Models of Motion with Objects

- Linear Translation
- Bouncing
- Rotation
- Seeking a Target
- Gravity and Friction
- Accelerating toward a Target
- Perspective (starfield)

Components of the Main Sketch

1. A class that encapsulates all data (instance vars) and behavior (methods) for moving objects
2. A global list to hold all moving objects
3. An event-handler (function) that creates and stores a new instance of the moving object class
4. An event-handler (function) that updates and draws all existing objects

Outline for a Graphic Object Class

1. class statement
2. A constructor to initialize new objects
   - Including instance vars necessary to maintain object state
3. A step() method to update state of the object
   - Including object location, and any other fields desired
4. A display() method to draw object on the sketch

Setting up the Program

```
# BoxMaker.py

from Processing import *
window(500, 500)

# Contains all objects that have been created
boxes = []

rectMode(CENTER)

…
```

BoxMaker.py

Driving Program Operation

```
…
# Create a new box and add to the master list
def createBox(o, e):
    x, y = mouseX(), mouseY()
    b = Box( x, y )
    boxes.append( b )

# Draw all boxes
def draw(o, e):
    background(0)
    for b in boxes:
        b.step()
        b.display()

# Create a new box whenever the mouse is clicked
onMousePressed += createBox

# Draw all
frameRate(20)
onLoop += draw
loop()
```

BoxMaker.py

Class Definition for a Simple Box Object

```
# A class to create a simple Box
class Box:
    # Constructor that initializes position
    def __init__(self, x, y):
        self.x = x
        self.y = y

    # This method advances the state of the object
    def step(self):
        pass

    # This method draws the object in its current state
    def display(self):
        fill(200)
        rect(self.x, self.y, 20, 20)
```

BoxMaker.py

Linear Translation

```
class Mover:
    def __init__(self, x, y):
        self.x = x              # Position
        self.y = y
        self.vx = -5.0          # Velocity
        self.vy = 5.0

    def step(self):
        self.x = self.x + self.vx
        self.y = self.y + self.vy

    def display(self):
        fill(200)
        rect(self.x, self.y, 20, 20)
```

(Also change the class mentioned at create time.)

How can we keep the boxes on the sketch?

BoxMover.py

Linear Translation

```
class Mover:
    def __init__(self, x, y):
        self.x = x              # Position
        self.y = y
        self.vx = -5.0          # Velocity
        self.vy = 5.0

    def step(self):
        #self.x = self.x + self.vx
        #self.y = self.y + self.vy
        w, h = width(), height()
        self.x = (self.x + self.vx + w) % w
        self.y = (self.y + self.vy + h) % h

    def display(self):
        fill(200)
        rect(self.x, self.y, 20, 20)
```

BoxMover.py

Rotation

```
class Rotator:
    def __init__(self, x, y):
        self.x = x              # Position
        self.y = y
        self.angle = 0.0        # Angle of rotation

    def step(self):
        self.angle = self.angle + 0.05

    def display(self):
        fill(200)
        pushMatrix()
        translate(self.x, self.y)
        rotate(self.angle)
        rect(0, 0, 20, 20)
        popMatrix()
```

How can we make the box orbit a point instead of rotate?

BoxRotator.py

Seeking a Target

```
class Seeker:
    def __init__(self, x, y):
        self.x = x              # Position
        self.y = y
        self.targetX = random( width() )
        self.targetY = random( height() )

    def step(self):
        # Move toward the target
        self.x = self.x + 0.02 * (self.targetX - self.x)
        self.y = self.y + 0.02 * (self.targetY - self.y)

        # Reset the target if it gets too close
        if dist(self.x, self.y, self.targetX, self.targetY) < 40:
            self.targetX = random( width() )
            self.targetY = random( height() )

    def display(self):
        fill(200)
        pushMatrix()
        translate(self.x, self.y)
        rect(0, 0, 20, 20)
        popMatrix()
```

How can we visualize the target?

