Software Application Development by Python

Graphical User Interface Demonstration

Tong Zhang (张 彤)

E-mail: zhangtong@sinap.ac.cn

Shanghai Institute of Applied Physics, CAS

2016-05-27

Key Points

- Basic knowledge of Python
  - Syntax,
  - Reserved keywords,
  - Control flows,
  - Functions,
  - Built-in modules/packages,
  - Rules, styles,
  - **Documentation**, ...
- Connect with third-party packages
  - Numerical calculation: `numpy`, `scipy`
  - Plotting: `matplotlib`
  - GUI: `wxPython`
  - ...
- Learning philosophy
  - Solving problem,
  - **Having fun**.

Documentation:

- https://docs.python.org/2.7/
- http://docs.scipy.org/doc/
- http://matplotlib.org/contents.html
- http://wxpython.org/Phoenix/docs/html/Window.html

GUI application demo — PYDRAW

Goals

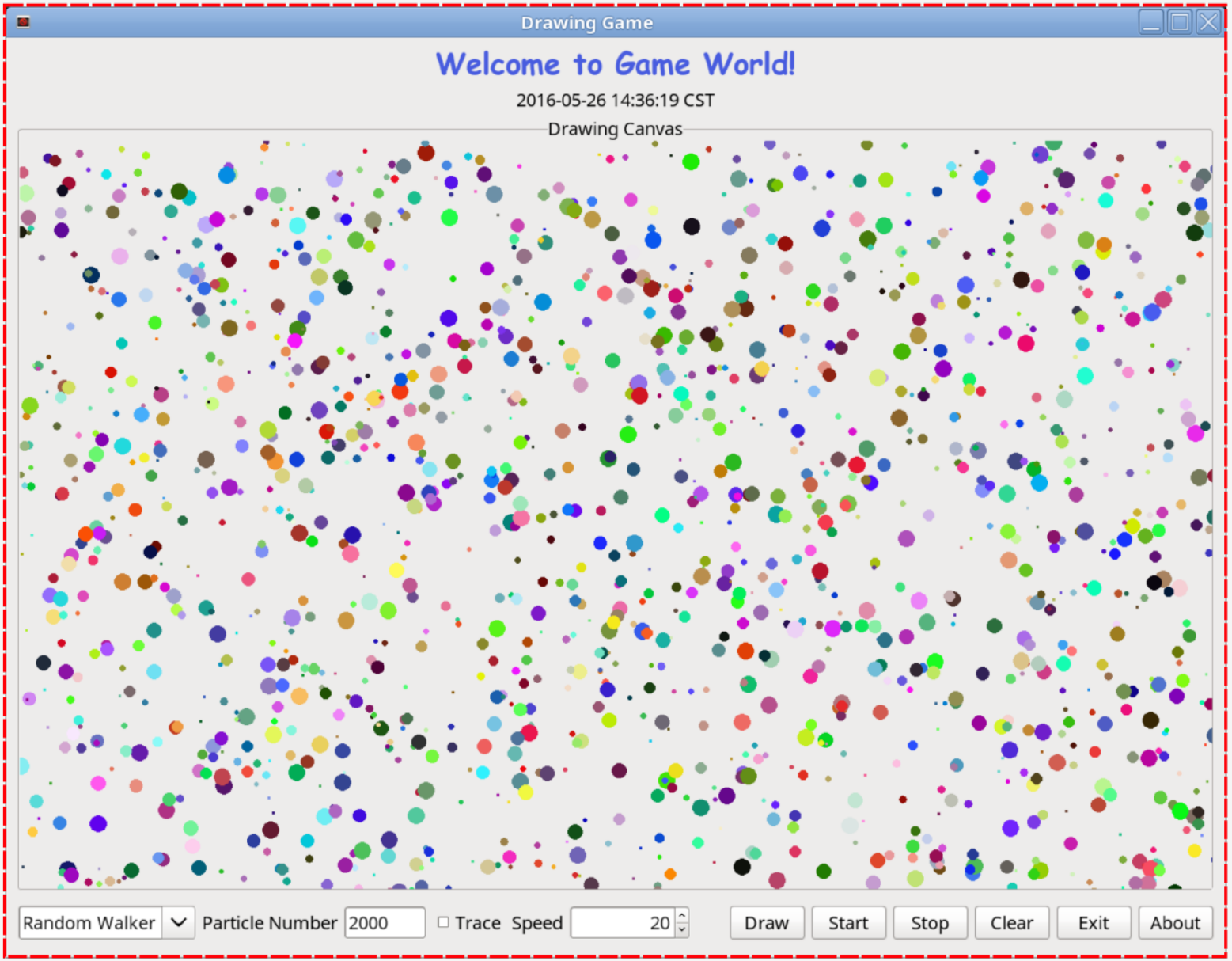
- Workflow of GUI application development
- Incorporate data to GUI skeleton
- Interactive with GUI
- Illustrate problem-solving procedure
- Python essential: multi-threading, efficient animation, OOP, . . .

