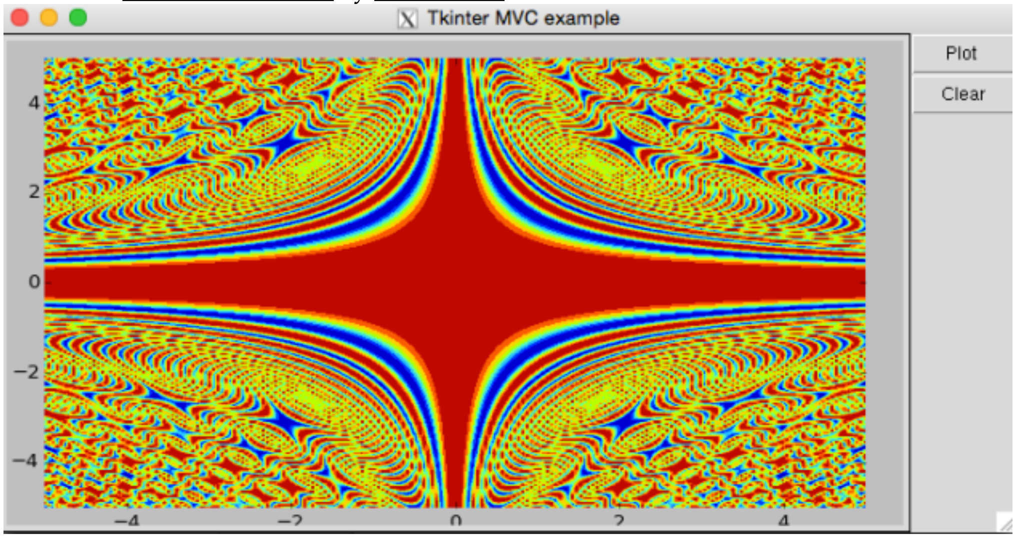
SukhbinderSingh.com

A self-directed apprenticeship……

An Example of Model View Controller Design Pattern with Tkinter Python

Posted on December 25, 2014 by sukhbinder



Model-view-controller (MVC) is the design pattern for successfully and efficiently relating the user interface to underlying data models.

This is a useful pattern for the reuse of object code and a pattern that allows to significantly reduce the time it takes to develop applications with user interfaces.

I have used MVC pattern in my ICP App (https://sukhbinder.wordpress.com/2014/04/24/python-app-to-perform-icp-alignment/) and the STL viewer (https://sukhbinder.wordpress.com/2014/01/13/slt-viewer-a-vtk-and-wxpython-powered-app/) but most of my scientific computing GUI is based on tkinter since its readlily available in standard pythan installation, so had this very basic MVC based framework for quick prototypes.

```
# -*- coding: utf-8 -*-

"""
Created on Feb 18 10:30:38 2014

@author: Sukhbinder Singh

A basic python example of Model–view–controller (MVC), a software design pattern for im

Mainly written as a quick start framework to quickly implement tkinter based GUI for pr


"""

import Tkinter as Tk
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import numpy as np




class Model():

    def __init__(self):
        self.xpoint=200
        self.ypoint=200
        self.res = None


    def calculate(self):
        x,y=np.meshgrid(np.linspace(-5,5,self.xpoint),np.linspace(-5,5,self.ypoint))
        z=np.cos(x**2*y**3)
        self.res = {"x":x,"y":y,"z":z}

class View():
    def __init__(self, master):
        self.frame = Tk.Frame(master)
        self.fig = Figure( figsize=(7.5, 4), dpi=80 )
        self.ax0 = self.fig.add_axes( (0.05, .05, .90, .90), axisbg=(.75,.75,.75), fram
        self.frame.pack(side=Tk.LEFT, fill=Tk.BOTH, expand=1)
        self.sidepanel=SidePanel(master)
        self.canvas = FigureCanvasTkAgg(self.fig, master=self.frame)
        self.canvas.get_tk_widget().pack(side=Tk.TOP, fill=Tk.BOTH, expand=1)
        self.canvas.show()

class SidePanel():
    def __init__(self, root):
        self.frame2 = Tk.Frame( root )
        self.frame2.pack(side=Tk.LEFT, fill=Tk.BOTH, expand=1)
        self.plotBut = Tk.Button(self.frame2, text="Plot ")
        self.plotBut.pack(side="top",fill=Tk.BOTH)
        self.clearButton = Tk.Button(self.frame2, text="Clear")
        self.clearButton.pack(side="top",fill=Tk.BOTH)

class Controller():
    def __init__(self):
        self.root = Tk.Tk()
        self.model=Model()
        self.view=View(self.root)
        self.view.sidepanel.plotBut.bind("<Button>",self.my_plot)
        self.view.sidepanel.clearButton.bind("<Button>",self.clear)

    def run(self):
        self.root.title("Tkinter MVC example")
        self.root.deiconify()
        self.root.mainloop()

    def clear(self,event):
        self.view.ax0.clear()
        self.view.fig.canvas.draw()

    def my_plot(self,event):
        self.model.calculate()
        self.view.ax0.clear()
        self.view.ax0.contourf(self.model.res["x"],self.model.res["y"],self.model.res["
        self.view.fig.canvas.draw()



if __name__ == '__main__':
    c = Controller()
    c.run()
```

