

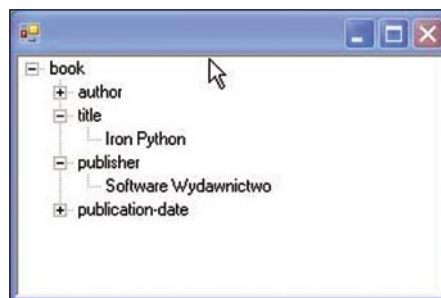
.NET + Python = IronPython

Dlaczego ktoś mógłby chcieć korzystać z Pythona na platformie .NET? Z jakich mechanizmów (narzędziach) dostępnych w świecie .NET może programista Pythona skorzystać? Odpowiedź jest jedna: możliwość wyboru. Wybór języka programowania sprowadza się często do indywidualnych preferencji i cech/typu projektu, nad którym pracujemy.

Python w przedstawianej implementacji jest interesującym rozwiązaniem na platformy .NET i Mono (fanom Javy polecamy Jython) i wydaje się, iż ma wszelkie dane ku temu, aby przyciągnąć do siebie nowych zwolenników. Skryptowy Python, stworzony we wczesnych latach 90-tych przez Guido van Rosuma, jest dziś wykorzystywany praktycznie na każdej powszechnej platformie (*Windows, Mac, Linux/Unix*) od komputerów stacjonarnych po palmtopy i telefony komórkowe (*Nokia*) w takich dziedzinach jak, wizualizacja i gry, networking, aplikacje webowe, desktopowe, inżynieryjne, bazy danych, etc. Jest używany do tworzenia prostych skryptów, jak i dużych aplikacji (*Zope, Plone*). Poniżej postaramy się przedstawić jego implementację na platformę .NET.

IronPython

Twórca IronPythona, Jim Hugunin, jest znany z wcześniejszych (udanych) implementacji Pythona na maszynę wirtualną Javy – Jython. Projekt na platformę .NET powstał około roku 2003 i jest przykładem świetnej i szybkiej implementacji dynamicznego języka skryptowego w środowisku CLR (*platformy .NET i Mono*). Interesujące jest oparcie się przez twórców na środowisku CLR, co w rezultacie zapewnia wykorzystywanie bibliotek .NET w tworzonych skryptach, aplikacjach desktopowych i webowych przy użyciu IronPythona. Co ciekawe, przy zachowaniu pewnych reguł, możliwe jest korzystanie ze standardowych bibliotek CPythona celem zwiększenia funkcjonalności tworzonej aplikacji o potrzebne elementy. Współpracę IronPythona z .NET pokażemy na kilku wybranych przykładach. Zaczniemy od napisania skryptu w Pythonie w rodzaju prostej przeglądarki plików XML z wykorzystaniem kontrolki *TreeView* i formularza *Form*. Tworzenie nowych rozszerzeń (*klas*) w C# dla IronPythona pokażemy na przykładzie podobnej aplikacji, gdzie komponent przeglądarki osadzimy tym razem w wyświetlanym formularzu. Korzystanie z interpretera IronPythona przedstawimy na przykładzie aplikacji – słownika. Na koniec, omówimy prosty przykład zaprze-



Rysunek 1. Odtworzony dokument XML

gnięcia IronPythona do grafiki 3D na przykładzie biblioteki *Irrlicht .NET 3D*. Zakładamy opanowanie przez czytelnika podstaw programowania w Pythonie i .NET. Koniecznie trzeba zwrócić uwagę, iż IronPython i CPython to dwie różne implementacje tego samego języka. Różnic w chwili obecnej jest sporo – od bardzo trywialnych, które sprowadzają się do wyświetlania różnych komunikatów o błędach, po takie, które wynikają z nieobecności takiego czy innego modułu, np. *cmath* lub *os*.