App Spec

- Tools: Python 2.7.6, wxPython 3.0.2.0 gtk2 (classic)
- OS: Linux Mint 17.3
- Install: ‘pip install pydraw’
- URL: https://github.com/Archman/pydraw
- Installer for windows
- Run: ‘pydraw’

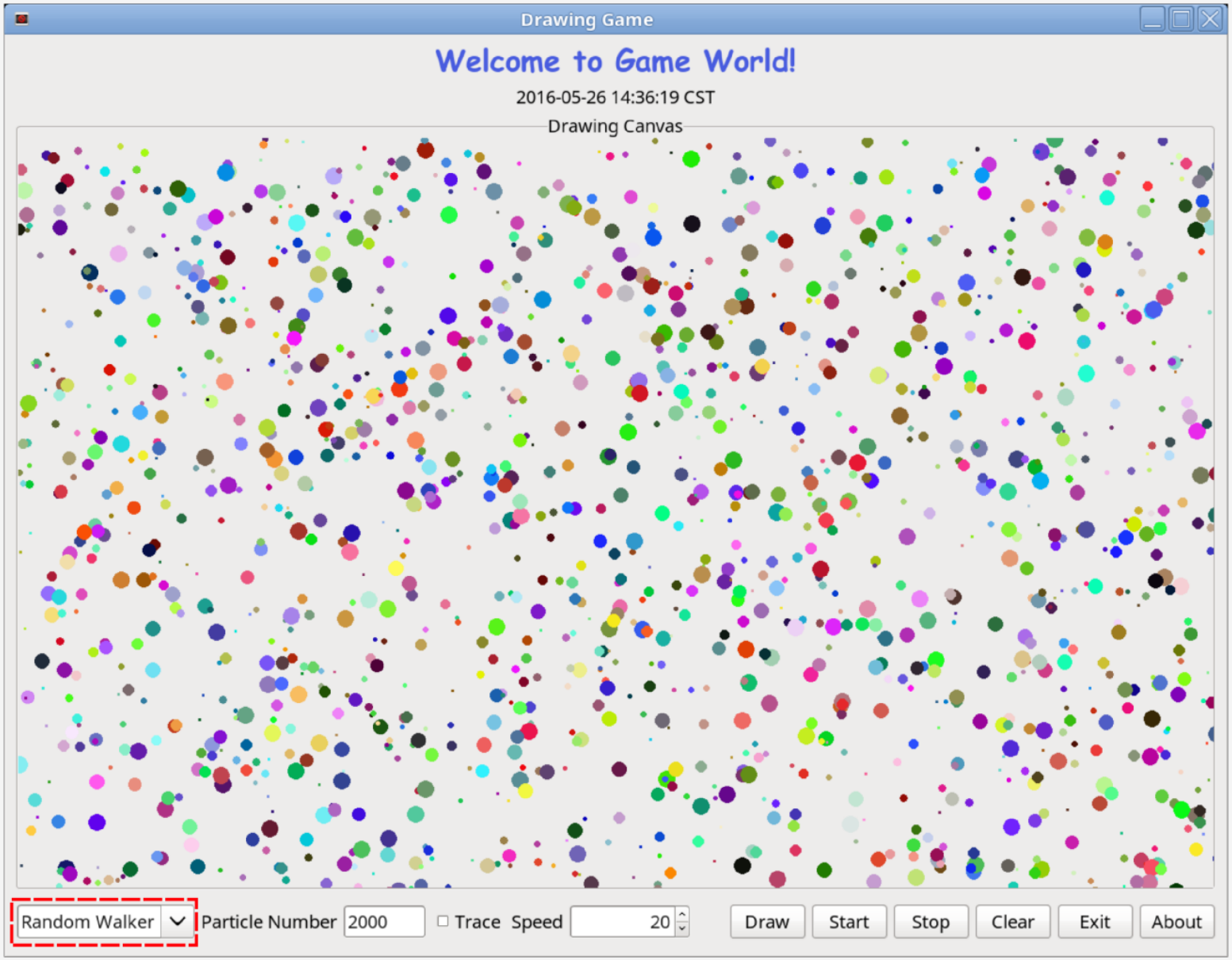
GUI application demo — PYDRAW

GUI application demo — PYDRAW



Frame

GUI application demo — PYDRAW

