxTimer.Interval = TimeSpan.FromSeconds(2);
    speedBoxTimer.Tick += SpeedBox_ValueChanged;

    speedBox.LostFocus += SpeedBox_ValueChanged;
    speedBox.TextChanged += (o, e) => {
        speedBoxTimer.Stop();
        speedBoxTimer.Start();

        var r = speed;
        double.TryParse(
            speedBox.Text.Replace(',', '.'),
            NumberStyles.Any,
            CultureInfo.InvariantCulture,
            out r
        );
        if (r < 0) r = 0;
        if (r > 60) r = 60;
        if (r.ToString("0.###", CultureInfo.InvariantCulture) ==
        speedBox.Text) {
            SpeedBox_ValueChanged(o, e);
        }
    };
    speedBox.KeyDown += (o, e) => {
        if (e.Key == Key.Return || e.Key == Key.Escape) {
            SpeedBox_ValueChanged(o, e);
        }
    };
```

```
        inputBox.LostKeyboardFocus += InputBox_ValueChanged;
        inputBox.KeyDown += (o, e) => {
            if (e.Key == Key.Return || e.Key == Key.Escape) {
                InputBox_ValueChanged(o, e);
                Keyboard.ClearFocus();
            }
        };
        inputBox.GotKeyboardFocus += (o, e) => { inputBox.Opacity = 1; };

        mainTimer.Tick += (ob, ea) => { if (!isPaused) NextFrame(); };
        mainTimer.Start();

        RoutedEventHandler t = (ob, ea) => DisplayHistoryEntry();
        trimTreeButton.Click += t;
        oriTreeButton.Click += t;
        gravTreeButton.Click += t;
        helpTreeButton.Click += t;
    }

    private void SelectTutorialPage(int i) {
        try {
            if (!File.Exists($"./tutorials/{i}.rtf")) return;
            using (var fs = File.OpenRead($"./tutorials/{i}.rtf")) {
                tutorialBox.SelectAll();
                tutorialBox.Selection.Load(fs, DataFormats.Rtf);
            }
            if (File.Exists($"./tutorials/{i}.json"))
            Load($"./tutorials/{i}.json");

            tutorialIndex = i;
            prevTutorialButton.IsEnabled = tutorialIndex != 0;
            nextTutorialButton.IsEnabled = File.Exists($"./tutorials/{i +
            1}.rtf");
        } catch (Exception) {
            MessageBox.Show(
                "Что-то пошло не так, и у нас не получилось загрузить
                туториал",
                "Туториал",
                MessageBoxButton.OK,
                MessageBoxImage.Error
            );
        }
    }

    // костыль потомучто richTextBox.Rtf = str; неработает на wpf
    // тоесть "он менят содержимое richTextBox-а на то что написано в строке
    document"
    private void SetRtf(string document) {
        var documentBytes = Encoding.UTF8.GetBytes(document);
```

```
        using (var reader = new MemoryStream(documentBytes)) {
            reader.Position = 0;
            richTextBox.SelectAll();
            richTextBox.Selection.Load(reader, DataFormats.Rtf);
        }
    }

    private void DisplayHistoryEntry() {
        var entry = theHistory[historyIndex];
        mainSlider.ToolTip = historyIndex.ToString();
        mainSlider.Value = historyIndex;
        entry.SetSettings(
            treeTrim,
            treeOrientation,
            treeGravity
        );
        SetRtf(entry.RtfGrammar);

        canvas.DisplayHistoryEntry(entry, treeHelp);
    }

    private void CanvasLegend(bool drawText = false) {
        if (colors == null) return; // just in case
        canvas.InitLegend(colors, theHistory.RuleNames, drawText);
        DisplayHistoryEntry();
    }

    private void RunParser(string input) {
        double historyProgress = 0;
        if (theHistory != null) {
            historyProgress = historyIndex / (double)(theHistory.Count() - 1);
        }

        inputBox.Text = input;
        theHistory = parser.Run(input);

        historyIndex = (int)(historyProgress * (theHistory.Count() - 1));

        mainSlider.Maximum = theHistory.Count() - 1;
        canvas.WriteString(input);
        CanvasLegend();
    }

    private void SetSpeed(double newValue) {
        if (newValue < 0) newValue = 0;
        if (newValue > 60) newValue = 60;

        speedBox.Text = newValue.ToString("0.###",
        CultureInfo.InvariantCulture);
```