Narzędzia

Standardowo, IronPython (binaria i kody źródłowe do pobrania z *CodePlex*) daje użytkownikowi zestaw podobnych narzędzi programistycznych co inne implementacje tego języka (*CPython*). Do dyspozycji jest konsola interpretera poleceń (*ipy.exe*), a uruchamianie skryptów i poleceń IronPythona odbywa się w znanych nam postaciach: wsadowo lub interaktywnie. W chwili obecnej, jedyne środowisko programistyczne (*IDE*) pozwalające tworzyć/edytować skrypty w tym języku to Visual Studio 2005 koniecznie w wersji Standard lub wyższej. W chwili obecnej brak wsparcia dla IronPythona w wersji Visual Studio Express, choć istnieje ono w Web Developer Express (po zainstalowaniu dodatkowego pakietu integrującego IronPythona z ASP .NET). Do tworzenia przykładów zawartych w niniejszym artykule autorzy korzystali z Visual Studio Express (C#) oraz popularnego IDLE (*Python*).

Skrypty

Omówienie IronPythona rozpoczniemy od skryptu, którego działanie polegać ma na odczytaniu pliku XML, a następnie odtworzeniu jego struktury w kontrolce *TreeView* jak na Rysunku 1. Treść skryptu przedstawiono w Listingu 1. Rozpoczynamy od zapewnienia sobie dostępu do standardowych modułów CPythona. Robimy to przez umieszczenie na początku skryptu następujących wierszy

```
import sys
sys.path.append(r"c:\python24\lib")
```

Autorzy są entuzjastami Pythona we wszystkich wcieleniach (czytaj: implementacjach).

Kontakt: ijmj.goldasz@gmail.com

Oczywiście, importujemy moduł `clr` (CLR), a dostęp do potrzebnych modułów .NET zapewniamy sobie dzięki metodzie `clr.AddReferenceByPartialName(...)`. W naszym skrypcie utworzymy 2 klasy – pierwszą o nazwie `xmlTree` dziedziczącej po klasie `TreeView`, drugą zaś (dziedziczącą po klasie `Form`) nazwiemy `HelloXML`. Klasa `xmlTree` posiada 2 zmienne: `pathToXML` (w której przechowywać będziemy ścieżkę do analizowanego pliku) oraz `root`, która posłuży nam do przechowania treści dokumentu (`XmlDocument`). Odtworzenie struktury dokumentu XML najłatwiej rozwiązać rekurencyjnie – stąd obecność w ciele klasy kolejnych metod. Pierwsza z nich o nazwie `AddNode(...)` porusza się rekurencyjnie po strukturze (drzewiastej) analizowanego dokumentu, dodając odwiedzane węzły `inXmlNode` do kolejnych węzłów `inTreeNode` naszej kontrolki. Druga metoda `PopulateTree()` to sterownik wczytujący żądany dokument do zmiennej `root` i wywołujący rekurencyjną metodę `AddNode(...)`. Na tym za-

danie odtworzenia struktury dokumentu XML się kończy. Chcąc umieścić (wyświetlić) naszą kontrolkę w formularzu, w konstruktorze klasy `HelloXML` tworzymy instancję klasy `xmlTree` i wywołujemy metodę `PopulateTree()`, Teraz wystarczy tylko dodać ją do formularza i w funkcji `Main` umieścić znaną skądinąd

```
Application.Run (HelloXML (nazwa_pliku_XML))
```

i efekt jest widoczny jak przedstawionej ilustracji w Rysunku 1. Chcąc sprowokować pojawienie się wyjątku, wywołamy skrypt z nazwą nieistniejącego dokumentu – zob. Rysunek 2.

