Eric Roberts
CSCI 121

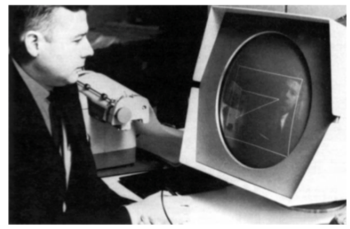
Handout #5
February 4, 2019

Simple Graphics

Simple Graphics

Eric Roberts
CSCI 121
February 4, 2018

Ivan Sutherland

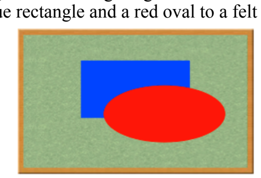


Turing Award winner Ivan Sutherland (who now lives in Portland) created the first graphical user interface as part of his 1963 MIT doctoral thesis on *Sketchpad*. This image, however, also features the DEC PDP-1. . . .

SPACEWAR!

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:

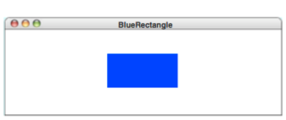


- Note that newer objects can obscure those added earlier. This layering arrangement is called the ***stacking order***.

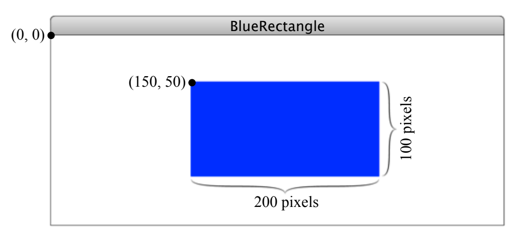
The `BlueRectangle` Program

```
def BlueRectangle():
    gw = GWindow(500, 200)
    rect = GRect(150, 50, 200, 100)
    rect.setColor("Blue")
    rect.setFilled(True)
    gw.add(rect)
```

rect



The Coordinate System



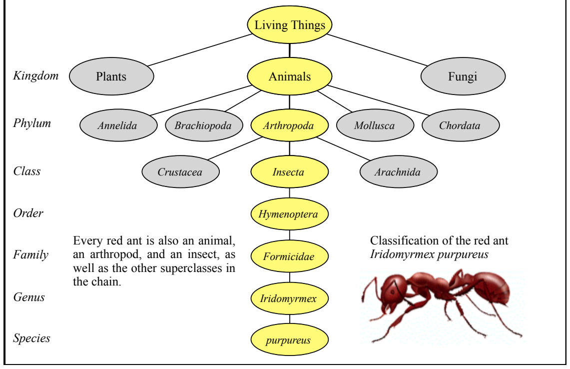
- Positions and distances on the screen are measured in terms of ***pixels***, which are the small dots that cover the screen.
- Unlike traditional mathematics, the graphics library defines the ***origin*** to be in the upper left corner. Values for the *y* coordinate increase as you move downward.

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.

Carl Linnaeus (1707–1778)

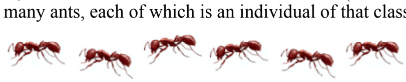
Biological Class Hierarchy



Instances *vs.* Patterns

Drawn after you, you pattern of all those.
—William Shakespeare, Sonnet 98

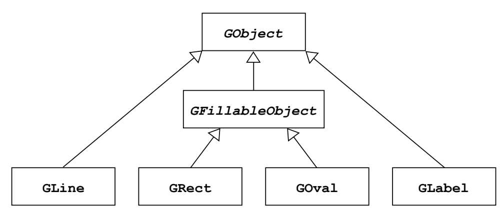
- In thinking about any classification scheme—biological or otherwise—it is important to draw a distinction between a class and specific instances of that class. In the most recent example, the designation *Iridomyrmex purpureus* is not itself an ant, but rather a ***class*** of ant. There can be (and 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 is 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.

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:

gw = GWindow(*width***,** *height***)**

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

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:

*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(` x_0`,` y_0`,` x_1`,` y_1`)`
Creates a line extending from (x_0, y_0) to (x_1, y_1).

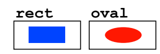
Methods shared by the `GFillableObject` subclasses:

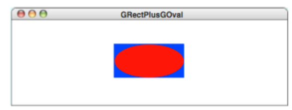
object`.setFilled(`*fill*`)`
If *fill* is `True`, fills in the interior of the object; if `False`, shows only the outline.

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

The `GRectPlusGOval` Program

```
def GRectPlusGOval():
    gw = GWindow(500, 200)
    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)
```



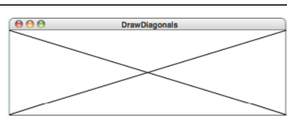


The `DrawDiagonals` Program

```
# Constants
GWINDOW_WIDTH = 500
GWINDOW_HEIGHT = 200

# Function: DrawDiagonals

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:

```
def HelloWorld():
    gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
    label = GLabel("hello, world", 100, 75)
    label.setFont("36px 'Helvetica Neue','SansSerif'")
    label.setColor("Red")
    gw.add(label)
```

Operations on the `GLabel` Class

Function to create a `GLabel`

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

Methods specific to the `GLabel` class

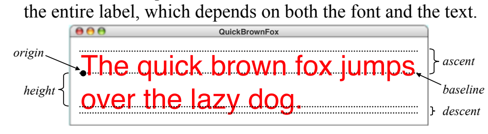
label`.setFont(`*font*`)`
Sets the font used to display the label as specified by the font string.

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 machine, the family is usually a sequence of fonts separated by commas, which typically ends with a standard family: `serif`, `sans-serif`, or `monospaced`.

The Geometry of the `GLabel` Class

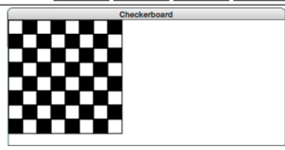
- The `GLabel` class relies on a set of geometrical concepts that are derived from classical typesetting:
  - The ***baseline*** is the imaginary line on which the characters rest.
  - The ***origin*** is the point on the baseline at which the label begins.
  - The ***height*** of the font is the distance 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 use the `getHeight`, `getAscent`, and `getDescent` methods to determine the corresponding property of the font. You can use the `getWidth` method to determine the width of the entire label, which depends on both the font and the text.



Program to Draw a Checkerboard

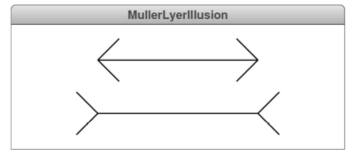
```
def Checkerboard():
    gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
    for i in range(N_ROWS):
        y = i * SQUARE_SIZE
        for j in range(N_COLUMNS):
            x = j * SQUARE_SIZE
            sq = GRect(x, y, SQUARE_SIZE, SQUARE_SIZE)
            sq.setFilled((i + j) % 2 != 0)
            gw.add(sq)
```

gw	i	j	x	y	sq



Exercise: The Müller-Lyer Illusion

- Exercise 10 on page 114 describes the ***Müller-Lyer illusion,*** which asks the viewer which of the two horizontal lines is longer in the following figure:



- Work with your neighbor to write the program to create this image. As a first step, figure out what constants you need to define to control the sizes and positioning of the components.