Simple Graphics

Ivan Sutherland

Turing Award winner Ivan Sutherland (who now lives in Portland) created the first graphical user interface as part of his 1963 MIT doctoral thesis on Sketchpad. This image, however, also features the DEC PDP-1.

The Graphics Model

- The Portable Graphics Library (pgl.py) uses a model based on the metaphor of a collage.
- A collage is similar to a child’s felt board that serves as a backdrop for colored shapes that stick to the felt surface. As an example, the following diagram illustrates the process of adding a blue rectangle and a red oval to a felt board:

  - Note that newer objects can obscure those added earlier. This layering arrangement is called the stacking order.

The Coordinate System

- Positions and distances on the screen are measured in terms of pixels, which are the small dots that cover the screen.
- Unlike traditional mathematics, the graphics library defines the origin to be in the upper left corner. Values for the y coordinate increase as you move downward.

The BlueRectangle Program

```python
def BlueRectangle():
gw = GWindow(WINDOW_WIDTH, WINDOW_HEIGHT)
rect = GRect(150, 50, 200, 100)
rect.setColor("Blue")
gw.add(rect)
```

`GWIndow_WIDTH = 500;`  
`GWIndow_HEIGHT = 200;`

`def BlueRectangle():`  
`gw = GWindow(WINDOW_WIDTH, WINDOW_HEIGHT)`  
`rect = GRect(150, 50, 200, 100)`  
`rect.setColor("Blue")`  
`gw.add(rect)`

SPACEWAR!

Eric Roberts  
CSCI 121  
September 11, 2019

Eric Roberts  
CSCI 121  
September 11, 2019
References

- Because objects can have complex internal structure, it is not always easy to represent them in diagrams that trace program execution.
- In many of the examples you will see in both the book and these slides, objects are represented using diagrams that seek to capture the conceptual nature of the object’s value.
- In Python, all objects are represented as references, which are values internal to the computer that serve as links to the data in the actual objects.
- In those cases in which it is important to emphasize the fact that objects are stored as references, those references appear in diagrams as arrows pointing from a variable to the object that contains the actual data. The next slide uses the pointer model to trace the execution of the BlueRectangle program.

Systems of Classification

- In the mid-18th century, the Scandinavian botanist Carl Linnaeus revolutionized the study of biology by developing a new system for classifying plants and animals in a way that revealed their structural relationships and paved the way for Darwin’s theory of evolution a century later.
- Linnaeus’s contribution was to recognize that organisms fit into a hierarchy in which the placement of individual species reflects their anatomical similarities.

Instances vs. Patterns

- In thinking about any classification scheme—biological or otherwise—it is important to draw a distinction between a class and specific instances of that class. In the most recent example, the designation *Iridomyrmex purpureus* is not itself an ant, but rather a class of ants. There can be (and usually are) many ants, each of which is an individual of that class.
- Each of these red ants is an instance of a particular class of ants. Each instance of the species *purpureus*, the genus *Iridomyrmex*, the family *Formicidae* (which makes it an ant), and so on. Thus, each ant is not only an ant, but also an insect, an arthropod, and an animal.

Biological Class Hierarchy

- Every red ant is also an animal, an arthropod, and an insect, as well as the other superclasses in the chain.

The GObject Hierarchy

- The classes that represent graphical objects form a hierarchy, part of which looks like this:

- The *GObject* class represents the collection of all graphical objects. The *GFillableObject* class unifies those objects that have a fillable interior.
Creating a `GWindow` Object

- The first step in writing a graphical program is to create a window using the following function declaration, where `width` and `height` indicate the size of the window:

  ```python
  gw = GWindow(width, height)
  ```

- The following operations apply to a `GWindow` object:

  ```python
  gw.add(object)  # Adds an object to the window.
  gw.add(object, x, y)  # Adds an object to the window after first moving it to (x, y).
  gw.remove(object)  # Removes the object from the window.
  gw.getWidth()  # Returns the width of the graphics window in pixels.
  gw.getHeight()  # Returns the height of the graphics window in pixels.
  ```

Operations on the `GObject` Class

- The following operations apply to all `GObject`s:

  ```python
  object.getX()  # Returns the x coordinate of this object.
  object.getY()  # Returns the y coordinate of this object.
  object.getWidth()  # Returns the width of this object.
  object.getHeight()  # Returns the height of this object.
  object.setColor(color)  # Sets the color of the object to the specified color.
  ```

- All coordinates and distances are measured in pixels.
- Each color is a string, such as "Red" or "White". The names of the standard colors are defined in Chapter 3.

Drawing Geometrical Objects

Functions to create geometrical objects:

- `GRect(x, y, width, height)`
  Creates a rectangle whose upper left corner is at (x, y) of the specified size.