Tworzenie rozszerzeń

Tworzenie rozszerzeń .NET dla IronPythona pokażemy na identycznym przykładzie jak poprzednio – zob. Listing 2. Nasze zadanie polega na osiągnięciu identycznej funkcjonalności jak

Listing 1. Pierwszy przykład - xmlTree

```
import sys
# Dodajemy dostep do standardowych moduLOW Pythona
sys.path.append(r"c:\python24\lib")
# Importujemy Common Language Runtime ...
import clr
# Formularze, kontrolki
clr.AddReferenceByPartialName("System.Windows.Forms")
clr.AddReferenceByPartialName("System.Drawing")
# XML
clr.AddReferenceByPartialName("System.Xml")
# ...oraz inne potrzebne moduLY
import System
from System.Windows.Forms import *
from System.Drawing import *
from System.Xml import *
# Tworzymy kontrolke przegladarki
# - dziedziczy po klasie TreeView
class xmlTree(TreeView):
    def __init__(self): # Domyslny bezparametrowy konstruktor
        self.Nodes.Clear()
        self.pathToXml = '' # Sciezka do pliku XML
        self.root = None
    # W metodzie PopulateTree generujemy structure pliku XML
    def PopulateTree(self):
        self.root = XmlDocument() # korzen
        try: # obsluga wyjatkow
            self.root.Load(self.pathToXml) # Otwieramy plik
            # Kasujemy wszystkie istniejace wezly
            self.Nodes.Clear()
            # Dodajemy pierwszy wezel (korzen) do drzewa
            self.Nodes.Add(TreeNode(self.root.
                DocumentElement.Name))
            tNode = TreeNode()
            tNode = self.Nodes[0] # Wskazanie na korzen
            # i rekurencyjnie zapelniamy cale drzewo
            self.AddNode(self.root.DocumentElement, tNode)
        except Exception, detail:
            # Komunikujemy blad
            MessageBox.Show(System.Convert.ToString(detail))
            self.Nodes.Clear()

        self.Nodes.Add(TreeNode(System.Convert.ToString(
            detail)))
    # Zadaniem rekurencyjnej metody AddNode jest dodawanie
    # kolejnych wezLOW do kontrolki
    def AddNode(self, inXmlNode, inTreeNode):
        tNode = TreeNode()
        i = 0
        # Gdy rodzic inXmlNode posiada dzieci,
        # to rekurencyjnie wedrujemy po drzewku
        if (inXmlNode.HasChildNodes):
            for node in inXmlNode.ChildNodes:
                xNode = inXmlNode.ChildNodes[i]
                inTreeNode.Nodes.Add(TreeNode(xNode.Name))
                tNode = inTreeNode.Nodes[i]
                self.AddNode(xNode, tNode)
                i+=1
        else:
            inTreeNode.Text = (inXmlNode.OuterXml)
    # Klasa HelloXML dziedziczy po klasie Form
class HelloXML(Form):
    # Konstruktor z parametrem w postaci nazwy pliku XML
    def __init__(self, filename):
        self.xmlTree = xmlTree() # Instancja klasy xmlTree !
        self.xmlTree.pathToXml = filename
        # Zapelniamy drzewko kontrolki ...
        self.xmlTree.PopulateTree()
        self.xmlTree.Dock = DockStyle.Fill
        # ... i dodajemy je do formularza
        self.Controls.Add(self.xmlTree)
        self.Size = Size(300,200)
        self.AutoSizeMode= AutoSizeMode.GrowAndShrink
    # Tu metoda Main jest na zewnatrz klasy
    def Main(filename):
        Application.Run (HelloXML (filename))
    # Na koniec: uruchamiamy skrypt z parametrem z postaci pliku
    # XML i wywolujemy metode Main
    if __name__=="__main__":
        import sys
        Main(sys.argv[1])
```

w poprzednim przykładzie. Tym razem jednak, rozpoczniemy od utworzenia kontrolki przeglądarki i wywołania jej w nowym skrypcie. Zaczynamy od utworzenia nowego projektu typu *Class Library* w Visual Studio i utworzenia nowej klasy o nazwie (nie spodzianka!) *xmlTree*. Oczywiście, klasa ta powinna dziedziczyć po klasie *TreeView*. W ciele klasy pojawiają się konstruktor z parametrem w postaci nazwy pliku XML, którego strukturę odtwarzamy oraz prywatne metody o znanej już funkcjonalności i nazwach: *PopulateTree()* i *AddNode(XmlNode, TreeNode)*. Zmienna *m_directoryPath* posłuży nam do przechowywania nazwy odtwarzanego pliku. Dodatkowo, ciało klasy uzupełnimy o nową publiczną właściwość o nazwie *newFile*. Pozwoli ona nam zarówno na odczyt nazwy analizowanego pliku, jak i odtworzenie

