Carnegie Mellon University in Qatar

CMU Graphics Guide v2
Including General 15-112 Notes

Authors: Jacob, Ebil; Hajizada, Salman.
Co-Author: Kumar, Ravin.
Reviewed by: Kumar, Ravin; Shikfa, Mohamed.

Publication date:
2023

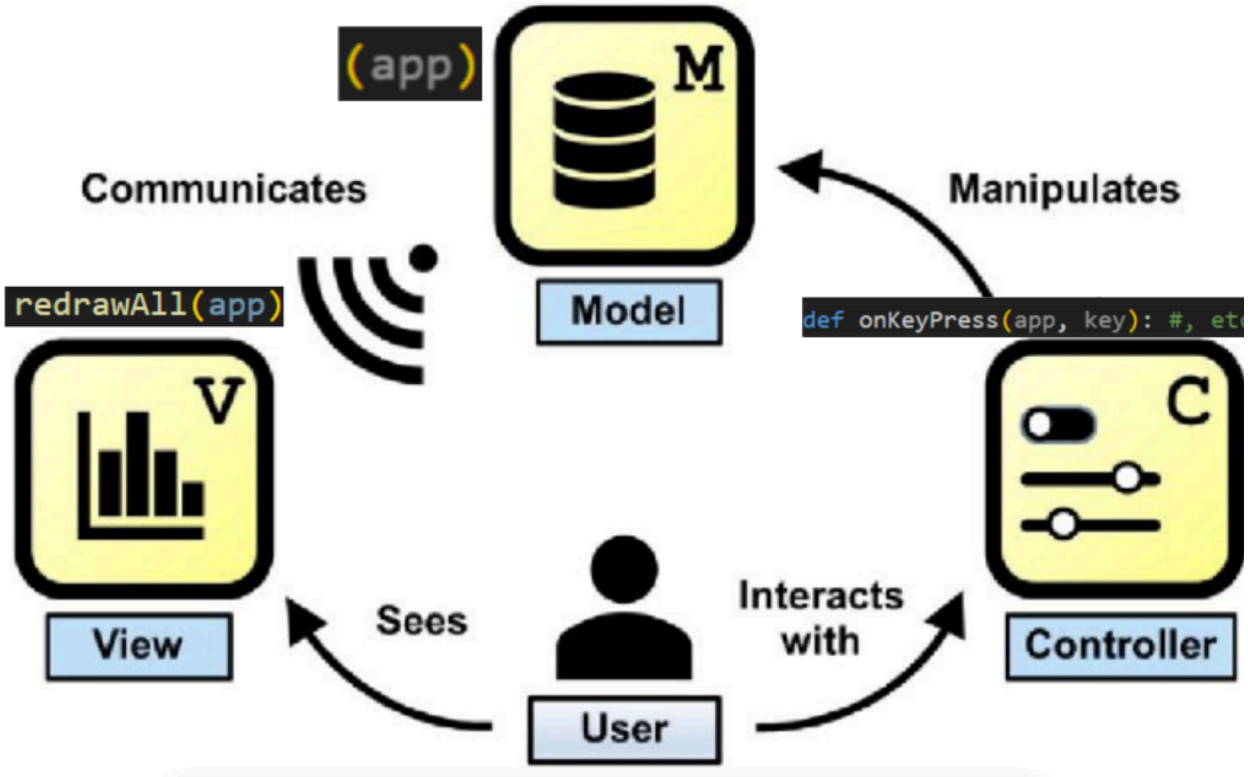
Document Version
Peer-reviewed version

Citation for published version (APA):
Jacob, E., & Hajizada, S. (2023). *CMU Graphics Documentation*. Carnegie Mellon Student Publications (No. 1)

Table of Contents

Model View Controller Outline

MVC (Model View Controller) is an approach used in graphics and animation.



View

- Draws elements
- Cannot change the model
- Is never called directly

Model

- Stores all data for the program
- Updated by controllers

Controllers

- Can change the data in the model
- Cannot change the view
- Are never called directly

MVC Violation

- Breaking any of the rules above results in a runtime error
- This crashes the program

Model

As mentioned previously, the model stores the information of the app.