```
        speedSlider.Value = newValue;

        var t = 1 / newValue;
        var max = int.MaxValue / 1e7;
        if (t > max || t <= 0) t = max;
        mainTimer.Interval = TimeSpan.FromSeconds(t);
    }

    private void Load(string path = autosavePath) {
        var colorBackup = colors;
        colors = null;
        try {
            var str = File.ReadAllText(path);
            JsonConvert.PopulateObject(str, this);
            if (colors == null) colors = colorBackup;
            reverseButton.IsChecked = isReversed;
            playButton.Content = isPaused ? "\u25b6" : "\u23f8";
            playButton.ToolTip = isPaused ? "Воспроизведение" : "Пауза";

            // это надо если у нас в сэйве нету inputString
            CanvasLegend();
        } catch (Exception) {
            if (path == autosavePath) return;
            MessageBox.Show(
                "Что-то пошло не так, и у нас не получилось загрузить файл",
                "Загрузка",
                MessageBoxButton.OK,
                MessageBoxImage.Error
            );
        } finally {
            if (colors == null) colors = colorBackup;
        }
    }

    private void Save(string path = autosavePath) {
        try {
            if (path.EndsWith(".png")) {
                CanvasLegend(true);
                ExportToPng(path, canvas);
                CanvasLegend(false);
            } else if (path.EndsWith(".svg")) {
                throw new NotImplementedException();
            } else if (path.EndsWith(".xaml")) {
                File.WriteAllText(path, XamlWriter.Save(canvas));
            } else {
                var str = JsonConvert.SerializeObject(this,
                Formatting.Indented);
                File.WriteAllText(path, str);
            }
```

```
    } catch (Exception) {
        if (path == autosavePath) return;
        MessageBox.Show(
            "Что-то пошло не так, и у нас не получилось сохранить файл",
            "Сохранение",
            MessageBoxButton.OK,
            MessageBoxImage.Error
        );
    }
}

#region events
private void MainSlider_ValueChanged(object o, EventArgs e) {
    historyIndex = (int)mainSlider.Value;
    DisplayHistoryEntry();
}

private void SpeedSlider_ValueChanged(object o, EventArgs e) {
    SetSpeed(speedSlider.Value);
}

private void SpeedBox_ValueChanged(object o, EventArgs e) {
    speedBoxTimer.Stop();
    // есть ли какойто менее костыльный способ парсить оба стиля дабла?
    var r = speed;
    double.TryParse(
        speedBox.Text.Replace(',', '.'),
        NumberStyles.Any,
        CultureInfo.InvariantCulture,
        out r
    );
    SetSpeed(r);
}

private void InputBox_ValueChanged(object o, EventArgs e) {
    inputBox.Opacity = 0;
    inputString = inputBox.Text;
}

private void MainWindow_Drop(object o, DragEventArgs e) {
    if (!e.Data.GetDataPresent(DataFormats.FileDrop)) return;
    var files = (string[])e.Data.GetData(DataFormats.FileDrop);
    Load(files[0]);
}

private void Hyperlink_MouseLeftButtonDown(object o, EventArgs e) {
    var hyperlink = (Hyperlink)o;
    Process.Start(hyperlink.NavigateUri.ToString());
}
```

```
private void NextFrame() {
    historyIndex += isReversed ? -1 : 1;

    if (historyIndex >= theHistory.Count() || historyIndex < 0) {
        historyIndex -= isReversed ? -1 : 1;
        if (!isPaused) TogglePause();
        return;
    }