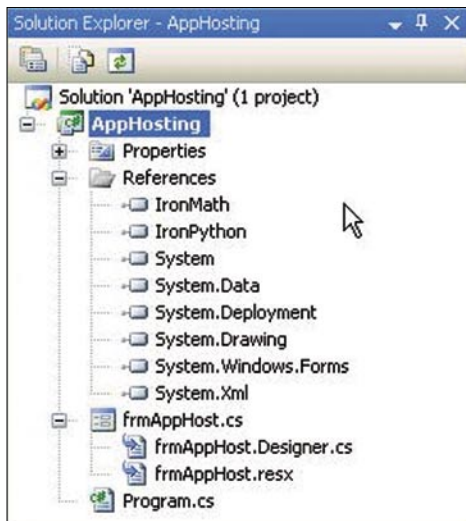
struktury pliku XML (pośrednio, poprzez wywołanie w treści właściwości metody *PopulateTree()*).

Tak utworzoną kontrolkę możemy bezproblemowo użyć w naszym skrypcie, co przedstawia Listing 3. Oprócz omawianej już zawartości w skrypcie pojawia się referencja do nowej kontrolki przy użyciu metody *AddReferenceToFile* modułu *clr* i import klasy *xmlTree* do naszego skryptu. W tym przypadku zaczynamy od utworzenia nowej klasy o nazwie *xmlViewer* dziedziczącej po klasie *Form*. W konstruktorze tworzymy instancję kontrolki *self.xmlTree* i odtwarzamy strukturę pliku XML. Teraz, wystarczy tylko dodać kontrolkę do formularza i po jego wywołaniu uzyskujemy identyczny efekt, jak w poprzednim przypadku.

Listing 2. Kontrolka *xmlTree* – C#

```
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using System.Xml;

// Dziedziczymy do klasy TreeView
public class xmlTree : System.Windows.Forms.TreeView{
    // składowe klasy ścieżka do pliku XML (z nazwa)
    private string m_directoryPath;
    // Bezparametrowy konstruktor
    public xmlTree() { InitializeComponent(); }
    // Przeciążony konstruktor z parametrem w postaci nazwy
    // pliku XML
    public xmlTree(string file){
        InitializeComponent(); // Inicjalizacja komponentu
        m_directoryPath = file;
        // Odtworzenie struktury pliku XML w kontrolce
        PopulateTree();
    }
    protected override void Dispose( bool disposing ){
        if( disposing ){
            if( components != null )
                components.Dispose();
        }
        base.Dispose( disposing );
    }
    private void InitializeComponent(){
        // Inicjalizacja kontrolki
    }
    // Odtwarzanie struktury pliku XML
    private void PopulateTree(){
        try{
            XmlDocument dom = new XmlDocument();
            dom.Load(m_directoryPath); // Pobranie pliku
            this.Nodes.Clear();
            // Tworzymy korzeń
            this.Nodes.Add(new TreeNode(dom.DocumentElement.
                Name));
            TreeNode tNode = new TreeNode();
            tNode = this.Nodes[0];
            // Rekurencyjnie wypełniamy kontrolkę węzłami
            // XmlNode
            AddNode(dom.DocumentElement, tNode);
        }
        // Obsługa wyjątków
        catch(XmlException xmlEx) {
            MessageBox.Show(xmlEx.Message);
        }
        catch(Exception ex) {
            MessageBox.Show(ex.Message);
        }
    }
    // Rekurencyjna metoda kopiująca strukturę dokumentu
    private void AddNode(XmlNode inXmlNode,
        TreeNode inTreeNode){
        XmlNode xNode; // wezeł DOM
        TreeNode tNode; // wezeł TreeNode
        XmlNodeList nodeList; // lista węzłów DOM
        int i;
        // Węzłowa po węzłach DOM do czasu napotkania
        // "bezdzielnego" węzła
        if (inXmlNode.HasChildNodes){
            nodeList = inXmlNode.ChildNodes;
            for(i = 0; i<=nodeList.Count - 1; i++){
                xNode = inXmlNode.ChildNodes[i];
                inTreeNode.Nodes.Add(new TreeNode(xNode.Name));
                tNode = inTreeNode.Nodes[i];
                AddNode(xNode, tNode); // Rekurencja...
            }
        } else {
            inTreeNode.Text = (inXmlNode.OuterXml).Trim();
        }
    }
    // new File zwraca nazwę dokumentu, ew. wyświetla/
    // generuje strukturę nowego pliku
    public XmlDocument newFile {
        get {
            return directoryPath;
        }
        set {
            m_directoryPath = value;
            PopulateTree();
        }
    }
}
```



