Animation: A Compelling Example of Objects and Inheritance

CS111 Computer Programming
Department of Computer Science
Wellesley College

Animation

We used `cs1graphics` earlier this semester to make static scenes. Now we revisit `cs1graphics` to make dynamic animations = scenes that change over time. It is remarkably easy to use Python’s classes/objects/inheritance to make a simple animation framework from scratch.

Example: Run the file `simpleAnimation.py` from Canopy. You can stop the animation by typing Ctrl-C Ctrl-C in the interaction pane.

The Composition of Animation

There are two main parts to our animation framework:

1. **Sprites** are the actresses in an animation. We create a cast of sprites to act in our play. Each sprite knows how to perform its own part. In particular it knows how to update its state for each time step of the animation.

2. **Animations** are the plays in which the sprites act. An animation has a canvas on which visual representations of the sprites are displayed. At each time step of the animation, each sprite is asked to update its state, which often changes how it appears on the canvas. As each sprite changes, we see a “movie” of the sprites’ performances.

 Concepts in this slide:
Sprites – the moving actors, 
animations – the static stage.

cs1graphics.py Details

The version of `cs1graphics.py` we’ve been using this semester has an undocumented feature we’ll use in our animations: `Drawable` objects have a `getAngle()` method that returns the angle by which the object has been rotated.

```python
s = Square(100, Point(250, 375))
s.getAngle() \rightarrow 0.0 \quad \# \text{default initial angle is 0.0}
s.rotate(90)
s.getAngle() \rightarrow 90.0
s.rotate(-12.5)
s.getAngle() \rightarrow 77.5
```

We’ll also use the `getReferencePoint()` method to get the (x,y) coordinates of the reference point of `Drawable` objects:

```python
position = s.getReferencePoint()
position.getX() \rightarrow 250.0
position.getY() \rightarrow 375.0
```
Just RotatingSquares

Run the file *rotatingSquaresAnimation.py* from Canopy.

```
from Animation import *
from RotatingSquare import *

rotatingSquares = Animation(800, 600, 'skyblue', 'RotatingSquares')

rotatingSquares.addSprite( 
    RotatingSquare(300, 400, 300, 'orange', 45, 5 )
)

rotatingSquares.addSprite( 
    RotatingSquare(500, 150, 250, 'green', 30, -1 )
)

rotatingSquares.start()
```

To Notice
An animation application will contain three steps:
1. Instantiate an object of class *Animation* to create the stage.
2. Instantiate one or more objects of class *Sprite* and add them to the animation object.
3. Start the animation by explicitly calling the method *start*.

Object-Based Diagram

Concepts in this slide:
The recipe for creating animation applications.

Animations consist of sprites that change state over time

Concepts in this slide:
One single state variable changing its value creates the animation effect.

To Notice
The sprites in slide 24-7 have an initial rotation angle (see also 24-6), and every moment, they change the value of this angle by a deltaAngle value. This creates the effect of rotation.
Animation Class: First Cut

from cs1graphics import *

class Animation:
    def __init__(self, width, height, color, title):
        # Create canvas for showing the sprites
        self.canvas = Canvas(width, height, color, title)
        self.sprites = []  # Keep track of sprites in animation

    def addSprite(self, sprite):
        self.sprites.append(sprite)
        sprite.addToCanvas(self.canvas)

    def start(self):
        while True:
            # Animation is infinite loop.
            # Stop it using Ctrl-C Ctrl-C.
            for sprite in self.sprites:
                sprite.step()  # Tell sprite to change by one step

Concepts in this slide:
An animation is a container of sprites that know how to change at each step.

Sprite Class:
Parent class of other sprite classes

class Sprite:
    def addToCanvas(self, canvas):
        canvas.add(self.square)

    def step(self):
        self.square.rotate(self.deltaAngle)

    def __repr__(self):
        return '{spriteName}<stateString>'.format(
           spriteName = self.__class__.__name__,
            stateString = self.stateString())

    def stateString(self):
        return ""

To Notice
The method format can take named parameters. In that case, the parameter names need to go inside the curly braces too in the format string.

Python special variables
When we create an instance of a class, Python will add to it some special variables that contain info about the class, for example:
self.__class__.__name__, which stores the class name as a string value.

RotatingSquare Class
w/o stateString method

from cs1graphics import *

class RotatingSquare(Sprite):
    '''Colored square that rotates.''

    def __init__(self, centerX, centerY, size, color, initialAngle, deltaAngle):
        sq = Square(size, Point(centerX, centerY))
        sq.setFillColor(color)
        sq.rotate(initialAngle)
        self.square = sq
        self.deltaAngle = deltaAngle
        self.color = color

    def addToCanvas(self, canvas):
        canvas.add(self.square)

    def step(self):
        self.square.rotate(self.deltaAngle)

Concepts in this slide:
A Sprite object will store in its instance variables values that are referenced in its methods.

Debugging Information
When debugging is turned on, we can see how sprites change over time. How does this work?