This entry was posted in coding, python and tagged python. Bookmark the permalink.

12 thoughts on “An Example of Model View Controller Design Pattern with Tkinter Python”

1. Pingback: Python:Best way to structure a tkinter application – IT Sprite
2. Pingback: 构造一个标准应用最好的方式 – CodingBlog

Tony C says:
on May 8, 2018 at 12:34 am
The whole point of separating the GUI components is so that the controller doesn’t need to have any knowledge of the GUI. It shoulw only call functions/methods in the View object.
the Controller should not be dealing with these GUI details.
self.view.sidepanel.plotBut.bind(“”, self.my_plot)
self.view.sidepanel.clearButton.bind(“”, self.clear)
clear() and my_plot() must not be in the controller, they should be hidden in the View object

Reply

Tony C says:
on May 8, 2018 at 3:45 am
See this gist for a better example of separating the GUI dependencies

```
try:
    import Tkinter as Tk # python 2
```

```
except ModuleNotFoundError:
    import tkinter as Tk # python 3

from model import Model
from view import View


class Controller:
    def __init__(self):
        self.root = Tk.Tk()
        self.model = Model()
        self.view = View(self.root, self.model)

    def run(self):
        self.root.title("Tkinter MVC example")
        self.root.deiconify()
        self.root.mainloop()
```

view raw controller.py hosted with ♥ by **GitHub**

```
# -*- coding: utf-8 -*-

"""
Created on May 7, 2018
This is an improvment on the original program posted here:
https://sukhbinder.wordpress.com/2014/12/25/an-example-of-model-view-controller-design-pattern-with-tkir

The original program tightly coupled GUI dependencies into the Controller class, which defeats the whole MVC 

The example below is an improvement on the original program.
Each of the Model, View, Controller and SidePanel objects are in their own modules.
The program now runs on Python 2 & Python3, without any modifications.


"""

from controller import Controller

if __name__ == '__main__':
    c = Controller()
    c.run()
```

view raw main.py hosted with ♥ by **GitHub**

```
import numpy as np


class Model:

    def __init__(self):
        self.xpoint = 200
        self.ypoint = 200
        self.res = None

    def calculate(self):
        x, y = np.meshgrid(np.linspace(-5, 5, self.xpoint), np.linspace(-5, 5, self.ypoint))
        z = np.cos(x ** 2 * y ** 3)
        self.res = {"x": x, "y": y, "z": z}
```

view raw model.py hosted with ♥ by **GitHub**

```
try:
    import Tkinter as Tk # python 2
except ModuleNotFoundError:
    import tkinter as Tk # python 3


class SidePanel():
    def __init__(self, root):
        self.frame2 = Tk.Frame(root)
        self.frame2.pack(side=Tk.LEFT, fill=Tk.BOTH, expand=1)
        self.plotBut = Tk.Button(self.frame2, text="Plot ")
        self.plotBut.pack(side="top", fill=Tk.BOTH)
        self.clearButton = Tk.Button(self.frame2, text="Clear")
        self.clearButton.pack(side="top", fill=Tk.BOTH)
```