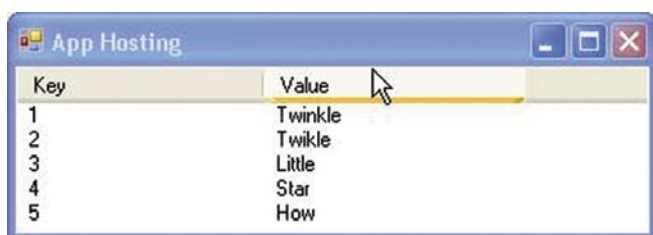
Rysunek 2. IronPython w Visual Studio – Solution Explorer

Hosting

Bardzo często przy projektowaniu aplikacji mamy do czynienia z potrzebą zwiększania funkcjonalności aplikacji czy automatyzacji określonych działań przy użyciu zewnętrznych skryptów (w dowolnym języku). Konieczny jest mechanizm interpretera udostępniającego określony interfejs aplikacji na zewnątrz w treści skryptu i umożliwiający pobranie wyniku działania skryptu z powrotem do otoczenia, z którego dane zostały wysłane. Działanie interpretera poleceń IronPythona zilustrujemy na przykładzie prostej aplikacji wczytującej zewnętrzny słownik (przy użyciu zewnętrznego skryptu .py) i wyświetlającej nowe dane w kontrolce listy (ListView). Takie podejście pozwala w naturalny sposób oddzielić logikę biznesową aplikacji od warstwy prezentacyjnej. Kod aplikacji przedstawiono w Listingu 4. Rozpoczynamy od utworzenia nowego projektu typu Windows Application w Visual Studio o przykładowej nazwie *frmAppHost*. Aby nasza aplikacja była w stanie interpretować wyniki działania wczytywanego skryptu, do projektu dodajemy referencje do nowych modułów: *IronPython.Modules* i *IronPython.Hosting* – zob. Rysunek 2. Nasza aplikacja jest prostym słownikiem, więc na początek deklarujemy zmienną *dictionary* klasy `Dictionary<int,string>` służącą przechowywaniu wczytywanych słów. W dalszej kolejności tworzymy interpreter Pythona o nazwie *engine* i przekierowujemy standardowe *we/wy* interpretera do nowego pliku instrukcjami

```
engine.SetStandardOutput(FileStreamObject);
engine.SetStandardError(FileStreamObject);
```

Potrzebny nam jeszcze nowy moduł *em* oraz słownik *locals* do przechowywania eksportowanych zmiennych (słownika – words (dictionary), nazwy pliku słownika – *myDictionaryFile*). Teraz wystarczy wykonać skrypt instrukcją `engine.ExecuteFile("getDictionary.py", em, locals);` aby odczytać zmo-



Rysunek 3. Słownik



Rysunek 4. Irrlicht .NET

dyfikowany już słownik i wyświetlić go w kontrolce listy *lvItems* naszego formularza jak na Rysunku 4.

