Graphical User Interfaces

1. User Interfaces
   - GUIs in Python with Tkinter
   - object oriented GUI programming
2. Mixing Colors
   - specification of the GUI
   - the widget Scale
3. Simulating a Bouncing Ball
   - layout of a basic GUI
   - Canvas and Button widgets
   - methods stop, start, animate
   - methods drawball() and map2table()

MCS 275 Lecture 5
Programming Tools and File Management
Jan Verschelde, 20 January 2017

Graphical User Interfaces

1. User Interfaces
   - GUIs in Python with Tkinter
   - object oriented GUI programming
2. Mixing Colors
   - specification of the GUI
   - the widget Scale
3. Simulating a Bouncing Ball
   - layout of a basic GUI
   - Canvas and Button widgets
   - methods stop, start, animate
   - methods drawball() and map2table()

User Interfaces

Interfaces define the dialogue with the user,
which concerns mainly input and output.

Compared to prompting and printing of command line interfaces,
graphical user interfaces offer

1. a friendly face of the program,
2. give the user control of the actions via buttons to press GUIs are *event driven*,
3. one picture says more than one thousand words.

Good design: interface is ***separate*** from the software that does the calculations or the transactions.

Tkinter, Tk, and Tcl

GUIs for Python programmers

The `Tkinter` (= `Tk` interface) library provides an object-oriented interface to the `Tk` GUI toolkit, the graphical interface development tool for `Tcl`, `Tk` = Tool Kit, `Tcl` = Tool Command Language.

Benefit: platform independent GUI development.

GUI programming in Python

Tkinter is a toolkit available to Python as a module:

```
>>> import tkinter
```

Often it needs to be installed separately.

There are alternative toolkits to build GUIs in Python.

Principles of GUI programming transfer to other toolkits.

Graphical User Interfaces

1. User Interfaces
   - GUIs in Python with Tkinter
   - object oriented GUI programming
2. Mixing Colors
   - specification of the GUI
   - the widget Scale
3. Simulating a Bouncing Ball
   - layout of a basic GUI
   - Canvas and Button widgets
   - methods stop, start, animate
   - methods drawball() and map2table()

Object Oriented GUI Programming

We will develop our GUIs in an object oriented manner.

