#11: Side-Scrolling and Time-Based Animation

SAMS SENIOR CS TRACK

Last Time

Used **input** and **output** to creative interactive applications.

Used **model**, **view**, and **event controllers** to create interactive graphics.

Today's Learning Goals

Use **side-scrolling** to recognize the difference between the model and the view.

Use a **time loop** to create animations in interactive graphics.

MVC and Side-scrolling

Side-scrolling

In video games, it is often the case that the world the game takes place in is bigger than the screen you play on. Video games use **side-scrolling** to let the character move around in that big world.

A classic example of this is Mario. Mario implements horizontal side-scrolling.

https://supermarioemulator.com/mario.php

Model vs View

To implement side-scrolling, you have to separate the **model** (the large world that Mario moves around in) from the **view** (the part of the world that you the player can see).

To do this, we keep track of most of the model in terms of that larger world. But we also keep track of two useful variables: **data.scrollX** and **data.scrollY**. These tell us where our screen is with respect to the world.

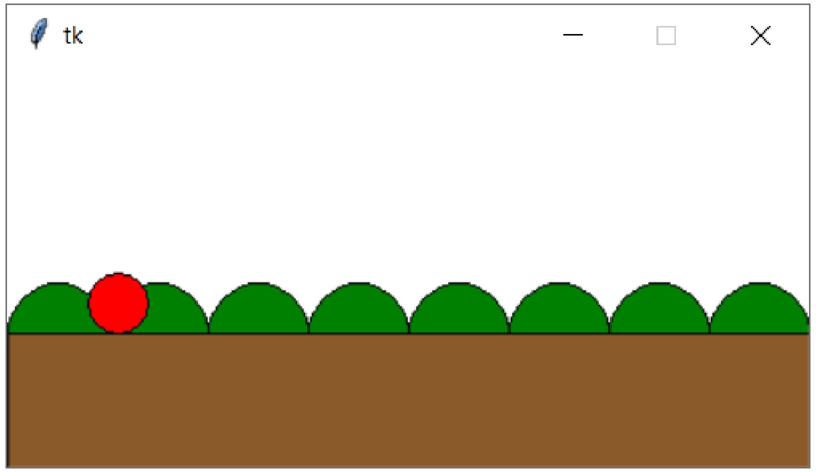
Let's use a paper demo as an example...

Side-scrolling: Setup

```
def init(data):
    data.mapWidth = data.width * 3
    data.bushes = []
    data.bushSize = 50
    for bushX in range(0, data.mapWidth, data.bushSize):
        data.bushes.append(bushX)
    data.playerX, data.playerSize = 40, 30
    data.groundY = data.height*2/3

def keyPressed(event, data):
    playerSpeed = 10
    if event.keysym == "Left":
        data.playerX -= playerSpeed
    elif event.keysym == "Right":
        data.playerX += playerSpeed

def redrawAll(canvas, data):
    for bushX in data.bushes:
        canvas.create_oval(bushX, data.groundY - data.bushSize/2,
                           bushX + data.bushSize, data.groundY + data.bushSize/2,
                           fill="green")
    canvas.create_rectangle(0, data.groundY, data.mapWidth, data.height, fill="tan4")
    canvas.create_oval(data.playerX, data.groundY - data.playerSize,
                       data.playerX + data.playerSize, data.groundY, fill="red")
```