Sam skrypt odczytujący słownik przedstawiony jest w Listingu 5. Jego treść nie odbiega wiele od tych, które widzieliśmy do tej pory. Oprócz znanych nam konstrukcji potrzebne są nam jeszcze typy ogólne (słownik), które importujemy instrukcją

```
from System.Collections.Generic import *
a sam słownik inicjujemy następująco
dict = Dictionary[Int32,String] ()
```

Listing 3. Przykład użycia kontrolki - Python

```
import sys # Dostep do standardowych modułów Pythona
sys.path.append(r"c:\python24\lib")
# Tradycyjnie, importujemy CLR i potrzebne biblioteki .NET
import clr
clr.AddReferenceByPartialName("System.Windows.Forms")
clr.AddReferenceByPartialName("System.Drawing")
from System.Windows.Forms import *
# Tworzymy referencje do utworzonej kontrolki przegladarki
clr.AddReferenceToFile("xmltree.dll") # Import kontrolki
import xmlTree
# Klasa xmlViewer to formularz - dziedziczy po klasie Form
class xmlViewer(Form):
    # Tworzymy konstruktor z parametrem w postaci nazwy
    # analizowanego pliku
    def __init__(self, filename):
        # Tworzymy instancje kontrolki i odtwarzamy structure
        # pliku XML
        self.xmlTree = xmlTree()
        self.xmlTree.newFile = filename
        # Dodajemy kontrolke przegladarki do formularza
        self.Controls.Add(self.xmlTree)
        # ...i ustawiamy parametry formularza
        self.AutoSize = True
        self.AutoSizeMode= AutoSizeMode.GrowAndShrink
# Identycznie jak poprzednio - Metoda Main
def Main(filename):
    Application.Run(xmlViewer(filename))
if __name__=="__main__":
    import sys
    Main(sys.argv[1])
```

Zwróćmy uwagę na sposób deklaracji typów ogólnych w IronPythonie. Działanie skryptu rozpoczynamy od sprawdzenia obecności słownika *myDictionaryFile*, aby w dalszej kolejności przejść do sekwencyjnego odczytu pliku i zapisu słów do słownika. Zwróćmy uwagę, że końcowe komunikaty pojawiają się w żądanym logu *application-log.txt*.

Zastosowanie: Grafika 3D

W aspekcie użycia IronPythona w rzeczywistych projektach, powstaje oczywiste pytanie, jak wygląda współpraca IronPythona

Listing 4. Użycie interpretera poleceń Pythona – C#

```
...
using IronPython.Hosting;
using IronPython.Modules;
...
private void frmAppHost_Load(object sender, EventArgs e)
{
    try
    {
        // Tworzymy słownik do przechowywania pobieranych słów
        Dictionary<int, string> dictionary =
            new Dictionary<int, string>();
        // Interpreter Pythona
        PythonEngine engine = new PythonEngine();
        // W języku Pythona: os.path.join(...)
        engine.AddToPath(Application.StartupPath);
        // Nowy log aplikacji - zamiast konsoli
        System.IO.FileStream fs = new System.IO.FileStream(
            "application-log.txt", System.IO.FileMode.Create);
        // Przekierowujemy standardowe we/wy do nowego pliku
        engine.SetStandardOutput(fs);
        engine.SetStandardError(fs);
        // Tworzymy nowy modul i słownik
        EngineModule em = engine.CreateModule();
        Dictionary<string, Object> locals =
            new Dictionary<string, object>();
        locals.Add("words", dictionary);
        locals.Add("myDictionaryFile", "dictionary.txt");
        engine.ExecuteFile("getDictionary.py", em, locals);
        engine.Shutdown();
        // Odczytujemy z powrotem liste
        dictionary = (Dictionary<int, string>)locals["words"];
        // Na koniec wypełniamy słownikiem liste
        foreach (KeyValuePair<int, string> item in
            dictionary) {
            ListViewItem lvItem = new ListViewItem(
                item.Key.ToString());
            lvItem.SubItems.Add(item.Value.Trim());
            lvItems.Items.Add(lvItem);
        }
    }
    catch (IronPython.Runtime.Exceptions.
        PythonNameErrorException E) {
        MessageBox.Show(E.Message);
    }
    catch (Exception E) {
        MessageBox.Show(E.Message);
    }
}
```