The constructor of our class defines

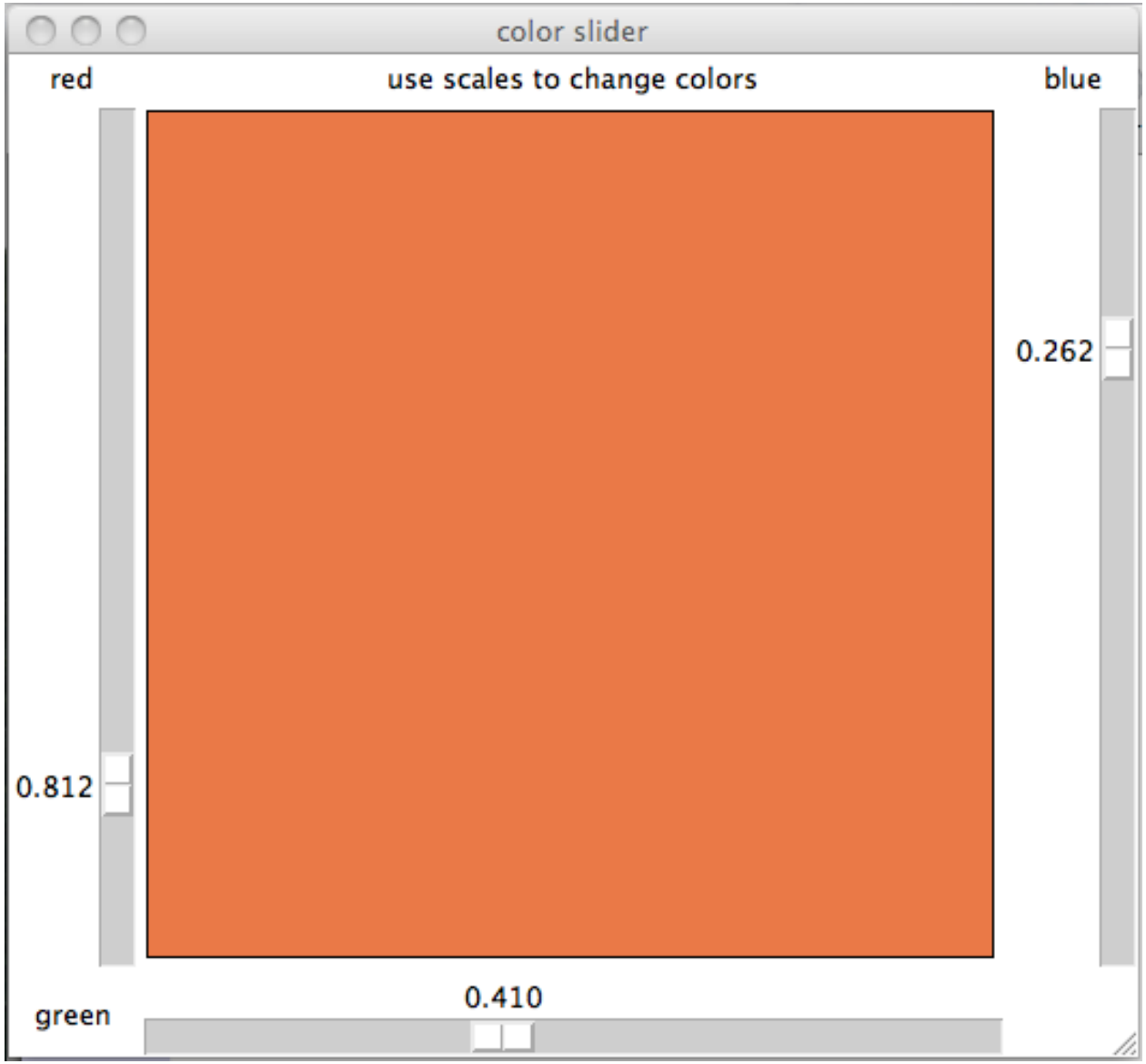
1. the data attributes of the GUI
2. the graphical appearance:
   1. which widgets we will use
   2. the position of the widgets in the window

The actions triggered by buttons are coded
in separate methods of the class.

Graphical User Interfaces

1. User Interfaces
   - GUIs in Python with Tkinter
   - object oriented GUI programming
2. Mixing Colors
   - specification of the GUI
   - the widget Scale
3. Simulating a Bouncing Ball
   - layout of a basic GUI
   - Canvas and Button widgets
   - methods stop, start, animate
   - methods drawball() and map2table()

an application: mixing red, green, blue



the class definition

```
from tkinter import Tk, Label, Canvas, Scale, DoubleVar

class ColorGUI(object):
    """
    Manipulate rgb color parameters with scale widgets.
    """
    def __init__(self, wdw, dim):
        """
        Defines a canvas and three scales.
        """
        wdw.title('color slider')
        self.dim = dim
        self.lab = Label(wdw, text='use scales to change colors')
        self.lab.grid(row=0, column=1)
        self.cnv = Canvas(wdw, width=self.dim, height=self.dim)
        self.cnv.grid(row=1, column=1)
        self.redscale(wdw)
        self.bluescale(wdw)
        self.greenscale(wdw)
```

Graphical User Interfaces

1. User Interfaces
   - GUIs in Python with Tkinter
   - object oriented GUI programming
2. Mixing Colors
   - specification of the GUI
   - the widget Scale
3. Simulating a Bouncing Ball
   - layout of a basic GUI
   - Canvas and Button widgets
   - methods stop, start, animate
   - methods drawball() and map2table()

the widget Scale

A scale is good for

1. setting default values for numerical input; and
2. restricting the range of the values.