BoxSeeker.py

Gravity

```
class Dropper:
    def __init__(self, x, y):
        self.x = x              # Position
        self.y = y
        self.vX = 0.0           # Velocity
        self.vY = 0.0
        self.aY = 0.2           # Acceleration

    def step(self):
        if self.y <= height(): # Update position if dropping
            self.x = self.x + self.vX
            self.y = self.y + self.vY
            self.vY = self.vY + self.aY

    def display(self):
        fill(200)
        pushMatrix()
        translate(self.x, self.y)
        rect(0, 0, 20, 20)
        popMatrix()
```

BoxDropper.py

Gravity and Bounce (and Friction)

```
class Bouncer:
    def __init__(self, x, y):
        self.x = x              # Position
        self.y = y
        self.vX = 0.0           # Velocity
        self.vY = 0.0
        self.aY = 0.2           # Acceleration
        self.friction = 0.7 # Friction factor

    def step(self):
        if self.y <= height():  # Update position if on sketch
            self.x = self.x + self.vX
            self.y = self.y + self.vY
            self.vY = self.vY + self.aY
        else:                   # Bounce when hit
            self.y = height()
            self.vY = -self.friction*self.vY

    def display(self):
        fill(200)
        pushMatrix()
        translate(self.x, self.y)
        rect(0, 0, 20, 20)
        popMatrix()
```

BoxBouncer.py

Accelerate Toward a Target

```
class Accelerator:
    def __init__(self, x, y):
        self.x = x           # Position
        self.y = y
        self.vX = 0.0        # Velocity
        self.vY = 0.0
        self.aX = 0.0        # Acceleration
        self.aY = 0.0
        self.targetX = 0.0   # Target position
        self.targetY = 0.0
        self.friction = random(0.9, 1.0)

    def display(self):
        fill(200)
        pushMatrix()
        translate(self.x, self.y)
        rect(0, 0, 20, 20)
        popMatrix()
```

Box is accelerated toward the mouse

```
def step(self):
    # Accelerate toward target
    self.aX = 0.002 * (self.targetX - self.x)
    self.aY = 0.002 * (self.targetY - self.y)

    # Update velocity and position
    self.vX = self.vX + self.aX
    self.vY = self.vY + self.aY
    self.vX = self.friction*self.vX
    self.vY = self.friction*self.vY
    self.x = self.x + self.vX
    self.y = self.y + self.vY

    # Reset target to mouse
    self.targetX = mouseX()
    self.targetY = mouseY()
```

BoxAccelerator.py

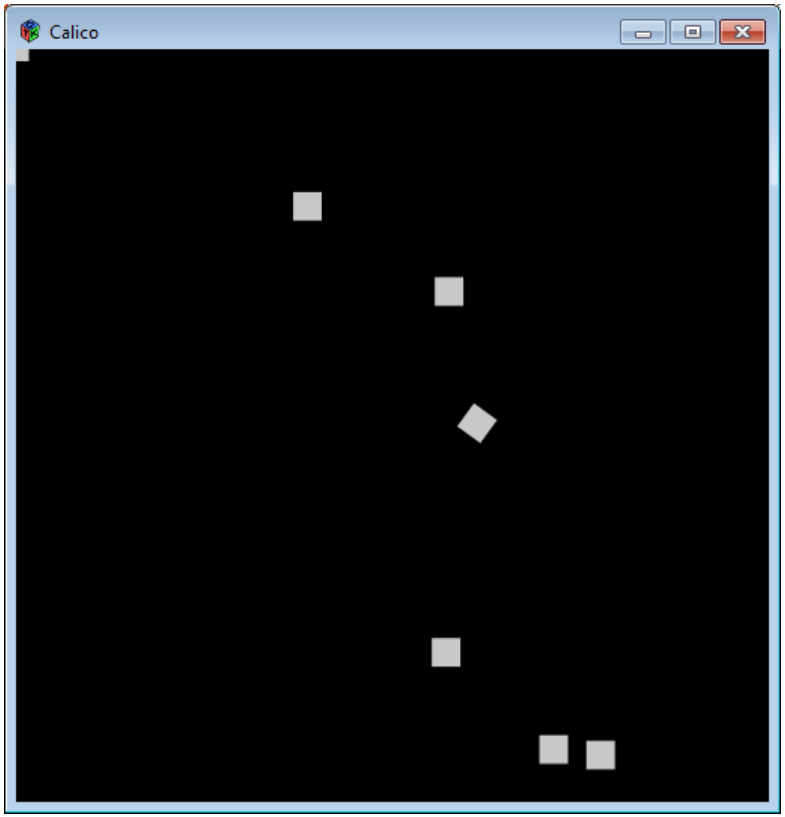
AllBoxes

```
class Box:
    …
class Mover:
    …
class Rotator:
   …
class Seeker:
    …
class Dropper:
    …
class Bouncer:
    …
class Accelerator:
    …

# Create boxes
w, h = width(), height()
boxes.append( Box( random(w), random(h) ) )
boxes.append( Mover( random(w), random(h) ) )
boxes.append( Rotator( random(w), random(h) ) )
boxes.append( Seeker( random(w), random(h) ) )
boxes.append( Dropper( random(w), random(h) ) )
boxes.append( Bouncer( random(w), random(h) ) )
boxes.append( Accelerator( random(w), random(h) ) )
```