W Sieci

- <http://www.python.org>
- <http://irrlicht.sourceforge.net>
- <http://www.jython.org>
- <http://www.codeplex.com/Wiki/View.aspx?ProjectName=IronPython>
- <http://www.asp.net/ironpython>

z innymi bibliotekami. Okazuje się, że całkiem nieźle. Do demonstracji w naszym przypadku posłużyliśmy się w znanym środowiskiem graficznym Irrlicht w wersji .NET. Napiszemy skrypt przedstawiony w Listingu 6, wyświetlający teksturowaną siatkę terenu. Jak zwykle, importujemy moduły *clr* i *System*, a referencję do środowiska 3D tworzymy przy użyciu konstrukcji *AddReferenceToFile(...)*. W rezultacie, możemy tu już zaimportować samo środowisko i potrzebne nam biblioteki (*Video*, *Core*, *Scene*, *GUI*). Scenę skonfigurujemy w konstruktorze klasy *IrrlichtExample*. W konstruktorze tworzymy instancję bazowej klasy *IrrlichtDevice* *self.device*, sterownik video *self.driver*, kamerę *self.camera* oraz

Listing 5. Użycie interpretera poleceń Pythona – skrypt

```
# Jak we wszystkich przypadkach - import modułów i bibliotek
import sys
sys.path.append(r"c:\python24\lib")
import string
import os
# Import .NET
import clr
from System import *
from System.Collections.Generic import *
dict = Dictionary[Int32, String]()
# Na początek, sprawdzamy istnienie słownika
if os.path.exists(myDictionaryFile):
    f=open(filename, 'r') # Otwarcie pliku
    str = '_'
    count = 0; # licznik słów
    while str!=:
        try:
            line = f.readline() # Odczyt wiersz po wierszu
            if line!='':
                # Dzielimy wiersz na części
                (index, word) = line.split(r",")
                # ... i dodajemy do słownika
                dict.Add(Int32(index), String(word));
                count = count+1 # Zliczamy pozycje
        except IOError, (errno, strno):
            # Obsługa wyjątku
            print "%s in line %s\n", errno, strno
    f.close()
if (count>1):
    # Uwaga: Wszystkie komunikaty pojawia się w logu
    print 'Loaded data from file -> ', filename
    print 'There are items->', count
    for item in words:
        print item.Key, ":", item.Value
words = dict # koniec!
```