In our application of mixing colors:

- The variable set by the scale is a double, ranging from 0 to 1 with resolution `1.0/256`.
- After every change in the variable, the method `show_colors` is executed.
- When the GUI starts up, the scale has value `0.5`.

code for one scale

To make the definition of `__init__` shorter,
we define the scales in separate methods.

```
def redscale(self, wdw):
    """
    Defines the scale to set the red intensity,
    stored in the DoubleVar red, a data attribute.
    """
    self.labred = Label(wdw, text='red')
    self.labred.grid(row=0, column=0)
    self.red = DoubleVar() # red intensity
    self.scared = Scale(wdw, orient='vertical', \
        length=self.dim, from_=0.0, to=1.0, \
        resolution=1.0/256, variable=self.red, \
        command=self.show_colors)
    self.scared.set(0.5)   # initial value of red scale
    self.scared.grid(row=1, column=0)
```

the callback function `show_colors`

```
def show_colors(self, val):
    """
    Displays a rectangle on canvas, filled with rgb colors.
    """
    xmd = self.dim/2+1
    ymd = self.dim/2+1
    mid = self.dim/2-3
    red = self.scared.get()
    green = self.scagrn.get()
    blue = self.scablu.get()
    print('r = %f, g = %f, r = %f' % (red, green, blue))
    hxr = '%.2x' % int(255*red)
    hxg = '%.2x' % int(255*green)
    hxb = '%.2x' % int(255*blue)
    color = '#' + hxr + hxg + hxb
    self.cnv.delete('box')
    self.cnv.create_rectangle(xmd-mid, ymd-mid, xmd+mid, ymd+mid,
        width=1, outline='black', fill=color, tags='box')
```

an explanation for `show_colors`

Key aspects of the method `show_colors`:

- The argument `val` of `show_colors` is the value of the scale, but we need the values of all three intensities.
- With `self.scared.get()` we get the red intensity.
  Green and blue intensities are set by the scales with names `scagrn` and `scablu` respectively.
- The `print` writes to the terminal.
- `hxr = '%.2x' % int(255*red)` converts the intensity as a float in $[0, 1]$ to a two-digit hexadecimal integer.
- The large rectangle written to canvas has tag `box` and with this name we can wipe out the previous color.

the “main” program

```
def main():
    """
    Makes the GUI and launches
    the main event loop.
    """
    top = Tk()
    ColorGUI(top, 400)
    top.mainloop()

if __name__ == '__main__':
    main()
```

A moving Billiard Ball

Consider a billiard ball rolling over a pool table, bouncing against the edges of the table.

We will develop first a very basic GUI and then later add extra features.

Basic elements of the GUI:

1. drawing on canvas
2. buttons call for action
3. animation of the rolling ball

Goal of this lecture: cover enough

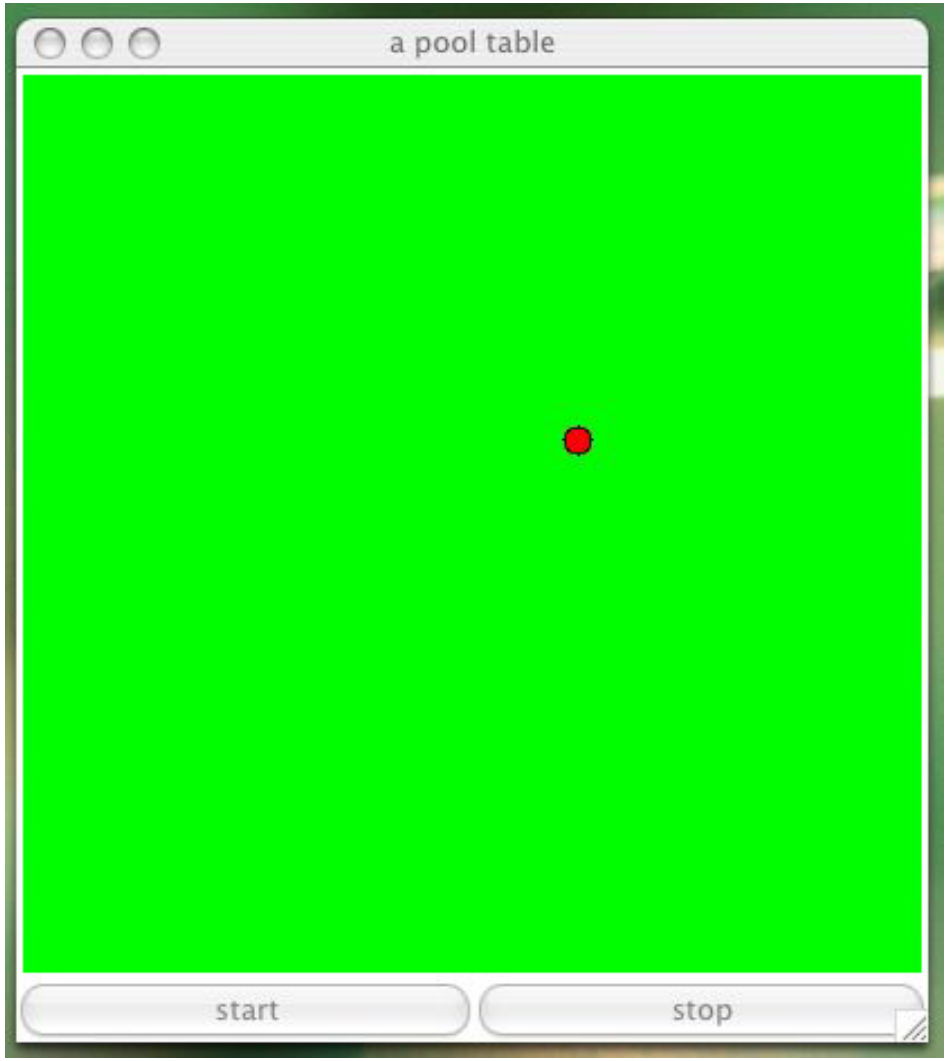
1. to start project one
2. to use in recursive drawings later