    DisplayHistoryEntry();
}

private void PrevFrame() {
    isReversed = !isReversed;
    NextFrame();
    isReversed = !isReversed;
}

private void FirstFrame() {
    historyIndex = !isReversed ? 0 : theHistory.Count() - 1;
    DisplayHistoryEntry();
}

private void LastFrame() {
    historyIndex = isReversed ? 0 : theHistory.Count() - 1;
    DisplayHistoryEntry();
}

private void TogglePause() {
    isPaused = !isPaused;
    playButton.Content = isPaused ? "\u25b6" : "\u23f8";
    playButton.ToolTip = isPaused ? "Воспроизведение" : "Пауза";
}

private void Reverse() {
    isReversed = !isReversed;
    reverseButton.IsChecked = isReversed;
    if (isPaused && isReversed) TogglePause();
}

private void NextFrameEvent(object o, EventArgs e) {
    NextFrame();
}

private void PrevFrameEvent(object o, EventArgs e) {
    PrevFrame();
}
```

```
private void FirstFrameEvent(object o, EventArgs e) {
    FirstFrame();
}

private void LastFrameEvent(object o, EventArgs e) {
    LastFrame();
}

private void TogglePauseEvent(object o, EventArgs e) {
    TogglePause();
}

private void ReverseEvent(object o, EventArgs e) {
    Reverse();
}

private void SaveEvent(object o, EventArgs e) {
    var dia = new SaveFileDialog();
    dia.Filter = "Сохранить текущее состояние (*.json)|*.json|Векторный
    рисунок дерева (*.xaml)|*.xaml|Растровый рисунок дерева
    (*.png)|*.png";
    dia.DefaultExt = "json";
    dia.FileName = "tree.json";
    dia.InitialDirectory =
    Environment.GetFolderPath(Environment.SpecialFolder.Desktop);

    var t = isPaused;
    isPaused = true;
    var rt = dia.ShowDialog();
    isPaused = t;
    if (rt != true) return; // так надо: !tr неработает

    Save(dia.FileName);
}

private void LoadEvent(object o, EventArgs e) {
    var dia = new OpenFileDialog();
    dia.Filter = "Загрузить текущее состояние (*.json)|*.json";
    dia.DefaultExt = "json";
    dia.FileName = "tree.json";
    dia.InitialDirectory =
    Environment.GetFolderPath(Environment.SpecialFolder.Desktop);

    var t = isPaused;
    isPaused = true;
    var rt = dia.ShowDialog();
    isPaused = t;
    if (rt != true) return; // так надо: !tr неработает
```

```
            Load(dia.FileName);
        }

        private void NextTutorialEvent(object o, EventArgs e) {
            SelectTutorialPage(tutorialIndex + 1);
        }

        private void PrevTutorialEvent(object o, EventArgs e) {
            SelectTutorialPage(tutorialIndex - 1);
        }

        #endregion

        static private void ExportToPng(string path, FrameworkElement element) {
            if (path == null) return;

            // Save current canvas transform
            var transform = element.LayoutTransform;
            // reset current transform (in case it is scaled or rotated)
            element.LayoutTransform = null;

            // Get the size of canvas
            var size = new Size(element.ActualWidth, element.ActualHeight);
            // Measure and arrange the surface
            // VERY IMPORTANT
            element.Measure(size);
            element.Arrange(new Rect(size));

            // Create a render bitmap and push the surface to it
            var renderBitmap = new RenderTargetBitmap((int)size.Width,
            (int)size.Height, 96d, 96d, PixelFormats.Pbgra32);
            renderBitmap.Render(element);

            // Create a file stream for saving image
            using (var outStream = new FileStream(path, FileMode.Create)) {
                // Use png encoder for our data
                var encoder = new PngBitmapEncoder();
                // push the rendered bitmap to it
                encoder.Frames.Add(BitmapFrame.Create(renderBitmap));
                // save the data to the stream
                encoder.Save(outStream);
            }

