Sklep MTG (Magic: The Gathering) Dokumentacja

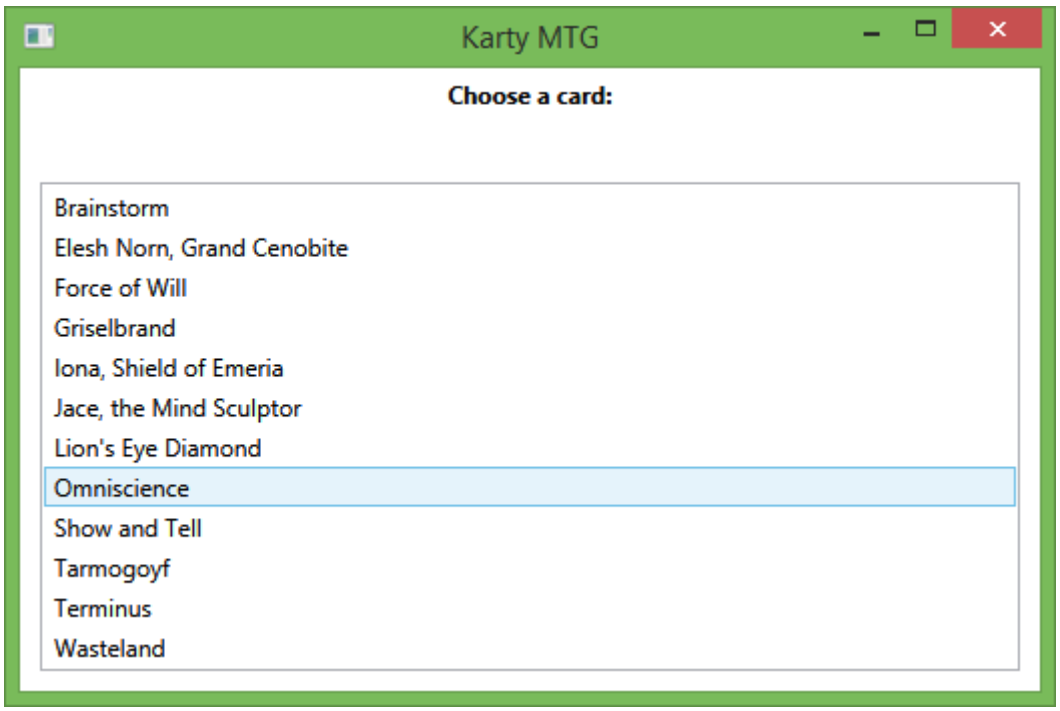
Autor:	Wersja:
Oskar Jończyk	1.0 (2017-02-06)

Aplikacja

Aplikacja została napisana za pomocą WPF (C#) jako źródło danych wykorzystując przygotowany wcześniej plik XML.
Import danych umożliwia znacznik XMLDataProvider:

```
<Window.Resources>
    <XmlDataProvider x:Key="CardData" Source="karty.xml" XPath="karty"/>
</Window.Resources>
```

1. Menu główne



W menu głównym wyświetlają się nazwy kart (znacznik karty/nazwa). Umieszczone są w elemencie **listbox**. Na górze nad nimi element **label**.

```
<Grid>
    <ListBox x:Name="CardNamesBox" ItemsSource="{Binding Source={StaticResource CardData}, XPath=karta/nazwa}" BorderThickness="1"
             MouseDoubleClick="CardNamesBox_MouseDoubleClick" Margin="10,10,10,10" VerticalAlignment="Bottom"/>
    <Label x:Name="label" Content="Choose a card:" HorizontalAlignment="Center" VerticalAlignment="Top" FontWeight="Bold"/>

</Grid>
```

Za pomocą podwójnego kliknięcia zostaje wywołana funkcja ‘CardNamesBox_MouseDoubleClick’ i następuje otwarcie nowego okna.

```
private void CardNamesBox_MouseDoubleClick(object sender, MouseButtonEventArgs e)
{
    var selectedXml = (XmlElement)CardNamesBox.SelectedItem;
    var parent = selectedXml.ParentNode;
    var selectedText = selectedXml.InnerText;
    new Window1(parent).Show();

}
```

Krótki opis funkcji:

a. Za pomocą '.SelectedItem' przypisujemy do zmiennej wybrany przez kliknięcie element
b. Następnie posiadając wybrany element „nazwa" (w XML karty/karta/nazwa) dostajemy się do jego rodzica elementu karta.
c. selectedText potrzebny był do celów debugowanie teraz jest zbędny
d. wywołujemy nowe okno jako argument przekazując wybraną kartę