Graphical User Interfaces

1. User Interfaces
   - GUIs in Python with Tkinter
   - object oriented GUI programming
2. Mixing Colors
   - specification of the GUI
   - the widget Scale
3. Simulating a Bouncing Ball
   - layout of a basic GUI
   - Canvas and Button widgets
   - methods stop, start, animate
   - methods drawball() and map2table()

Simulating a Bouncing Ball

layout of basic GUI



Specifications of the GUI

A GUI consists of several components (called *widgets*).

Our first basic layout uses

1. a canvas to draw a ball
2. a button to start the animation
3. a button to stop the animation

Actions triggered by buttons:

1. start:
   1. the initial position of the ball is random
   2. the ball start rolling in a random direction, bouncing off against the edges of the table

   After a stop, the balls rolls from the previous position, but in a different random direction.
2. stop: the animation stops

Object Oriented Design

```
class BilliardBall(object):
    """
    GUI to simulate billiard ball movement.
    """
    def __init__(self, wdw, dimension, increment, delay):
        """
        Determines the layout of the GUI.
        """
    def animate(self):
        """
        Performs the animation.
        """
    def start(self):
        """
        Starts the animation.
        """
    def stop(self):
        """
        Stops the animation.
        """
```

the main program

```
def main():
    """
    Defines the dimensions of the canvas
    and launches the main event loop.
    """
    top = Tk()
    dimension = 400 # dimension of pool table
    increment = 10  # increment for coordinates
    delay = 60      # how much sleep before update
    BilliardBall(top, dimension, increment, delay)
    top.mainloop()

if __name__ == "__main__":
    main()
```

The code for the class is considered as a module.
The last line allows to run the module directly as a script.

the constructor `__init__`

The object data attributes are

1. constants: `dimension`, `increment`, and `delay` are the parameters of the GUI:
   - `dimension`: size of the square canvas,
   - `increment`: step size of one billiard ball move
   - `delay`: time between canvas updates;
2. variables:
   1. `togo`: animation on or off,
   2. `xpt`: x coordinate of the ball,
   3. `ypt`: y coordinate of the ball;
3. widgets:
   1. canvas spans two columns, in row 0,
   2. start button on row 1, column 0,
   3. stop button on row 1, column 1.

code for __init__

```
def __init__(self, wdw, dimension, increment, delay):
    """
    Determines the layout of the GUI.
    wdw : top level widget, the main window,
    dimension : determines the size of the canvas,
    increment : step size for a billiard move,
    delay : time between updates of canvas.
    """
    wdw.title('a pool table')
    self.dim = dimension # dimension of the canvas
    self.inc = increment
    self.dly = delay
    self.togo = False # state of animation
    # initial coordinates of the ball
    self.xpt = random.randint(10, self.dim-10)
    self.ypt = random.randint(10, self.dim-10)
```