            // Restore previously saved layout
            element.LayoutTransform = transform;
        }
    }
}
```

1.7 TreeCanvas.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Shapes;
using ParserLib;

namespace ParserApp {
    internal class TreeCanvas : Canvas {
        private readonly FontFamily font = new FontFamily("Consolas");
        private Dictionary<string, Brush> colorDict;
        private TextBox inputBox;
        private HistoryEntry lastHistoryEntry;
        private bool lastHelp;
        private const int CharWidth = 20;
        private const int TextStart = 15;
        private const int FontSize = 26;
        private const int LegendFontSize = 12;
        private const int TextblockTop = 10;
        private const int TreeTop = 50;
        private const int TreeVSpace = 15;
        private const int RectHeight = 10;

        public void WriteString(string text) {
            var pos = TextStart;

            // удаляем весь старый текст (если он есть)
            foreach (var tb in Children.OfType<TextBlock>().ToList()) {
                Children.Remove(tb);
            }

            foreach (var chr in text) {
                var txt = new TextBlock();
                txt.FontSize = FontSize;
                txt.Text = chr.ToString();
                txt.FontFamily = font;
                SetTop(txt, TextblockTop);
                SetLeft(txt, pos);
                Children.Add(txt);
                pos += CharWidth;
            }

            if (inputBox == null) {
                inputBox = Children.OfType<TextBox>().First();
            }
            SetTop(inputBox, TextblockTop);
```

```
        SetLeft(inputBox, TextStart);
        inputBox.Width = pos - TextStart;
        inputBox.Height = FontSize;
        inputBox.FontSize = FontSize - 8;
    }

    public void InitLegend(
        List<Brush> colors,
        IEnumerable<string> names,
        bool drawText = false
    ) {
        // удаляем все старые кружочки
        foreach (var el in Children.OfType<Ellipse>().ToList()) {
            Children.Remove(el);
        }

        // удаляем все старые подписи
        foreach (var tb in Children.OfType<TextBlock>().ToList()) {
            if (tb.FontSize == LegendFontSize) Children.Remove(tb);
        }

        colorDict = new Dictionary<string, Brush>();

        var i = 0;
        var pos = 5;
        var originalColorsLength = colors.Count;
        foreach (var ruleName in names) {
            if (i == colors.Count) colors.Add(colors[i %
            originalColorsLength]);
            var value = colors[i];
            colorDict[ruleName] = value;

            var el = new Ellipse();
            el.Width = el.Height = LegendFontSize;
            el.Fill = value;
            el.ToolTip = ruleName;
            SetTop(el, pos);
            SetRight(el, 5);
            Children.Add(el);

            // изменение цветов при клике на Ellipse
            var currentIndex = i;
            el.MouseDown += (o, e) => {
                var c = GetColor();
                if (c != null) colors[currentIndex] = c;
                InitLegend(colors, names);
                DisplayHistoryEntry(lastHistoryEntry, lastHelp);
            };
```

```
            if (drawText) {
                var txt = new TextBlock();
                txt.FontSize = LegendFontSize;
                txt.Text = ruleName;
                txt.FontFamily = font;
                SetTop(txt, pos);
                SetRight(txt, 20);
                Children.Add(txt);
            }

            pos += 19;
            i++;
        }
    }

    private void DrawRect(HistoryToken tok, int pos) {
        var rect = new Border();
        rect.CornerRadius = new CornerRadius(5, 5, 5, 5);
        var end = tok.EndPos;
        if (end == -1) {
            end = pos;
            rect.CornerRadius = new CornerRadius(5, 0, 0, 5);
        }

        rect.Background = colorDict[tok.Name];
        rect.Height = RectHeight;
        rect.Width = (end - tok.StartPos) * CharWidth;
        rect.ToolTip = tok.Name;

        if (tok.Trimmable) rect.Opacity = .5;

        SetTop(rect, TreeTop + tok.DisplayLevel * TreeVSpace);
        SetLeft(rect, TextStart + tok.StartPos * CharWidth);
        Children.Add(rect);
    }

    public void DisplayHistoryEntry(HistoryEntry entry, bool help) {
        lastHistoryEntry = entry;
        lastHelp = help;

        // удаляем строе дерево
        foreach (var tb in Children.OfType<Border>().ToList()) {
            Children.Remove(tb);
        }

        foreach (var tok in entry) {
            DrawRect(tok, entry.CursorPos);
        }
```

```
            // удаляем стрые линии
            foreach (var line in Children.OfType<Line>().ToList()) {
                Children.Remove(line);
            }

            if (help) DrawConventionalTree(entry.GetEdges());
        }

        private void DrawConventionalTree(Dictionary<HistoryToken, HistoryToken>
        edges) {
            foreach (var edge in edges) {
                var line = new Line();
                line.X1 = GetNodeCenterX(edge.Key);
                line.Y1 = GetNodeCenterY(edge.Key);
                line.X2 = GetNodeCenterX(edge.Value);
                line.Y2 = GetNodeCenterY(edge.Value);
                line.Stroke = Brushes.Orange;
                line.StrokeThickness = 2;
                Children.Add(line);
            }
        }

        private double GetNodeCenterX(HistoryToken node) {
            var end = node.EndPos;
            if (end == -1) {
                end = lastHistoryEntry.CursorPos;
            }

            return TextStart + node.StartPos * CharWidth + (end - node.StartPos) *
            CharWidth / 2.0;
        }

        private double GetNodeCenterY(HistoryToken node) {
            return TreeTop + node.DisplayLevel * TreeVSpace + RectHeight / 2.0;
        }


        static private Brush GetColor() {
            var dia = new System.Windows.Forms.ColorDialog();
            if (dia.ShowDialog() == System.Windows.Forms.DialogResult.OK) {
                var c = dia.Color;
                return new SolidColorBrush(Color.FromArgb(c.A, c.R, c.G, c.B));
            }
            return null;
        }
    }
}
```