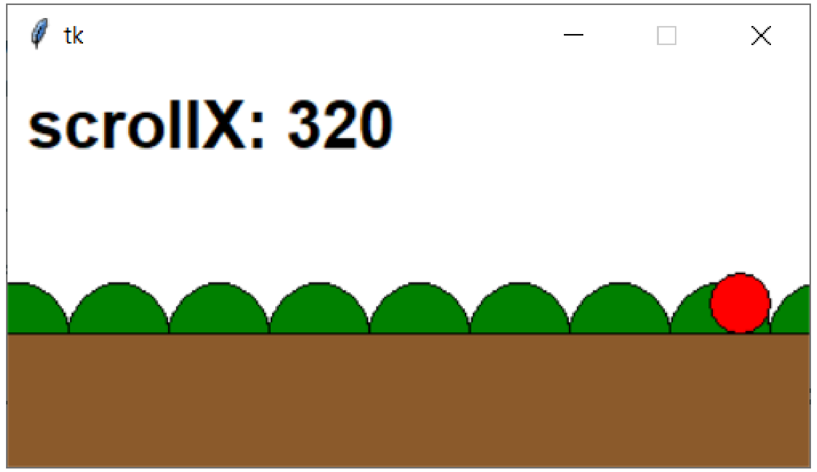


Side-scrolling: Player Movement

```
def init(data):
    ...
    data.scrollX = 0

def keyPressed(event, data):
    ...
    buffer = 10
    if (data.playerX + data.playerSize + buffer) >= (data.scrollX + data.width):
        data.scrollX += playerSpeed
    elif (data.playerX - buffer) <= data.scrollX:
        data.scrollX -= playerSpeed

def redrawAll(canvas, data):
    for bushX in data.bushes:
        canvas.create_oval(bushX - data.scrollX, data.groundY - data.bushSize/2,
                           bushX + data.bushSize - data.scrollX,
                           data.groundY + data.bushSize/2,
                           fill="green")
    canvas.create_rectangle(0 - data.scrollX, data.groundY,
                            data.mapWidth - data.scrollX, data.height, fill="tan4")
    canvas.create_oval(data.playerX - data.scrollX, data.groundY - data.playerSize,
                       data.playerX + data.playerSize - data.scrollX, data.groundY,
                       fill="red")
    canvas.create_text(10, 10, text="scrollX: " + str(data.scrollX),
                       font="Arial 25 bold", anchor="nw")
```



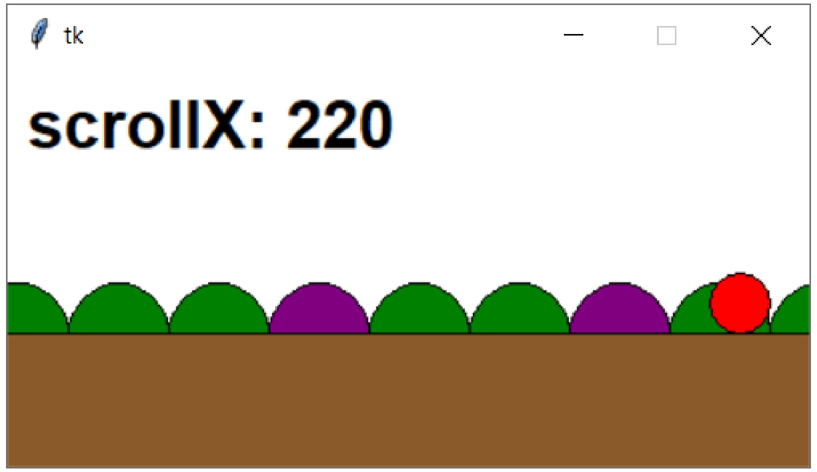
scrollX: 320

Side-scrolling: Mouse Location

```
def init(data):
    ...
    for bushX in range(0, data.mapWidth, data.bushSize):
        data.bushes.append([bushX, "green"])
    ...

def mousePressed(event, data):
    viewX = event.x
    x = data.scrollX + viewX
    y = event.y
    for bush in data.bushes:
        if bush[0] <= x <= (bush[0] + data.bushSize) and \
           (data.groundY - data.bushSize/2) <= y <= data.groundY:
            bush[1] = "purple"

def redrawAll(canvas, data):
    for bush in data.bushes:
        [bushX, color] = bush
        canvas.create_oval(bushX - data.scrollX,
                           data.groundY - data.bushSize/2,
                           bushX + data.bushSize - data.scrollX,
                           data.groundY + data.bushSize/2, fill=color)
    ...
```