It is stored inside the app object and its attributes are accessed through app.[attribute]

Some attribute names are "reserved" i.e. you cannot name an attribute using these names.

```
#reserved attributes
app.width            # Width of canvas
app.height           # Height of canvas
app.stepsPerSecond   # Steps per second (see controllers
                     # for more). Default is 30
app.background      # Sets background color
```

Otherwise, you can use any attribute name you want

```
#other
app.x
app.y
app.camelCase
app.hello123
```

Controllers

These functions are never called directly. Each has a specific purpose.

```
# Must have the same order and number of parameters

def onAppStart(app):
    """
    Runs once when app starts
    """

def onMousePress(app, mouseX, mouseY):
    """
    Runs every time a mouse is clicked
    """

#passes mouse coordinates into parameters
def onMouseRelease(app, mouseX, mouseY):
    """
    Runs every time a mouse is released
    """

def onMouseMove(app, mouseX, mouseY):
    """
    Runs when mouse isn't pressed and moved
    """

def onMouseDrag(app, mouseX, mouseY):
    """
    Runs when mouse is pressed and moved
    """
```

```
# key is passed into parameters (case-sensitive)
# escape, left, right, up, down, space (shift doesn't
display)
def onKeyPress(app, key):
    """
    Runs when a key is pressed
    """


def onKeyHold(app, keys : list):
    """
    Runs when keys are held down
    """


def onKeyRelease(app, key):
    """
    Runs when a key is released
    """


def onStep(app):
    """
    Runs a certain amount of times per second
    Frequency is determined by app.stepsPerSecond
    """
```

The name of the parameters could be changed but the order and number cannot

View

The view and all of its helper functions are called inside this function:

```
def redrawAll(app):
```

Drawing functions

```
    drawRect(left, top, width, height,
             fill="color", border="color",
             borderWidth=2, opacity=100,
             rotateAngle=0)
    # draws it from the left-top corner of the rect

    drawLabel(string, centerX, centerY,
              size=12, font="arial",
              bold=False, italic=False,
              fill="color", border="color",
              borderWidth=2, opacity=100,
              rotateAngle=0)

    drawOval(centerX, centerY, width, height,
             fill="color", border="color",
             borderWidth=2, opacity=100,
             rotateAngle=0)

    drawCircle(centerX, centerY, radius,
               fill="color", border="color",
               borderWidth=2, opacity=100,
               rotateAngle=0)
```

```
drawLine(x1, y1, x2, y2,
         lineWidth=1,
         dashes=False or tuple[dashWidth, dashGap],
         arrowStart=False, arrowEnd=False,
         fill = "color", border = "color", opacity=100)
# x1, y1 are start of line, 2 is end of line
# adds arrow at start or end of line


drawRegularPolygon(centerX, centerY, radius, noOfPoints,
                   fill = "color", border = "color",
                   borderWidth=2, opacity=100,
                   rotateAngle=0)
# "radius is the distance from the center to any vertex"
# (R. Kumar, personal communication, September 25, 2023)



drawStar(centerX, centerY, radius, noOfPoints,
         roundness=0, fill = "color", border = "color",
         borderWidth=2,opacity=100, rotateAngle=0)
# roundness is 0 to 100
# if roundness == 100 then the star looks like a polygon
  with noOfPoints * 2 sides

drawArc(centerX, centerY, width, height,
        startAngle, sweepAngle,
        fill = "color", border = "color", borderWidth=2,
        opacity=100, rotateAngle=0)
# since arc is a part of an oval, first 4 args are the
  same as oval
# angles are in anti-clockwise direction
# sweepAngle = 360 fills the entire arc
```

```
getImageSize(url)
# returns a tuple with x,y dimensions
drawImage(url, left, top, width=int, height=int)
```

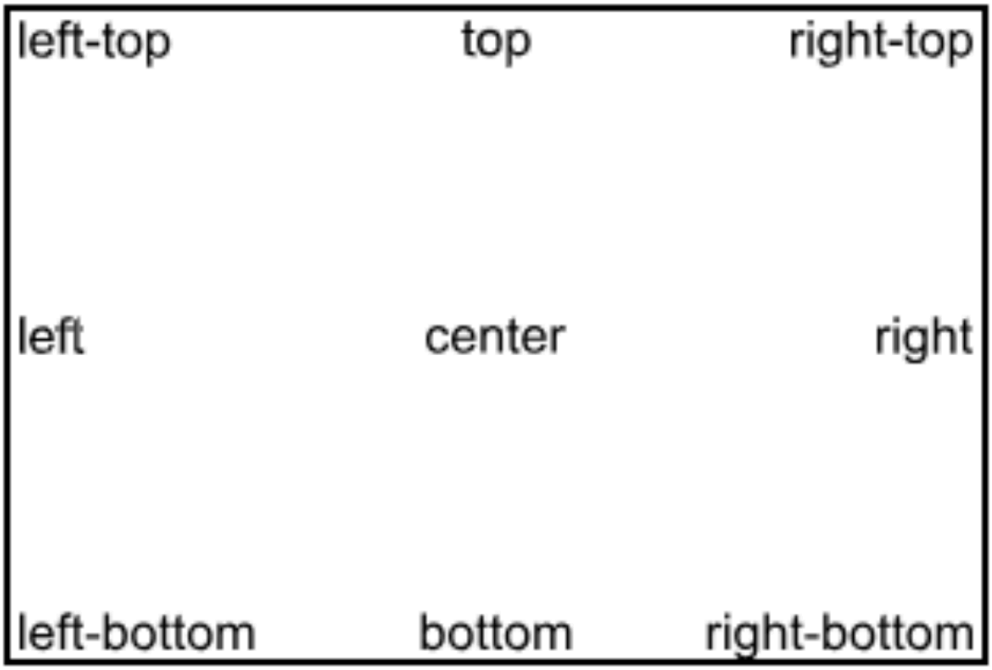
Notice how the coordinates specified in the parameters typically represent where the center of the shape will be located, except rectangles where it shows its top-left corner and line where it shows the two endpoints..