- `GOval(x, y, width, height)`
  Creates an oval that fits inside the rectangle with the same dimensions.

- `GLine(x0, y0, x1, y1)`
  Creates a line extending from (x0, y0) to (x1, y1).

Methods shared by the `GFillableObject` subclasses:

- `object.setFilled(fill)`
  If `fill` is `True`, fills in the interior of the object; if `False`, shows only the outline.

- `object.setFillColor(color)`
  Sets the color used to fill the interior, which can be different from the border.

The `GRectPlusGOval` Program

```python
def GRectPlusGOval():
    gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
    rect = GRect(150, 50, 200, 100)
    rect.setFilled(True)
    rect.setColor("Blue")
    gw.add(rect)
    oval = GOval(150, 50, 200, 100)
    oval.setFilled(True)
    oval.setColor("Red")
    gw.add(oval)
```

The `DrawDiagonals` Program

```python
GWINDOW_WIDTH = 500
GWINDOW_HEIGHT = 200

def DrawDiagonals():
    gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
    gw.add(GLine(0, 0, GWINDOW_WIDTH, GWINDOW_HEIGHT))
    gw.add(GLine(0, GWINDOW_HEIGHT, GWINDOW_WIDTH, 0))
```

The `GLabel` Class

- You can display a string in the graphics window using the `GLabel` class, as illustrated by the following function that displays the string "hello, world" on the graphics window:

```python
def GraphicsHelloWorld():
    gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
    msg = GLabel("hello, world", 50, 100)
    gw.add(msg)
```
Operations on the GLabel Class

Function to create a GLabel

\[ \text{GLabel}(\text{text}, x, y) \]

Creates a label containing the specified text that begins at the point \((x, y)\).

Methods specific to the GLabel class

\[ \text{label}.\text{setFont}(\text{font}) \]

Sets the font used to display the label as specified by the font string.

The font is a string composed of the following components:
- The font style, which is usually missing or italic.
- The font weight, which is usually missing or bold.
- The font size, which is a number followed by a suffix indicating the units (typically pt, px, or em).
- The font family, which is the name of the font. Because the set of fonts differs on different machines, the family is usually a sequence of fonts separated by commas, which typically ends with a standard family: serif, sans-serif, or monospace.

The Geometry of the GLabel Class

- The GLabel class relies on a set of geometrical concepts that are derived from classical typesetting:
  - The baseline is the imaginary line on which the characters rest.
  - The origin is the point on the baseline at which the label begins.
  - The height of the font is the distance between successive baselines.
  - The ascent is the distance characters rise above the baseline.
  - The descent is the distance characters drop below the baseline.

- You can apply the getWidth, getHeight,getAscent, and getDescent methods to determine the geometrical properties of a GLabel.

Using a Larger Font

```
def BiggerHelloWorld():
gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
msg = GLabel('hello, world', 50, 100)
msg.setFont('36pt Times New Roman')
gw.add(msg)
```

Using a Fancier Font

```
def FancyHelloWorld():
gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
msg = GLabel('hello, world', 50, 100)
msg.setFont('36pt Palatino Blackletter')
gw.add(msg)
```

Centering a GLabel

- When you add a GLabel to the graphics window, it is often useful to center that label, either on the window as a whole or within some shape.
- Before you can figure out where to place the label, you need to know its dimensions. Unfortunately, you can’t determine the dimensions until after you have created the label.
- The strategy for centering a label at the point \((x, y)\) is then:
  - Create a GLabel without setting its location.
  - Call the setFont method to set the desired font.
  - Determine the horizontal position of the origin by subtracting half the width of the label from \(x\).
  - Determine the vertical position of the baseline by adding half the ascent of the font.
  - Add the GLabel at the newly computed position.

The CenteredHelloWorld Program

```
def CenteredHelloWorld():
gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
msg = GLabel('hello, world')
msg.setFont('50pt sansserif')
x = (gw.getWidth() - msg.getWidth()) / 2
y = (gw.getHeight() + msg.getAscent()) / 2
fw.add(msg, x, y)
```
Program to Draw a Checkerboard

```c
def CheckeredBoard():
    gw = Window(WINDOW_WIDTH, WINDOW_HEIGHT)
    sq = (gw.get澠().width + SQUARE_SIZE) / 2
    for row in range(NROWS):
        for col in range(N_COLUMNS):
            x = col * SQUARE_SIZE
            y = row * SQUARE_SIZE
            sqr = Square(x, y, SQUARE_SIZE, SQUARE_SIZE)
            sqr.setFill((row + col) % 2 == 0)
            gw.add(sqr)
```

Checkerboard