Listing 6. Przykład użycia środowiska grafiki 3D – Irrlicht .NET

```

# Tradycyjnie, importujemy potrzebne biblioteki
import clr
clr.AddReferenceToFile("Irrlicht.NET") # Irrlicht!
import Irrlicht
from Irrlicht import *
from Irrlicht.Video import *
from Irrlicht.Core import *
from Irrlicht.Scene import *
from Irrlicht.GUI import *
import System

# Prosta klasa, przy pomocy ktorej wyswietlimy tworzona
# scene
class IrrlichtExample:
    def __init__(self):
        try:
            # Tworzymy instancje klasy IrrlichtDevice
            self.device = IrrlichtDevice(DriverType.DIRECT3D8)
            self.device.ResizeAble = True;
            self.device.WindowCaption =
                "Iron Python + Irrlicht"
            # Pobieramy sterownik video
            self.driver = self.device.VideoDriver
            # ... oraz samo GUI
            self.env = self.device.GUIEnvironment
            self.driver.SetTextureCreationFlag(
                TextureCreationFlag.ALWAYS_32_BIT, True)
            # Menedzer sceny
            smgr = self.device.SceneManager
            # Tworzymy kamere i ustawiamy ja na scenie
            self.camera = smgr.AddCameraSceneNodeFPS()
            self.camera.Position=Vector3D(1900*2,255*2,3700*2)
            self.camera.Target= Vector3D(2397*2,343*2,2700*2)
            self.camera.FarValue=12000.0
            # Odtad kursor bedzie niewidoczny
            self.device.CursorControl.Visible=False
            # Generujemy teren
            terrain = smgr.AddTerrainSceneNode(
                "terrain-heightmap.bmp", None, -1,
                Vector3D(), Vector3D(40, 4.4, 40),
                Color(255,255,255,255))
            terrain.SetMaterialFlag(
                MaterialFlag.LIGHTING, False);
            terrain.SetMaterialType(MaterialType.DETAIL_MAP);
            # Ustawiamy tekstury terenu i skalujemy je
            terrain.SetMaterialTexture(0, self.driver.
                GetTexture("terrain-texture.jpg"));
            terrain.SetMaterialTexture(1, self.driver.
                GetTexture("detailmap3.jpg"));
            terrain.ScaleTexture(1.0, 20.0);
            # Wlaczamy detector kolizji,...
            selector = smgr.CreateTerrainTriangleSelector(
                terrain, 0)
            anim = smgr.CreateCollisionResponseAnimator(
                selector, self.camera, Vector3D(60,100,60),
                Vector3D(0,0,0), Vector3D(0,50,0), 0.0005);
            # ...ktory dodajemy do kamery
            self.camera.AddAnimator(anim);
            self.driver.SetTextureCreationFlag(
                TextureCreationFlag.CREATE_MIP_MAPS, False);
            # Ustawiamy kolejno tekstury otoczenia
            # (gora/dol/lewo/prawo/przod/tyl)
            smgr.AddSkyBoxSceneNode(
                self.driver.GetTexture("irrrlicht2_up.jpg" ),
                self.driver.GetTexture("irrrlicht2_dn.jpg"),
                self.driver.GetTexture("irrrlicht2_lf.jpg"),
                self.driver.GetTexture("irrrlicht2_rt.jpg"),
                self.driver.GetTexture("irrrlicht2_ft.jpg"),
                self.driver.GetTexture(
                    "irrrlicht2_bk.jpg"), None, 0);
            self.driver.SetTextureCreationFlag(
                TextureCreationFlag.CREATE_MIP_MAPS, True);
            # Wyswietlamy grafike az do zamkniecia aplikacji
            while self.device.Run():
                if self.device.WindowActive:
                    self.device.VideoDriver.BeginScene(
                        True, True, Color(0,100,100,100))
                    self.device.SceneManager.DrawAll();
                    self.device.VideoDriver.EndScene()
            self.device.CloseDevice()
        except Exception, detail:
            print detail
            # Metoda Main(), w ktorej utworzymy instancje naszej
            # klasy
            def Main():
                game = IrrlichtExample() # Instancja klasy Irrlicht
            # Gotowe...
            if __name__=="__main__":
                Main()

```

menedżera sceny (*smgr*). Ustawiamy kamerę w dogodnym miejscu na scenie (*self.camera.Position*), a teren generujemy techniką heightmap. Wynik jak na Rysunku 4.

Podsumowanie

Zakres i możliwości IronPython (i samego Pythona) są ogromne. IronPython pozwala użytkownikowi Pythona poruszać się bardzo sprawnie po platformie .NET. Posiada w pełni dynamiczny zestaw prostych i złożonych typów, jak i mechanizm automatycznego zarządzania pamięcią. Najważniejszy obszar zastosowania Pythona to szeroko pojęty RAD (*Rapid Application Development*). Z drugiej strony,

programista .NET dostaje do ręki dynamiczne narzędzie, mogące łatwo służyć uzupełnieniu lub zwiększeniu funkcjonalności opracowywanych aplikacji. To narzędzie, które może być użyte nie tylko do tworzenia mniej lub bardziej złożonych skryptów, ale także w dużych projektach programistycznych; pole możliwych zastosowań jest ogromne (aplikacje desktopowe, ASP. NET, Web frameworks hosting, itd). Aktualnie, wadą jego jest brak wsparcia dla serwisów webowych (*Web Services*). Skorzystają na nim wszyscy ci, którzy szukając dynamicznej alternatywy (lub uzupełnienia) dla statycznego C#. Krótko mówiąc, ograniczeń jest niewiele, a zabawa przednia. ■