AllBoxes.py

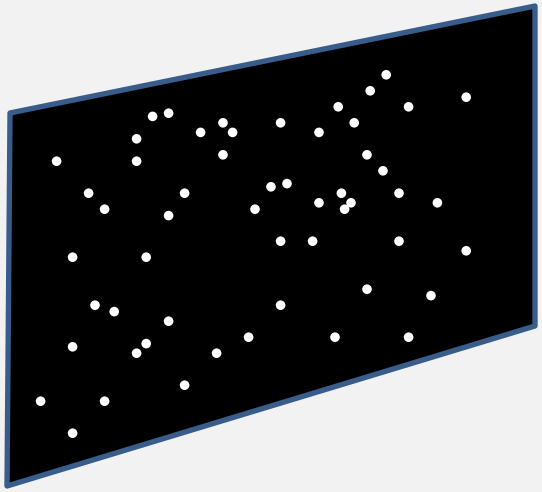
AllBoxes

- Note
  - Other field values can be changed instead, such as fill color, scale, width, height, …
  - The main program never changed.
  - Each object encapsulates its own behavior, so all can coexist.

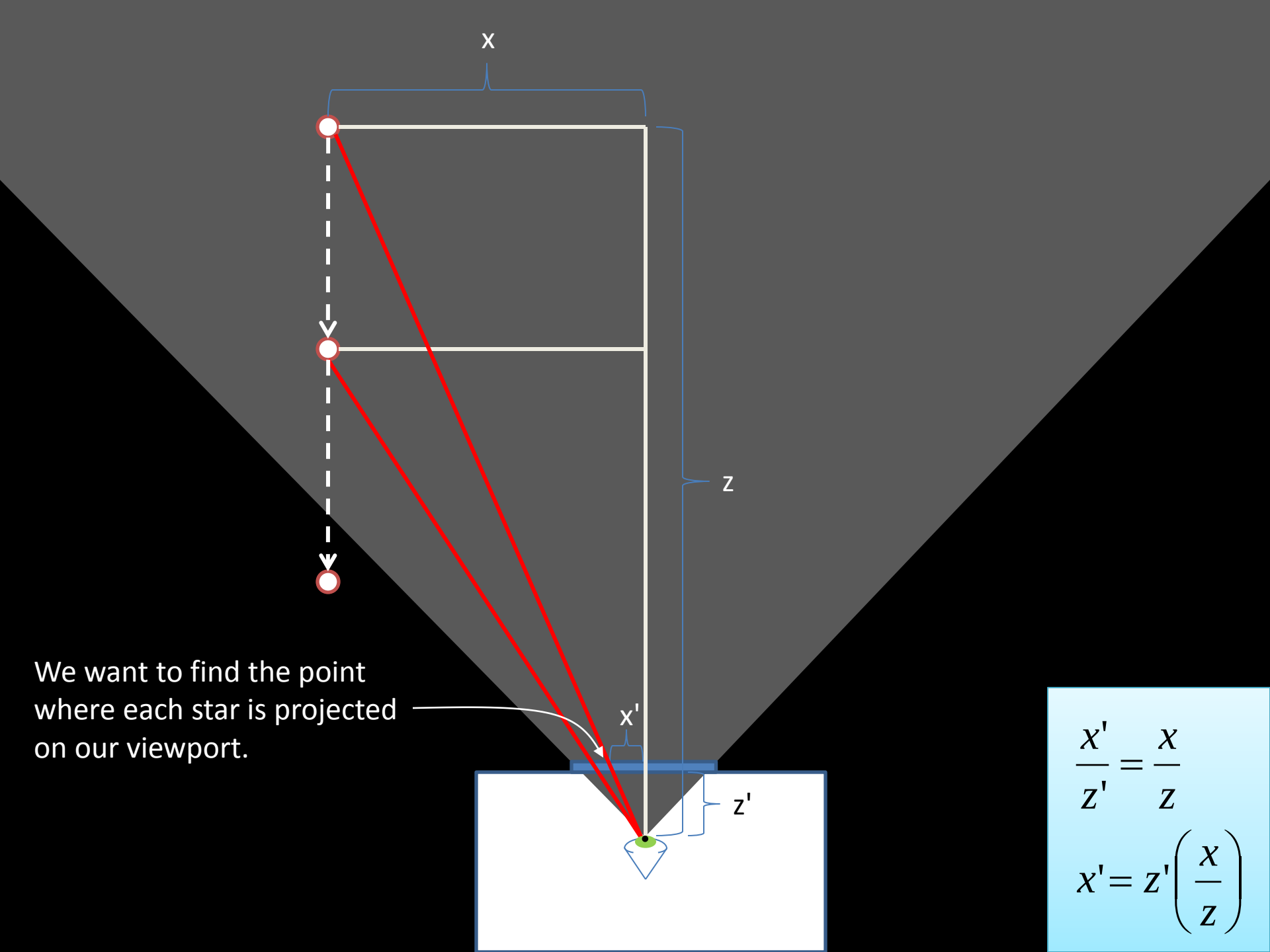


AllBoxes.py

A starfield using transformations



StarField.py



We want to find the point where each star is projected on our viewport.