2 ParserLib

2.1 ParserLib.csproj

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net45</TargetFramework>
    <OutputType>Library</OutputType>
    <GenerateAssemblyInfo>false</GenerateAssemblyInfo>
  </PropertyGroup>

  <ItemGroup>
    <Content Include="parsers\*.*">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
    </Content>
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="System.ValueTuple" Version="4.5.0" />
  </ItemGroup>

</Project>
```

2.2 HistoryEntry.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

namespace ParserLib {
    public class HistoryEntry : IEnumerable<HistoryToken> {
        private HistoryToken[] treeRanges;
        public string RtfGrammar { get; }
        public int CursorPos { get; }

        private bool isTrimmed = false;
        private Dictionary<HistoryToken, HistoryToken> edges;

        internal HistoryEntry(HistoryToken[] ranges, string rtf) {
            RtfGrammar = rtf;
            treeRanges = ranges;
            CursorPos = treeRanges.Max(e => Math.Max(e.StartPos, e.EndPos - 1)) +
            1;
        }

        public void SetSettings(bool trim, bool orientation, bool gravity) {
            isTrimmed = trim;
```

```
    CalculateDisplayLevels(orientation ^ gravity);
    if (gravity) InvertDisplayLevels();
    edges = null;
}

public Dictionary<HistoryToken, HistoryToken> GetEdges() {
    if (this.edges != null) return this.edges;
    var edges = new Dictionary<HistoryToken, HistoryToken>();
    var stacks = new HistoryToken[CursorPos];
    foreach (var tok in this.OrderBy(e => -e.RecLevel)) {
        var end = tok.EndPos;
        if (end == -1) end = CursorPos;

        for (var i = tok.StartPos; i < end; i++) {
            if (stacks[i] != null) {
                edges[stacks[i]] = tok;
            }
            stacks[i] = tok;
        }
    }
    this.edges = edges;
    return edges;
}

private void InvertDisplayLevels() {
    var maxDisplayLevel = this.Max(e => e.DisplayLevel);
    foreach (var tok in this) {
        tok.DisplayLevel = maxDisplayLevel - tok.DisplayLevel;
    }
}

private void CalculateDisplayLevels(bool orientation = false) {
    var recLvs = new int[CursorPos];

    IEnumerable<HistoryToken> t = this.OrderBy(e => -e.RecLevel);
    if (orientation) t = t.Reverse();
    foreach (var tok in t) {
        var end = tok.EndPos;
        if (end == -1) end = CursorPos;

        var slice = new ArraySegment<int>(recLvs, tok.StartPos, end -
        tok.StartPos);
        tok.DisplayLevel = slice.Max();

        for (var i = tok.StartPos; i < end; i++) {
            recLvs[i] = tok.DisplayLevel + 1;
        }
    }
}
```

```
        public override string ToString() {
            return string.Join(" ", (object[])treeRanges);
        }

        public IEnumerator<HistoryToken> GetEnumerator() {
            var t = treeRanges.Where(e => !e.Name.StartsWith("\""));
            if (isTrimmed) t = t.Where(e => !e.Trimmable);
            return t.GetEnumerator();
        }

        IEnumerator IEnumerable.GetEnumerator() {
            return GetEnumerator();
        }
    }
}
```