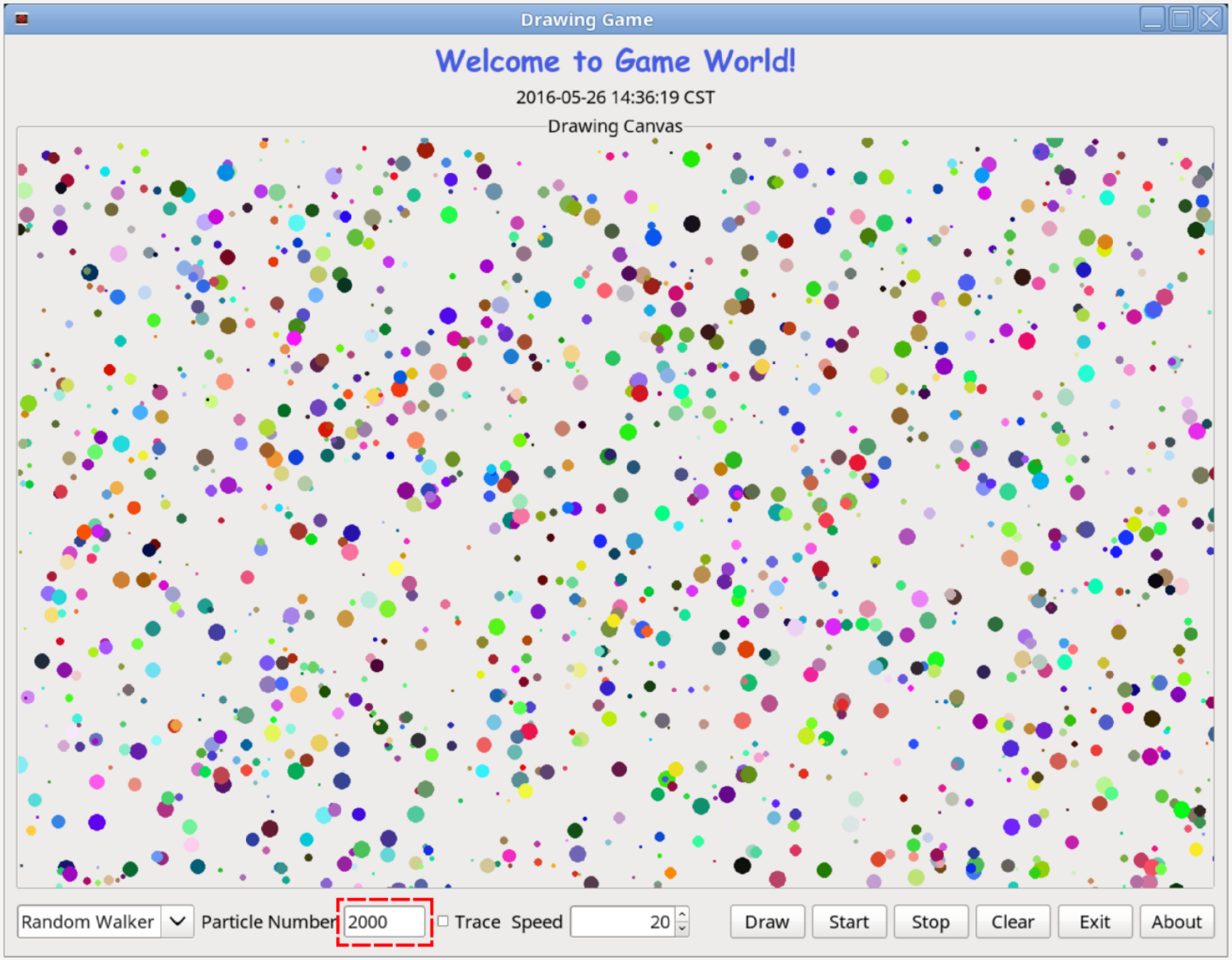


ComboBox

GUI application demo — PYDRAW

StaticText

GUI application demo — PYDRAW

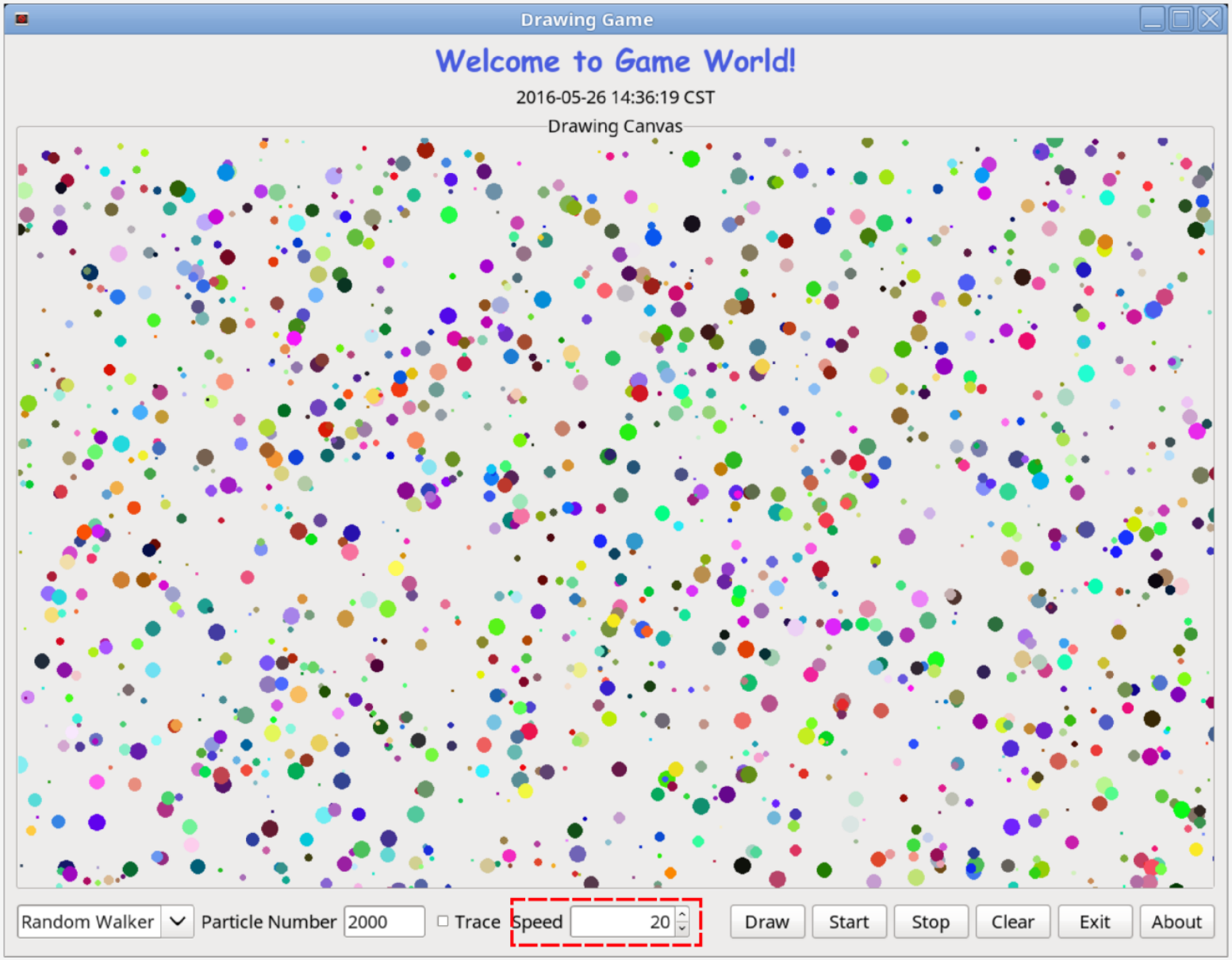


TextCtrl

GUI application demo — PYDRAW

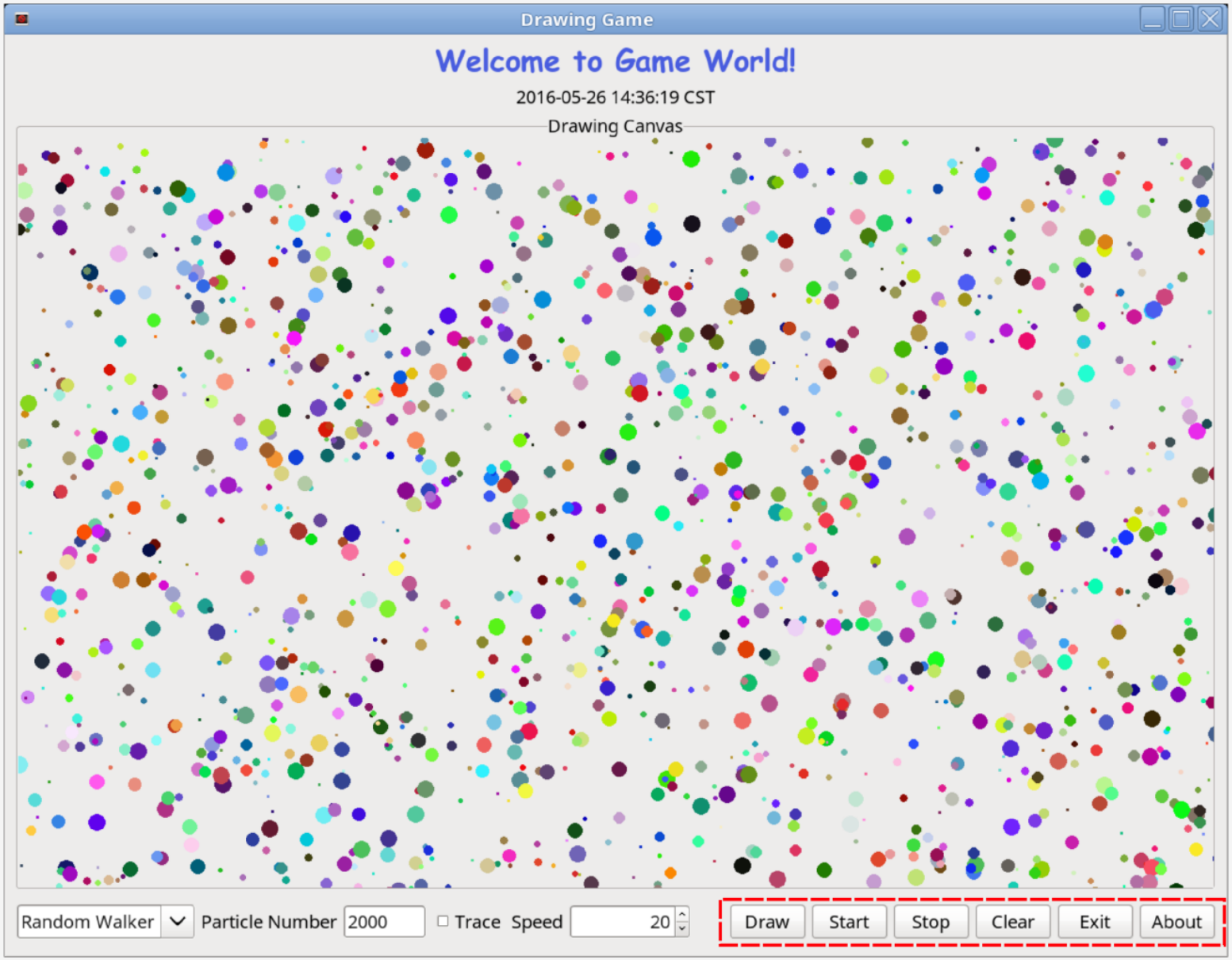
CheckBox

GUI application demo — PYDRAW



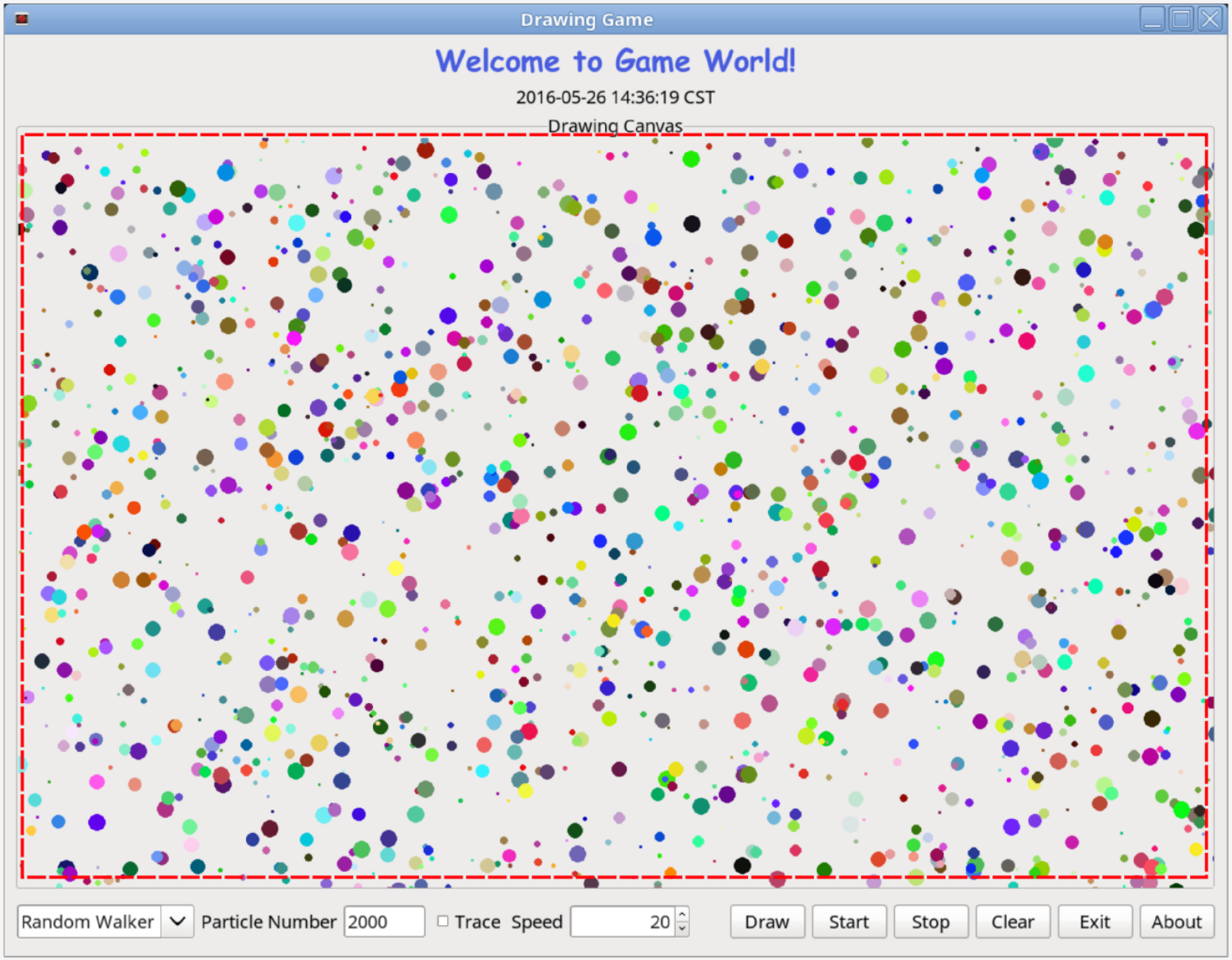
SpinCtrl

GUI application demo — PYDRAW