Graphical User Interfaces

1. User Interfaces
   - GUIs in Python with Tkinter
   - object oriented GUI programming
2. Mixing Colors
   - specification of the GUI
   - the widget Scale
3. Simulating a Bouncing Ball
   - layout of a basic GUI
   - Canvas and Button widgets
   - methods stop, start, animate
   - methods drawball() and map2table()

Canvas and Buttons – code for __init__ continued

```
self.cnv = Canvas(wdw, width=self.dim,\
    height=self.dim, bg='green')
self.cnv.grid(row=0, column=0, columnspan=2)
self.bt0 = Button(wdw, text='start',\
    command=self.start)
self.bt0.grid(row=1, column=0, sticky=W+E)
self.bt1 = Button(wdw, text='stop',\
    command=self.stop)
self.bt1.grid(row=1, column=1, sticky=W+E)
```

The buttons trigger actions `start()` and `stop()` which are methods of the `BilliardBall` class.

Graphical User Interfaces

1. User Interfaces
   - GUIs in Python with Tkinter
   - object oriented GUI programming
2. Mixing Colors
   - specification of the GUI
   - the widget Scale
3. Simulating a Bouncing Ball
   - layout of a basic GUI
   - Canvas and Button widgets
   - methods stop, start, animate
   - methods drawball() and map2table()

methods `stop()` and `start()`

```
def start(self):
    """
    Starts the animation.
    """
    self.togo = True
    self.animate()

def stop(self):
    """
    Stops the animation.
    """
    self.togo = False
```

The `animate` method will

1. draw the ball on canvas;
2. generate a random direction;
3. move the ball in the direction using increment, as long as `togo == True`;
4. update the canvas after a delay.

the method `animate()`

```
def animate(self):
    """
    Performs the animation.
    """
    self.drawball()
    angle = random.uniform(0, 2*pi)
    vcx = cos(angle)
    vsy = sin(angle)
    while self.togo:
        (xpt, ypt) = (self.xpt, self.ypt)
        self.xpt = xpt + vcx*self.inc
        self.ypt = ypt + vsy*self.inc
        self.cnv.after(self.dly)
        self.drawball()
        self.cnv.update()
```

Graphical User Interfaces

1. User Interfaces
   - GUIs in Python with Tkinter
   - object oriented GUI programming
2. Mixing Colors
   - specification of the GUI
   - the widget Scale
3. Simulating a Bouncing Ball
   - layout of a basic GUI
   - Canvas and Button widgets
   - methods stop, start, animate
   - methods drawball() and map2table()

the method `DrawBall()`

The ball is represented as a red circle,
with a radius of 6 pixels:

```
def drawball(self):
    """
    Draws the ball on the pool table.
    """
    xpt = self.map2table(self.xpt)
    ypt = self.map2table(self.ypt)
    self.cnv.delete('dot')
    self.cnv.create_oval(xpt-6, ypt-6, xpt+6, ypt+6,\
        width=1, outline='black', fill='red', tags='dot')
```

As the ball moves, the values of the coordinates
may exceed the canvas dimensions.
With `map2table()` we get the bouncing effect.

the method `map2table()`

On canvas: (0,0) is at the topleft corner. Recall that `self.dim` stores the number of pixel rows and columns on canvas.

```
def map2table(self, pnt):
    """
    Keeps the ball on the pool table.
    """
    if pnt < 0:
        (quo, rest) = divmod(-pnt, self.dim)
    else:
        (quo, rest) = divmod(pnt, self.dim)
    if quo % 2 == 1:
        rest = self.dim - rest
    return rest
```

Imagine a succession of pool tables next to each other.
When the quotient `quo` is odd, we reflect,
otherwise we copy remainder `rest`.

Exercises

1. Adjust the GUI to simulate the movement of a billiard ball to work for rectangular pool tables.
2. Modify the GUI to simulate the movement of a billiard ball to use Entry widgets to allow the user to enter the initial position of the ball.
3. Add an extra row to the GUI, displaying in two Entry widgets the coordinates of the direction vector.
4. Add an Entry widget that will display the distance the ball has traveled. The value displayed in this Entry widget should change as the ball rolls.