2.3 HistoryToken.cs

```
namespace ParserLib {
    public class HistoryToken {
        public string Name { get; }
        public int StartPos { get; }
        public int EndPos { get; }

        internal int RecLevel { get; }
        public bool Trimmable { get; }
        public int DisplayLevel { get; internal set; }

        internal HistoryToken(ParserTreeToken tok) {
            Name = tok.Name;
            StartPos = tok.StartPos;
            EndPos = tok.EndPos;
            RecLevel = tok.RecLevel;
            DisplayLevel = RecLevel;
            Trimmable = tok.ChildCount == 1 && tok.EndPos >= 0;
        }

        public override string ToString() {
            if (EndPos == -1) return $"{StartPos}:-({Name}, {RecLevel})";
            return $"{StartPos}:{EndPos}({Name}, {RecLevel})";
        }
    }
}
```

2.4 Parser.cs

```
using System.Diagnostics;
using System.IO;
```

```
namespace ParserLib {
    public class Parser {
        private string name;
        public Parser(string name) {
            this.name = name;
        }

        public ParserHistory Run(string input) {
            var rtf = File.ReadAllText("parsers/" + name + ".rtf");
            var tree = new ParserHistory(rtf, input);

            var process = new Process();
            process.StartInfo.CreateNoWindow = true;
            process.StartInfo.UseShellExecute = false;
            process.StartInfo.RedirectStandardInput = true;
            process.StartInfo.RedirectStandardOutput = true;
            process.OutputDataReceived += (sender, args) => {
                tree.Add(args.Data);
            };
            process.StartInfo.FileName = "parsers/" + name + ".exe";
            process.Start();
            process.BeginOutputReadLine();

            process.StandardInput.Write(input);
            process.StandardInput.Close();

            process.WaitForExit();
            return tree;
        }
    }
}
```

2.5 ParserHistory.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

namespace ParserLib {
    public class ParserHistory : IEnumerable<HistoryEntry> {
        private Stack<ParserTreeToken> stack = new Stack<ParserTreeToken>();
        private List<ParserTreeToken> state = new List<ParserTreeToken>();
        private int prevPos = -1;

        /// <summary>
        /// Inherited parser parameters.
        /// </summary>
```

```
public string OriginalRtf { get; }
public string InputString { get; }

public IEnumerable<string> RuleNames => RtfBuilder.GetNames(OriginalRtf);

private List<HistoryEntry> history = new List<HistoryEntry>();

private HistoryToken[] CopyState() {
    return state.Select(e => e.Clone()).ToArray();
}

internal ParserHistory(string rtf, string input) {
    OriginalRtf = rtf;
    InputString = input;
}

private void SaveState() {
    var tokens = CopyState();
    var rtf = RtfBuilder.Build(OriginalRtf, stack);
    var r = new HistoryEntry(tokens, rtf);
    history.Add(r);
}

public void Add(string line) {
    if (line == null) return;
    line = line.Trim();
    if (line == "") return;

    if (line.StartsWith("eval failed: SyntaxError:")) {
        // todo
        return;
    }

    var words = line.Split(new string[] { " " },
    StringSplitOptions.RemoveEmptyEntries);
    var pos = int.Parse(words[0].Split(':').Last()) - 1;

    var hasFailed = prevPos > pos;
    while (prevPos > pos) {
        var t = state.Last();
        if (t.EndPos == -1) break;
        t.Parent.ChildCount--;
        state.RemoveAt(state.Count - 1);
        prevPos = t.EndPos;
    }
    prevPos = pos;