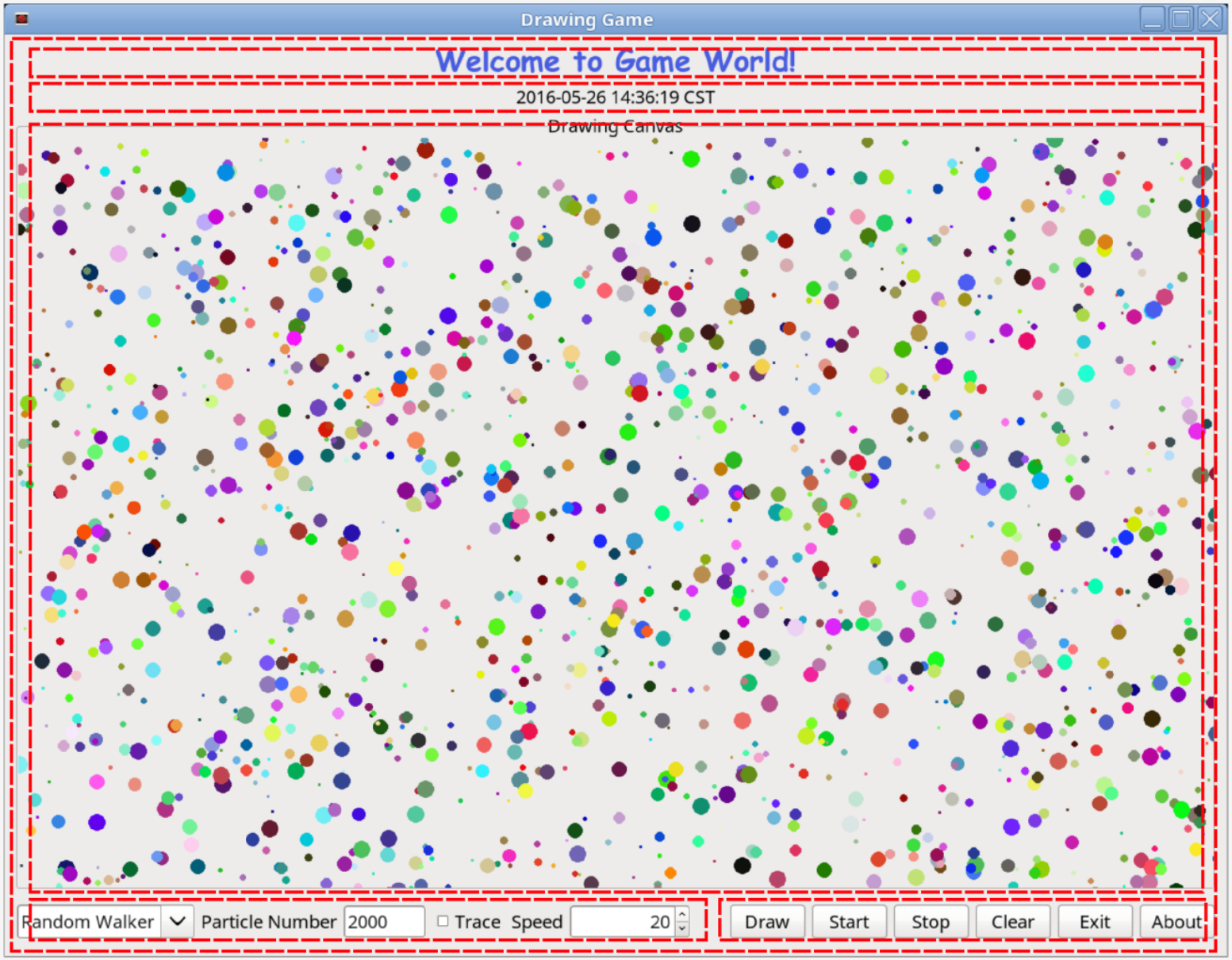
Button

GUI application demo — PYDRAW



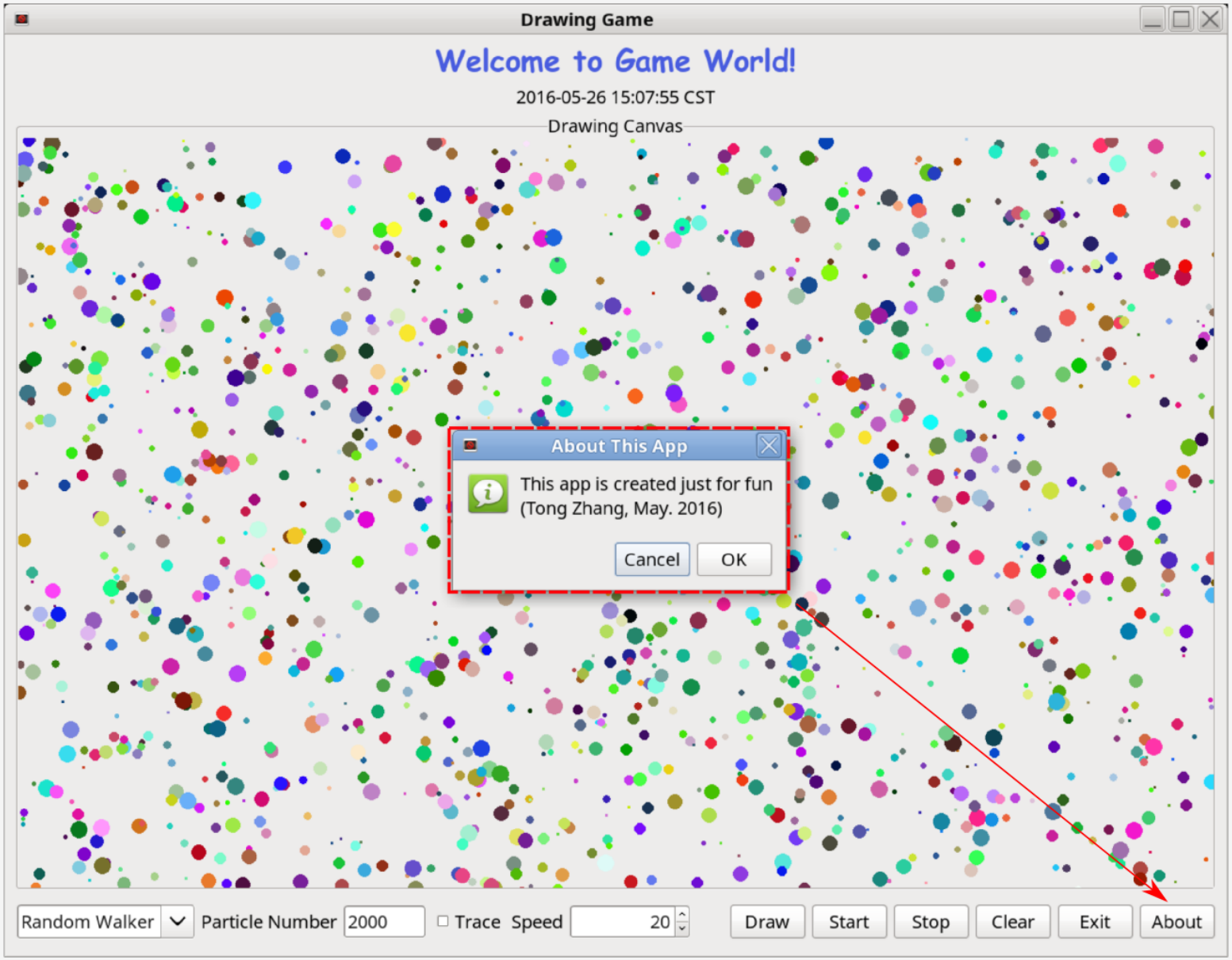
Panel

GUI application demo — PYDRAW



Sizer: GUI object container

GUI application demo — PYDRAW



MessageDialog

Event Binding

GUI application demo — PYDRAW

Create GUI app with wxPython:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" Drawing application for fun.

    Tong Zhang
    2016-05-24 15:30:04 PM CST
"""
import wx

class DrawFrame(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)
def run():
    app = wx.App()
    drawFrame = DrawFrame(None, -1, 'Drawing Game')
    drawFrame.Center()
    drawFrame.SetSize((1360, 1058))
    drawFrame.Show()
    app.MainLoop()
if __name__ == '__main__':
    run()
```

GUI application demo — PYDRAW

Wirte `DrawFrame` class:

1. Make plan for the GUI object container layout, i.e. `Sizer`;
2. Create GUI objects regarding different functionality;
3. Place GUI objects into right place;
4. Bind correct event callbacks to GUI objects;
5. Debuging/Testing/Packaging/Deploying;
6. Always documenting.