Alignment

As mentioned above, most shapes auto-align to the center, i.e. the center of the shape is drawn at the coordinates. This can be changed with another property:

```
align = "direction"
```

The value can be determined with this diagram:



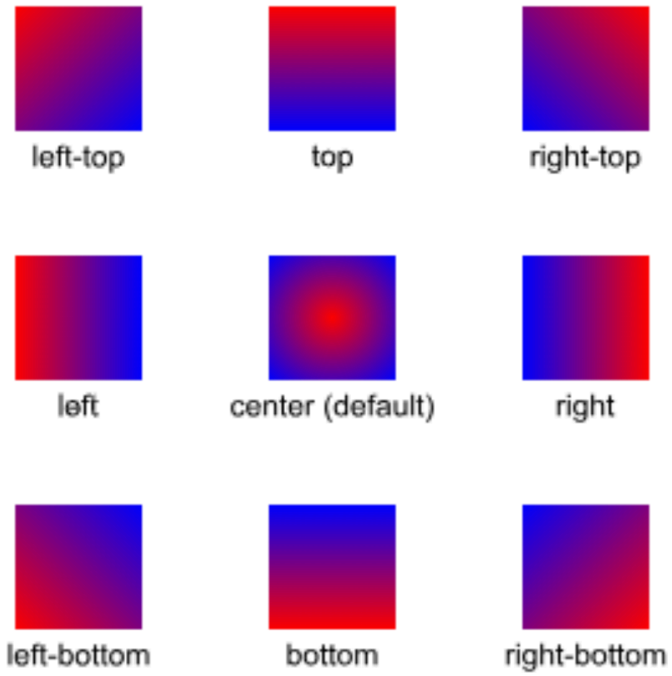
Color

There are a number of built-in colors that can be used, such as "red", "blue", "yellow", etc. However, if you want a specific color, the below function should be used:

```
rgb(redValue, greenValue, blueValue)
# Each value is an int 0 - 255 inclusive
```

If instead of one color you would want a gradient, the following function should be used:

```
gradientColor = gradient(color1, color2, ...,
                         start="center")
# Produces a gradient using several colors in a certain
  direction
# At least 2 colors required
# (Kumar, personal communication, September 25, 2023)
```



Miscellaneous built-in functions

```
# creates a sound object
track = Sound("url")
# plays a sound object
track.play(loop=False, restart = False)
# pauses sound
track.pause()

# returns distance between points (x1, y1) and (x2, y2)
distance(x1, y1, x2, y2) -> distance (pixels)

# returns angle from point (x1, y1) to (x2, y2), 0 being
# straight up
angleTo(x1, y1, x2, y2) -> angle (degrees)

# returns a point a number of pixels from (x1, y1)
# in the given direction(angle)
getPointInDir(x1, y1, angle, length) -> (x2, y2)

# returns a properly rounded value
rounded(float)

# creates a list with x rows and y columns
makeList(x, y)
```

Additional 15-112 notes

Collision detection

- Point - Circle

  If distance from center <= radius, point is inside circle

```
# Takes point coordinate x, y
# Returns True if point inside circle of
# Center cx, cy. Radius r
def isPointInCircle(x, y, cx, cy, r):
    if distance(x, y, cx, cy) <= r:
        return True
    else:
        return False
```

- Circle - Circle

  The circles intersect if the distance between their centers is no greater than the sum of their radii.

```
# Returns true if circle 1 and circle 2 intersect
# Circle 1 - center x1, y1, radius r1
# Circle 2 - center x2, y2, radius r2
def circleCollision(x1, y1, r1, x2, y2, r2):
    if distance(x1, y1, x2, y2) <= r1 + r2:
        return True
    else:
        return False
```

- Rectangle-Rectangle

  The right edge of rectangle1 is on or to the right of the left edge of rectangle0. That is, (right1 >= left2).

  The right edge of rectangle0 is on or to the right of the left edge of rectangle1. That is, (right2 >= left1).

  The bottom edge of rectangle1 is on or below the top edge of rectangle0. That is, (bottom1 >= top2).