    if (hasFailed) SaveState();
```

```
            if (words[1] == "rule.enter") {
                var val = 0;
                ParserTreeToken parent = null;
                if (stack.Count != 0) {
                    parent = stack.Peek();
                    parent.ChildCount++;
                    var dict = stack.Peek().Dict;
                    dict.TryGetValue(words[2], out val);
                    dict[words[2]] = val + 1;
                }
                var t = new ParserTreeToken(parent, words[2], val, pos,
                stack.Count);
                state.Add(t);
                stack.Push(t);
            } else {
                var t = stack.Pop();
                t.EndPos = pos;
                if (words[1] != "rule.match") {
                    t.EndPos = -2;
                    t.Parent.ChildCount--;
                    var i = state.IndexOf(t);
                    state.RemoveRange(i, state.Count - i);
                }
            }

            SaveState();
        }

        public IEnumerator<HistoryEntry> GetEnumerator() =>
        history.GetEnumerator();
        IEnumerator IEnumerable.GetEnumerator() => history.GetEnumerator();
        public HistoryEntry this[int i] => history[i];
    }
}
```

2.6 ParserTreeToken.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace ParserLib {
    internal class ParserTreeToken {
        public ParserTreeToken Parent { get; }
        public string Name { get; }
        public int Index { get; }
        public int StartPos { get; }
        public int RecLevel { get; }
        public int EndPos { get; set; }
```

```
        public int ChildCount { get; set; }
        public Dictionary<string, int> Dict { get; }

        public ParserTreeToken(ParserTreeToken parent, string name, int index, int
        startPos, int recLevel) {
            if (name.StartsWith("_") && name.EndsWith("_") && name.Length > 1) {
                var chars = Enumerable.Range(0, name.Length / 2 - 1)
                    .Select(i => (char)Convert.ToUInt16(name.Substring(i * 2 + 1,
                    2), 16));
                name = '"' + string.Join("", chars) + '"';
            }
            Parent = parent;
            Name = name;
            Index = index;
            StartPos = startPos;
            RecLevel = recLevel;
            EndPos = -1;
            Dict = new Dictionary<string, int>();
        }

        public override string ToString() {
            return $"{StartPos}:{EndPos}({Name}, {Index})";
        }

        public HistoryToken Clone() {
            return new HistoryToken(this);
        }
    }
}
```

2.7 RtfBuilder.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;

namespace ParserLib {
    internal class RtfBuilder {
        private string[] lines;

        private RtfBuilder(string rtf) {
            lines = rtf.Split('\n'); //.Where(e=>e.StartsWith("\\cf")).ToArray();
        }

        private void HighlightIdentifier(ParserTreeToken t) {
            if (t.Parent == null) return;
            var lineIndex = Array.FindIndex(lines, e => e.StartsWith("\\cf2 " +
            t.Parent.Name));
```

```
            var line = lines[lineIndex];

            var i = 0;
            var regex = @"\\b\{\}(\\cf[0-9] )" + Regex.Escape(t.Name);
            line = Regex.Replace(line, regex, m => {
                if (i++ == t.Index) return @"\b " + m.Groups[1].Value + t.Name;
                return m.Groups[1].Value + t.Name;
            });
            line = line.Replace("\\b{}", "");

            lines[lineIndex] = line;
        }

        private string End() {
            return string.Join("\n", lines).Replace("{}", "0");
        }

        public static string Build(string rtf, IEnumerable<ParserTreeToken>
        tokens) {
            var builder = new RtfBuilder(rtf);
            foreach (var tok in tokens) {
                builder.HighlightIdentifier(tok);
            }
            return builder.End();
        }

        public static IEnumerable<string> GetNames(string rtf) {
            return rtf.Split('\n')
                .Where(e => e.StartsWith("\\cf"))
                .Select(e => e.Split(new string[] { "\\cf4" },
                StringSplitOptions.None)[0].Split(' ')[1]);
        }
    }
}
```

1 Лист регистрации изменений

Изм.	Номера листов				Всего листов в документе	№ документа	Входящий № сопроводит. докум. и дата	Подпись	Дата
	изменен-ных	заменен-ных	новых	аннул-лирован-ных					