2. Okno Karty

Wyświetla zdjęcie karty oraz informacje o niej zawarte w pliku XML. Do poziomego ułożenia tekstu i zdjęcia obok siebie zastosowałem element **StackPanel**.

```
<Grid>
    <StackPanel Height="auto" Width="auto" Orientation="Horizontal">
        <Image x:Name="image1" HorizontalAlignment="Left" Height="Auto" VerticalAlignment="Top" Margin="5,5,5,5"/>
        <Label x:Name="label1" Content="" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Center" FontSize="20"/>
    </StackPanel>
</Grid>
```

Do wyświetlenia obrazka został użyty element image. Warto też wspomnieć o atrybucie Content w Label, który pozostał puste lecz zainicjowany ponieważ będzie potrzebny później.

```
namespace Sklep_MTG
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class Window1 : Window
    {
        public Window1(XmlNode node)
        {
            InitializeComponent();
            for(var i = 0; i < node.ChildNodes.Count - 1; ++i)
            {
                label1.Content += node.ChildNodes[i].Name.First().ToString().ToUpper()+node.ChildNodes[i].Name.Substring(1)+": ";
                if (node.ChildNodes[i].InnerText.Length != 0)
                {
                    label1.Content += node.ChildNodes[i].InnerText + "\n\n";
                } else
                {
                    label1.Content += node.ChildNodes[i].Attributes[0].Value+"\n\n";
                }
            }
            image1.Source = new BitmapImage( new Uri(node.ChildNodes[4].InnerText, UriKind.RelativeOrAbsolute));
        }
    }
}
```

Window1 to klasa automatycznie wygenerowana przez Visual Studio przy tworzeniu nowego okna w Designerze. Konstruktor zwykle składa się z jednej linijki **InitializeComponent()**. Jako, że ma ona wyświetlać inne dane dla każdej karty trochę została przeze mnie zmodyfikowana.

a. Pętla **for** iteruje po wszystkich elementach elementu <karta>(z wyjątkiem ostatniego).
b. Pierwsza linia w pętli for wyczytuje nazwę znacznika, zamienia pierwszą literę na dużą i na końcu dodaje dwukropek
c. Jeśli wartość obecnego elementu nie jest pusta to wypisujemy tą wartość
d. Jeśli jest pusta to odczytujemy wartość pierwszego atrybutu tego znacznika (wynika to ze sposobu w jaki opisujemy karty w pliku XML)
e. Na końcu odczytujemy adres obrazka z pliku XML i tworzymy nowy obiekt **BitmapImage**.

Dane (plik XML)

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE karty [
  <!ELEMENT karty (karta*)>
  <!ELEMENT karta (nazwa,kolor,rarity,cena,url)>
  <!ATTLIST karta id ID #REQUIRED>
  <!ELEMENT nazwa (#PCDATA)>
  <!ELEMENT kolor EMPTY>
  <!ATTLIST kolor kolor (Blue|Red|Green|Black|White|Multicolor|Neutral) #REQUIRED>
  <!ELEMENT rarity EMPTY>
  <!ATTLIST rarity rarity (Common|Uncommon|Rare) #REQUIRED>
  <!ELEMENT cena (#PCDATA)>
  <!ELEMENT url (#PCDATA)>
]>
<karty xmlns="">
  <karta id="brainstorm">
    <nazwa>Brainstorm</nazwa>
    <kolor kolor="Blue"/>
    <rarity rarity="Common"/>
    <cena>3,99 $</cena>
    <url>data/images/brainstorm.jpg</url>
  </karta>
  <karta id="eleshnorn">
    <nazwa>Elesh Norn, Grand Cenobite</nazwa>
    <kolor kolor="White"/>
    <rarity rarity="Rare"/>
    <cena>5,99 $</cena>
    <url>data/images/eleshnorn.jpg</url>
  </karta>
```

Poprawność danych w pliku jest zachowana dzięki zastosowaniu DTD. I tak jak widzimy kolejne co następuje po <!DOCTYPE karty[:

a. Dokument karty może się składać z 0 lub wielu liczby kart
b. **Każda** karta musi zawierać nazwę, kolor, rzadkość, cenę i url, a także atrybut ‘id’
c. Każdy element z #PCDATA zawiera dane wprowadzane przez użytkownika
d. Kolor i rarity **zawsze** jest elementem pustym jednak musi zawierać atrybut o wartości wybranej z listy dostępnych podanych w nawiasach. Ułatwia to dodawanie danych użytkownikowi dzięki podpowiedziom (np. Intellisense w Visual Studio) ale też nie pozwala na wprowadzenie nieprawdziwych danych (np. nieistniejącego w tej grze koloru karty).

Jedynym nieweryfikowalnym elementem jest obrazek. W dokumencie przechowywany jest jego adres. Tutaj trzeba polegać na użytkowniku, że wprowadzi poprawne dane i zapewni odpowiedni obrazek.

Jeśli dokumentacja jest niejasna lub niewystarczająca proszę o kontakt z autorem: **o.jonczyk@gmail.com**.

Autor: Oskar Jończyk