  The bottom edge of rectangle0 is on or below the top edge of rectangle1. That is, (bottom2 >= top1)

```
#Takes in dimensions of 2 rectangles
#Returns true if they overlap
def rectanglesOverlap(left1, top1, width1, height1,
                      left2, top2, width2, height2):
    bottom1 = top1 + height1
    bottom2 = top2 + height2
    right1 = left1 + width1
    right2 = left2+width2
    if bottom1 >= top2 and bottom2 >= top1 and\
       right1 >= left2 and right2 >= left1:
        return True
    else:
        return False
```

Pausing and taking steps

```
    def onStep(app):
        if not app.paused:
            takeStep(app)

    def takeStep(app):    # Keeps running if app is not
paused
        app.counter += 1

    def onKeyPress(app, key):
        if key == 'p':  # pauses the app
            app.paused = not app.paused
        elif key == 's' and app.paused:  # takes a step
            takeStep(app)
```

Timing events

```
    #30 times per second by default
    def onStep(app):
        app.timer += 1
        # If you want to run it 5 times per second
        desired = 5
        interval = app.stepsPerSecond // desired
        if app.timer % interval == 0:
            runCode()
```

List unpacking with polygons

```
    # store the points in a list:
    points = [100, 100, 50, 200, 300, 300, 250, 50]

    # use *points to unpack the values
    # and draw the polygon using this list:
```

```
drawPolygon(*points, fill='cyan', border='black')
```

2D List board

```
#an example code of how a board can be set up and drawn
# board variables (assume board starts at 0,0)
app.rows
app.cols
app.boardWidth
app.boardHeight

# Calculates cell size based on board dimensions
def getCellSize(app):
    width = app.boardWidth / app.cols
    height = app.boardHeight / app.rows
    return (width, height)

# Calculate coordinates of cell based on row and column
def getCellCoordinate(app, row, col):
    width, height = getCellSize(app)
    x = col * width
    y = row * height
    return (x, y)

# Draws a cell
def drawCell(app, row, col):
    x, y = getCellCoordinate(app, row, col)
    width, height = getCellSize(app)
    drawRect(x, y, width, height)

# Draws the board
def drawBoard(app):
    for row in range(app.rows):
        for col in range(app.cols):
            drawCell(app, row, col)
```

Get cell from coordinates

```
#Returns row and col selected based on coordinates
def getCell(app, x, y):
    dx = x - app.boardLeft
    dy = y - app.boardTop
    row = dy // app.cellHeight
    col = dx // app.cellWidth
    if (0 <= row < app.rows) and (0 <= col < app.cols):
        return (row, col)
    else: # if coordinates out of bounds
        return None
```

Select cell with mouse

```
#Stores the cell row and col selected with mouse
def onMousePress(app, x, y):
    selectedCell = getCell(app, x, y)
    if selectedCell != None:
        app.selection = selectedCell
    #If click is outside board, store None
    else:
        app.selection = None
```

Move cell selection with keys

```
# When in onKeyPress, passed in with different parameters
def moveSelection(app, drow, dcol): # e.g. (app, -1, 0) if up
    row, col = app.selection
    newRow = (row + drow) % app.rows
    newCol = (col + dcol) % app.cols
    app.selection = (newRow, newCol)
```

Animation questions strategy

1. Write the onStep function
    - Write down the logic without worrying about variables existing or not (e.g. you can write down '`app.cx += 1`' without app.cx existing)
    - You can also create any helper functions that you need
2. Write the redrawAll function
    - What needs to be drawn to the screen every frame?
    - You can try to sketch what you need to display to help
    - What could change based on the state of the program? A pause screen, a gameover screen?
    - Again, ignore variables not existing
3. Now write the onAppStart function
    - Having written the onStep and redrawAll function you should now know what variables you need to initialize
    - If the program should restart, write a helper function (e.g. restartApp) that sets the app values
4. Write any needed remaining controller functions
    - e.g. onMousePress, onKeyPress, onKeyHold, etc.
    - Should handle all input the program needs

References

CMU CS Academy. (n.d.). https://academy.cs.cmu.edu/docs

Bonus

If the distance function is too complicated to understand, we've devised a simpler way to calculate the distance without using lists. You can use the formula below:

```
def distance(x1, y1, x2, y2):
    return math.dist(range(x1, y1+1 if y1 > x1 else y1-1,
           y1-x1) if y1 - x1 != 0 else map(int, (str(x1) +
           " " + str(y1)).split()) , range(x2, y2+1 if y2 >
           x2 else y2-1, y2-x2) if y2 - x2 != 0 else
           map(int, (str(x2) + " " + str(y2)).split()))
```

...or just use this:

```
def distance(x1, y1, x2, y2):
    return math.dist(map(int, (str(x1) + " " +
           str(y1)).split()), map(int, (str(x2) + " " +
           str(y2)).split()))
```

! If anybody can explain what these functions do, you will receive a prize of 1 double-up drink of your choice. !