view raw side_panel.py hosted with ❤ by **GitHub**

```

try:
    import Tkinter as Tk # python 2
except ModuleNotFoundError:
    import tkinter as Tk # python 3

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure

from side_panel import SidePanel


class View:
    def __init__(self, root, model):
        self.frame = Tk.Frame(root)
        self.model = model
        self.fig = Figure(figsize=(7.5, 4), dpi=80)
        self.ax0 = self.fig.add_axes((0.05, .05, .90, .90), facecolor=(.75, .75, .75), frameon=False)
        self.frame.pack(side=Tk.LEFT, fill=Tk.BOTH, expand=1)
        self.sidepanel = SidePanel(root)

        self.sidepanel.plotBut.bind("<Button>", self.plot)
        self.sidepanel.clearButton.bind("<Button>", self.clear)

        self.canvas = FigureCanvasTkAgg(self.fig, master=self.frame)
        self.canvas.get_tk_widget().pack(side=Tk.TOP, fill=Tk.BOTH, expand=1)
        self.canvas.show()

    def clear(self, event):
        self.ax0.clear()
        self.fig.canvas.draw()

    def plot(self, event):
        self.model.calculate()
        self.ax0.clear()
        self.ax0.contourf(self.model.res["x"], self.model.res["y"], self.model.res["z"])
        self.fig.canvas.draw()
```

view raw view.py hosted with ❤ by **GitHub**

Reply

sukhbinder says:
on May 8, 2018 at 10:08 pm
Hi Tony,, just saw this. I think the view and model are not separate in the implementation that is in the above gist

and if the model changes, the view will need to change… . Will check this in detail… thanks for sharing!!.

Cheers

sukhbinder says:
on May 8, 2018 at 10:06 pm
Hi Tony

Thanks for your feedback.

As the post states, this is a very basic example to create quick prototypes, so you might be right.

But what I have learned from static typed languages is that you define the model and views as complete separate entities, and the controller takes an instantiation of both model and views as parameters. If nothing breaks and the controller does all of the communication then yes, an application is MVC.

Not necessary for the presented simple and quick prototypes but we might look at other design patterns such as Singleton, Factory and others that can help address your points.

Have you tried anything better? I will be happy to learn a simple way to achieve this.

Many thanks for reading and commenting
Sukhbinder

Reply

sathish says:
on May 11, 2018 at 7:49 am
Thanks for your code .

I am currently getting error:
AttributeError: '_tkinter.tkapp' object has no attribute 'deconify'

Reply

sukhbinder says:
on May 11, 2018 at 10:24 am
Hi Satish which version of python and tk are you using? This code is tested in 2.7

Reply

sathish says:
on May 12, 2018 at 11:54 am
I am currently using python 3.6.Could please about why we using root.deconify and what is the purpose of that?

Reply

sukhbinder says:
on May 12, 2018 at 12:14 pm
The purpose of deconify to force detraw of the window. You can comment it. Should not affect your functionality. Please refer to Tkinter python 3 documentation. I think you should get an equivalent keyword if you need it. Thanks

Reply

Travis says:
on November 13, 2018 at 8:28 pm
If you want this to work with Python 3.6, make these changes:
1. Change import Tkinter as Tk
TO:
import tkinter as Tk

2. Change self.ax0 = self.fig.add_axes((0.05, .05, .90, .90), axisbg=(.75,.75,.75), frameon=False)
TO:
self.ax0 = self.fig.add_axes((0.05, .05, .90, .90), facecolor=(.75,.75,.75), frameon=False)

3. Change self.canvas.show()
TO:
self.canvas.draw()

Reply

sukhbinder says:
on November 13, 2018 at 10:45 pm
Thanks Travis!!

Reply