Big Idea

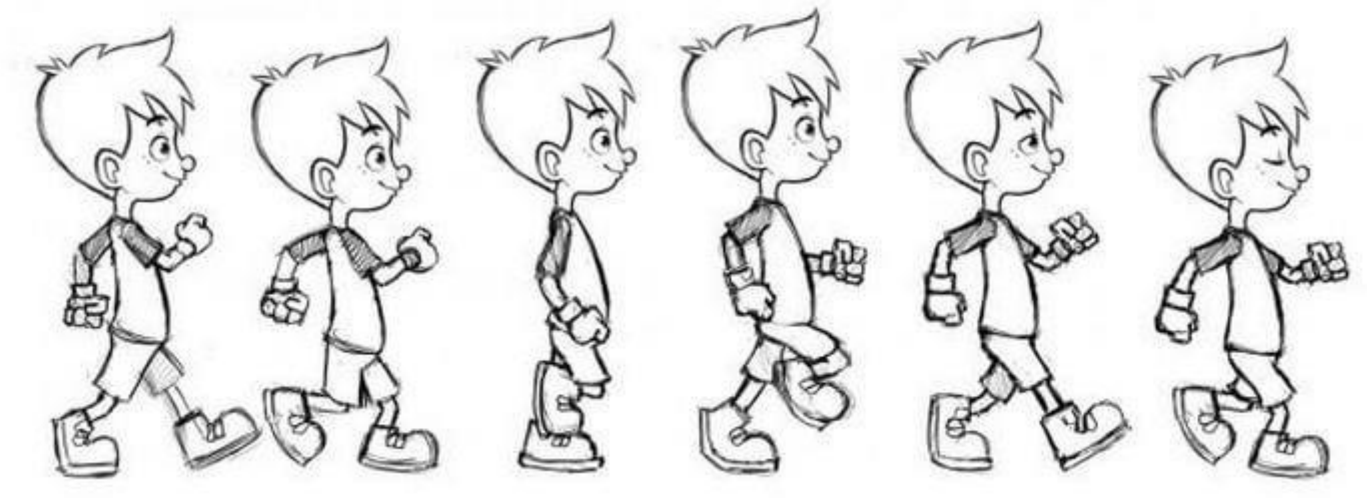
When writing interactive graphics code, always consider whether the data you're using is from the point of view of the **model** or the **view**.

Converting between the two will then be easy!

Time-Based Animation

Animation

Animation is the process of making graphics look like they are moving by changing them slightly as time passes. We can create animations in tkinter too!



Change Data Over Time

To animate graphics, we will add a **time loop** to our graphics framework. This time loop will call a new function, **timerFired(data)**, every **data.timerDelay** milliseconds. Therefore, the function will be called at a specific rate over time.

By changing data in timerFired, we can make the program state change over time! And if we set data.timerDelay to be a small number, like 100ms (or 10 times per second), the change will look continuous.

To add timerFired, we'll have to update our starter code:

http://krivers.net/15112-s19/notes/notes-animations-part2.html#starter-code

Example: tracking time passed

```
def init(data):
    data.timerDelay = 1000 # one second
    data.timeCount = 0

def keyPressed(event, data):
    if event.keysym == "Up":        data.timerDelay *= 2
    elif event.keysym == "Down":    data.timerDelay //= 2

def timerFired(data):
    data.timeCount += 1

def redrawAll(canvas, data):
    s = "Time Passed: " + str(data.timeCount) + "\n" + \
        "timerDelay: " + str(data.timerDelay)
    canvas.create_text(data.width/2, data.height/2, font="Arial 32 bold", text=s)
```

Example: moving shape

```
def init(data):
    data.boxXSpeed = 10
    data.boxX = 10
    data.boxY = data.height/2

def timerFired(data):
    data.boxX += data.boxXSpeed

def redrawAll(canvas, data):
    canvas.create_rectangle(data.boxX - 20, data.boxY - 20,
                            data.boxX + 20, data.boxY + 20, fill="green")
```

Example: different rates

```
def init(data):
    data.timerCount = 0
    data.circles = [ ]

def timerFired(data):
    data.timerCount += 1
    x, y = random.randint(0, data.width), random.randint(0, data.height)
    data.circles.append([x, y, "red"])
    if data.timerCount % 5 == 0:
        x, y = random.randint(0, data.width), random.randint(0, data.height)
        data.circles.append([x, y, "blue"])

def redrawAll(canvas, data):
    for circle in data.circles:
        [x, y, color] = circle
        canvas.create_oval(x - 5, y - 5, x + 5, y + 5, fill=color)
```

Today's Learning Goals

Use **side-scrolling** to recognize the difference between the model and the view.

Use a **time loop** to create animations in interactive graphics.