```
# create draw button
draw_btn = wx.Button(self, label='&Draw')
self.Bind(wx.EVT_BUTTON, self.onDraw, draw_btn)

def onDraw(self, e):
    # execute when Draw button is pushed
    # first initialize graphic drawing device context
    # then start initial drawing,
    # i.e. randomly draw circles with random color and size
    # see file: drawing.py
    self.initBuffer()  # line 218 to 224
    self.draw()        # line 226 to 239
```

Place GUI object into container, i.e. layout

```
# the sizer strategy here:
# draw_btn is placed into a horizontal box (hbox2),
# hbox2 also contains other 5 Buttons,
# hbox2 is put into a larger horizontal box (hbox),
# hbox is put into a larger vertical box (vbox),
# finally, layout vbox in the Frame.
# details see: file drawing.py line 73 to 114.

hbox2 = wx.BoxSizer(wx.HORIZONTAL)
hbox2.Add(draw_btn,  0, wx.TOP | wx.BOTTOM, 6)

hbox = wx.BoxSizer(wx.HORIZONTAL)
hbox.Add(hbox2, 0, wx.ALIGN_RIGHT)

vbox = wx.BoxSizer(wx.VERTICAL)
vbox.Add(hbox, 0, wx.EXPAND | wx.LEFT | wx.RIGHT | wx.BOTTOM, 10)

self.SetSizer(vbox)
```

GUI application demo — pydraw

Basically, ‘pydraw’ is designed to demo the multi-threading feature of Python, this is illustrated by button ‘Start’, when invoked, random drawing begins as Checkbox with ‘Random Walker’ option.

```
# in 'Random Walker' mode, 'Start' button is binded
# with the below event: _drawCircles()
# in which, RandomWalker instance is first created,
# then transfer the created instance to WorkerThread class
# to create drawing thread, execute start() method to trig
# drawing process
# details see: file models.py line 102 to 163

def _drawCircles(self):
    randomModel = models.RandomWalker(self, self.particles,
                                      self.buffer, self.color,
                                      200, self.stop_ms)
    self.randomModel_worker = models.WorkerThread(randomModel)
    self.randomModel_worker.start()
```

GUI application demo — PYDRAW

To make mode of particles, `Particle` class, p = Particle()

- pos: p.x, p.y, color: p.color, circle: p.radius,
- type: p.type, electron, proton, etc.
- speed: p.v, to simulate physics procedure, weight, p.w, etc.
- as many as you can imagine

```
1  # here, particle is modeled to be colorful circle
2  # in Cartesian coordinates,
3  # x, y, r, and color should be the basic properties
4  # details see: file models.py line 17 t0 70
5  class Particle(object):
6      def __init__(self, x, y, radius=1.0, color=None):
7          self._x, self._y, self._r = x, y, radius
8          if color is None:
9              self._color = wx.Colour(0,0,255,200)
10         else:
11             self._color = color
12     @property
13     def color(self):
14         return self._color
15     @color.setter
16     def color(self, color):
17         self._color = color
18 # same apply to x,y,radius
```

GUI application demo — PYDRAW

Number crunching is handled by `Simulator` class, and two subclasses: `RandomWalker` and `RobotDrawer`, `move()` method define the behavior for each particle, complex manipulation is possible (e.g. particle tracking).

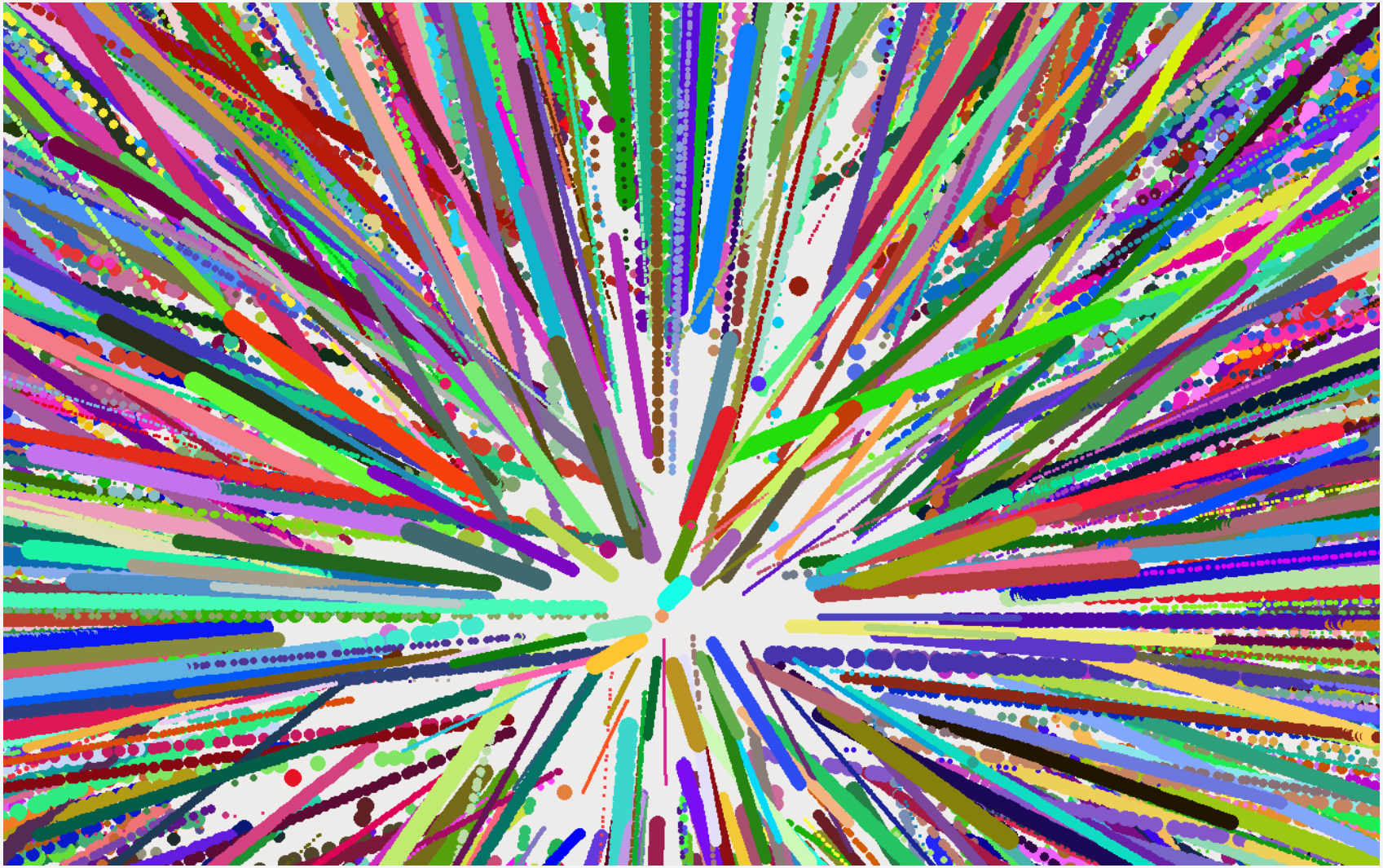
```
# just simple arithmetic
# e.g.:
# details see: file models.py line 72 to 194