$$\frac{x'}{z'} = \frac{x}{z}$$

$$x' = z'\left(\frac{x}{z}\right)$$

Star Class

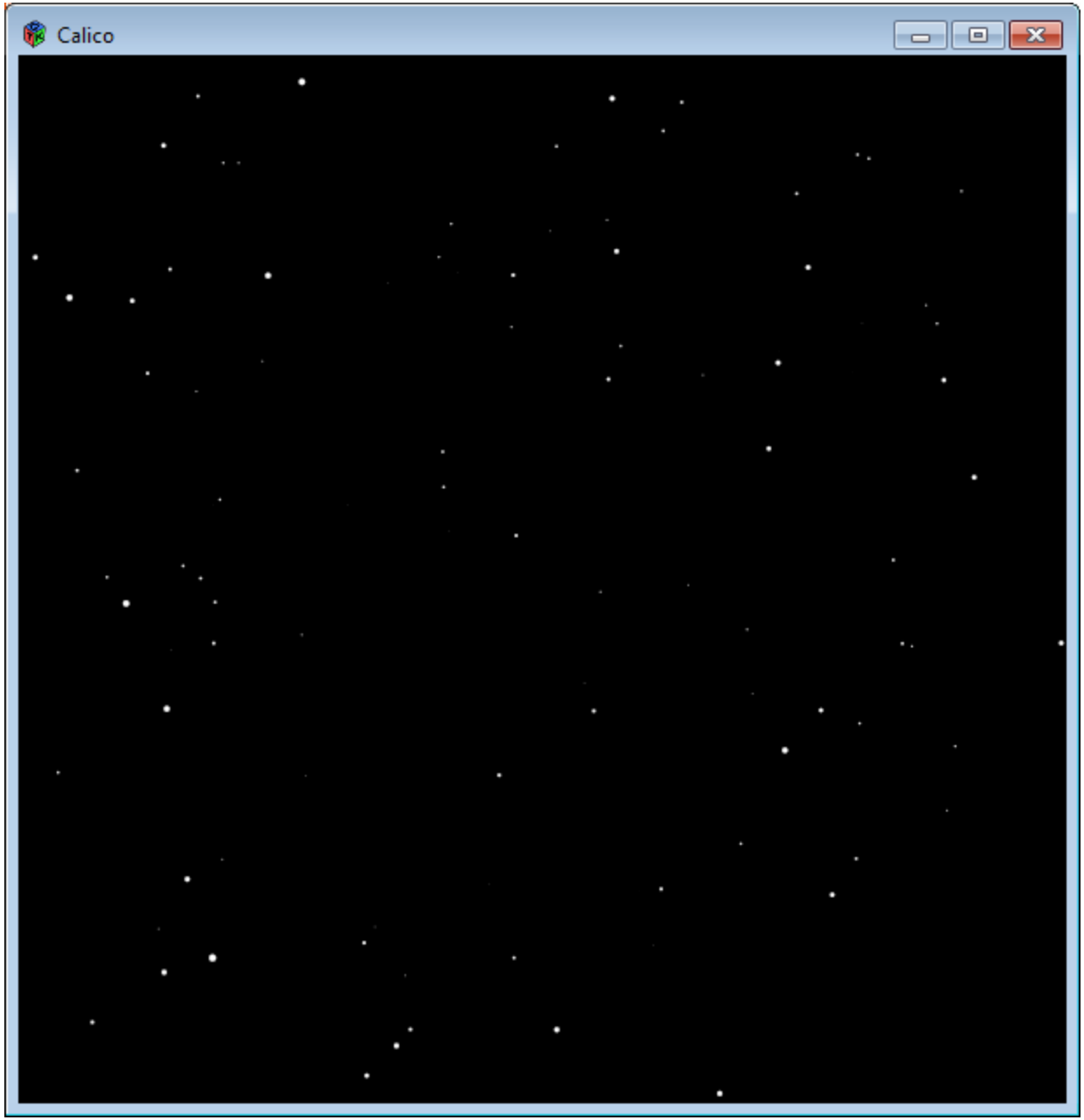
```
class Star:
    def __init__(self):
        self.x = random(-2000, 2000)
        self.y = random(-2000, 2000)
        self.z = random(0, 1000)

    def step(self):
        self.z = self.z - 5;
        if self.z <= 0.0:
            self.reset()

    def reset(self):
        # Reset star to a position far away
        self.x = random(-2000, 2000)
        self.y = random(-2000, 2000)
        self.z = 1000.0

    def display(self):
        # Project star only viewport
        offsetX = 100.0*(self.x/self.z)
        offsetY = 100.0*(self.y/self.z)
        scaleZ = 0.0001*(1000.0-self.z)

        # Draw this star
        pushMatrix()
        translate(offsetX, offsetY)
        scale(scaleZ)
        ellipse(0,0,50,50)
        popMatrix()
```

StarField.py

Star Class

```
# List of all stars
stars = []

stroke(255)
strokeWeight(5)
rectMode(CENTER)

# Init all stars
for i in range(200):
    stars.append( Star() )

# Draw all stars
def draw(o, e):
    background(0)

    pushMatrix()

    # Draw all stars wrt center of screen
    translate(0.5*width(), 0.5*height())

    for i in range( len(stars) ):
        stars[i].step()
        stars[i].display()

    popMatrix()

# Draw all
frameRate(50)
onLoop += draw
loop()
```

StarField.py



StarField.py