# RandomWalker:
def _move_cop(self):
    for idx, p in enumerate(self.other_particles):
        p.x -= self.delx[idx]
        p.y -= self.dely[idx]

# RobotDrawer:
def move(self, n=None):
    for i, p in enumerate(self.particles):
        p.x += self.r_omega[i] * np.sin((self.theta[i]
                                - self.omega[i] * n))
        p.y -= self.r_omega[i] * np.cos((self.theta[i]
                                - self.omega[i] * n))
```

GUI application demo — PYDRAW

Initialized particles (2000)

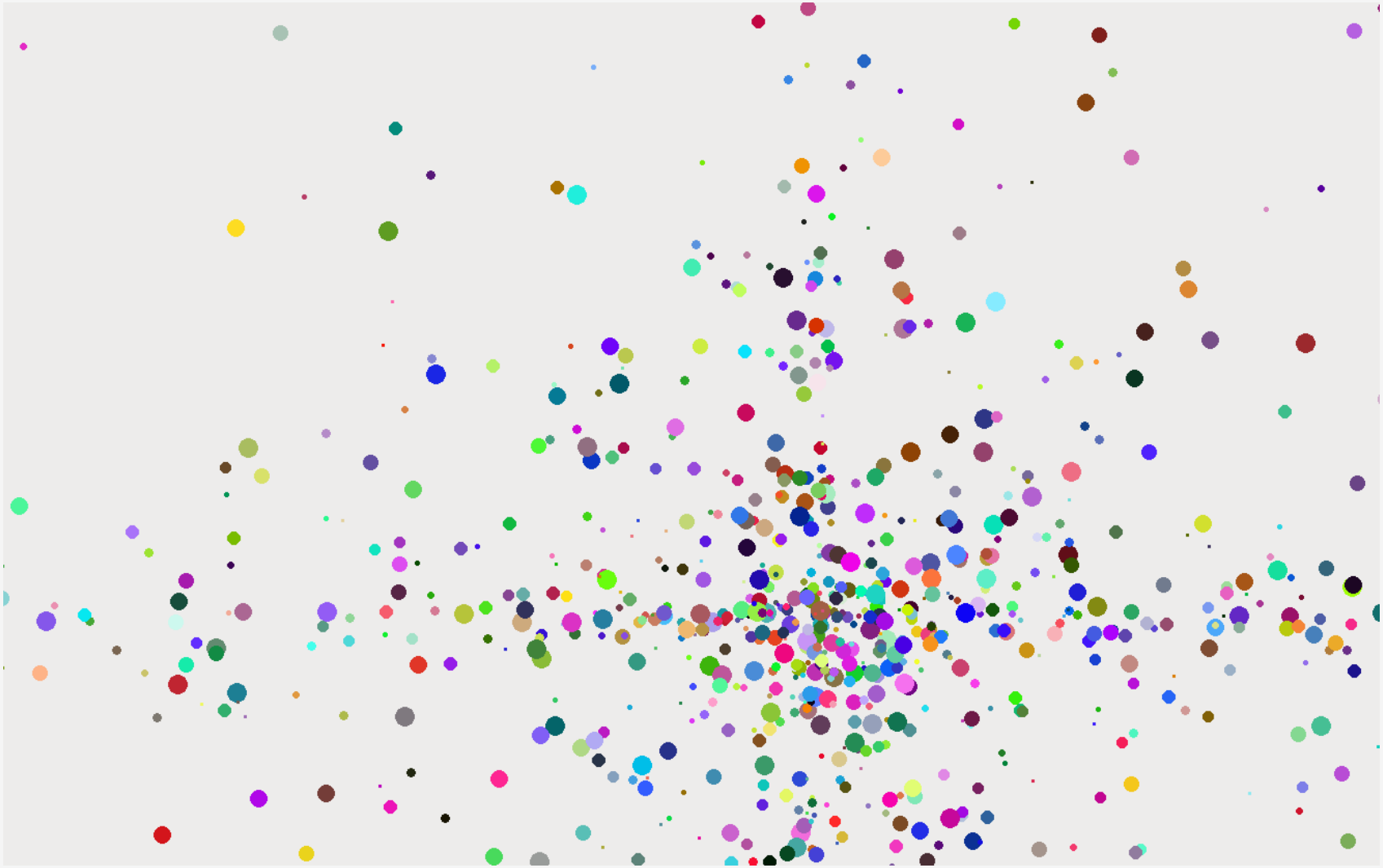
GUI application demo — PYDRAW

Random Walker with trace



GUI application demo — PYDRAW

Random Walker (converge)



Robot Drawer (circles)

Have Fun!