Concepts in this slide:
The __repr__ method in each class makes possible to print out debugging info.
For example:

```python
def stateString(self):
    return "color={c}; deltaAngle={da}; angle={a}".format(
        c = self.square.getFillColor(),
        da = self.deltaAngle,
        a = self.square.getAngle()
    )
```

```
Animation
rs1 = RotatingSquare(300, 400, 300, 'orange', 45, 5)
str(rs1) à 'RotatingSquare<color=orange; deltaAngle=5; angle=45.0>'
rs2 = RotatingSquare(500, 150, 250, 'green', 30, -1)
str(rs1) à 'RotatingSquare<color=green; deltaAngle=-1; angle=30.0>'
```

```python
For example:

```python
def stateString(self):
    return "color={c}; deltaAngle={da}; angle={a}".format(
        c = self.square.getFillColor(),
        da = self.deltaAngle,
        a = self.square.getAngle()
    )
```

```
Animation
rs1 = RotatingSquare(300, 400, 300, 'orange', 45, 5)
str(rs1) à 'RotatingSquare<color=orange; deltaAngle=5; angle=45.0>'
rs2 = RotatingSquare(500, 150, 250, 'green', 30, -1)
str(rs1) à 'RotatingSquare<color=green; deltaAngle=-1; angle=30.0>'
```

**Concepts in this slide:**
Every class that inherits from Sprite should override the method stateString.

---

**Inheritance at work**

```python
# In Sprite
def __repr__(self):
    return '{spriteName}<{stateString}>'.format(
        spriteName = self.__class__.__name__,
        stateString = self.stateString()
    )
```

```python
# In RotatingSquare
def stateString(self):
    return "color={c}; deltaAngle={da}; angle={a}".format(
        c = self.color,
        da = self.deltaAngle,
        a = self.square.getAngle()
    )
```

```python
rs1 = RotatingSquare(300, 400, 300, 'orange', 45, 5)
str(rs1) à 'RotatingSquare<color=orange; deltaAngle=5; angle=45.0>'
```

**Concepts in this slide:**
# str(rs1) works by calling first __repr__ in the parent class, which then calls stateString in child class.

---

**Animation Class: debugging**

---

**Add a new sprite**

In rotatingSquaresAnimation.py add a new RotatingSquare sprite and run the app again.

Check the debugging information.

Does it make sense?

```python
rotatingSquares = Animation(800, 600, 'skyblue', 'RotatingSquares')
rotatingSquares.addSprite(RotatingSquare(300, 400, 300, 'orange', 45, 5))
rotatingSquares.addSprite(RotatingSquare(500, 150, 250, 'green', 30, -1))
rotatingSquares.start()
```
Try it out: Rotating squares that move horizontally

Run the file movingRotatingSquaresAnimation.py from Canopy.

from Animation import *
from MovingRotatingSquare import *
movingRotatingSquares = Animation(800, 600, 'skyblue', 'MovingRotatingSquares')
movingRotatingSquares.addSprite( MovingRotatingSquare(300, 400, 300, 'orange', 45, 5, 3) )
movingRotatingSquares.addSprite( MovingRotatingSquare(500, 150, 250, 'green', 30, -1, -2) )
movingRotatingSquares.start()

Changing State in movingRotatingSquaresAnimation

<table>
<thead>
<tr>
<th>step Num</th>
<th>angle of orange rect (deltaAngle = 5)</th>
<th>x coord of orange rect (deltaX = 3)</th>
<th>angle of green rect (deltaAngle = -1)</th>
<th>x coord of Green rect (deltaX = -2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>45</td>
<td>300</td>
<td>30</td>
<td>500</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>303</td>
<td>29</td>
<td>498</td>
</tr>
<tr>
<td>2</td>
<td>55</td>
<td>306</td>
<td>28</td>
<td>496</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>309</td>
<td>27</td>
<td>494</td>
</tr>
<tr>
<td>4</td>
<td>65</td>
<td>312</td>
<td>26</td>
<td>492</td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td>315</td>
<td>25</td>
<td>490</td>
</tr>
</tbody>
</table>

To Notice
The variable deltaX affects the x coordinate of a sprite. By changing x it moves the sprite horizontally. The sign + or – for the deltaX controls direction: left to right or right to left, respectively.

Sprite Inheritance: MovingRotatingSquare

class MovingRotatingSquare(RotatingSquare):
    '''Colored square that rotates and moves horizontally with speed deltaX'''
    def __init__(self, centerX, centerY, size, color, initialAngle, deltaAngle, deltaX):
        # Explicitly invoke parent class constructor
        RotatingSquare.__init__(self, centerX, centerY, size, color, initialAngle, deltaAngle)
        # remember deltaX
        self.deltaX = deltaX
    def step(self):
        RotatingSquare.step(self)  # rotate the square
        self.square.move(self.deltaX, 0)  # move the square
    def stateString(self):
        return RotatingSquare.stateString(self) + ": dx={dx}; x={x}".format(dx=self.deltaX, x=self.square.getReferencePoint().getX())

Debugging info for MovingRotatingSquares

(MovingRotatingSquares[#sprites=2; step#=0; Sprites:
  MovingRotatingSquare[color=orange; deltaAngle=5; angle=45.0; deltaX=3; x=300.0]
  MovingRotatingSquare[color=green; deltaAngle=-1; angle=30.0; deltaX=-2; x=500.0])

(MovingRotatingSquares[#sprites=2; step#=1; Sprites:
  MovingRotatingSquare[color=orange; deltaAngle=5; angle=50.0; deltaX=3; x=303.0]
  MovingRotatingSquare[color=green; deltaAngle=-1; angle=29.0; deltaX=-2; x=498.0])

(MovingRotatingSquares[#sprites=2; step#=2; Sprites:
  MovingRotatingSquare[color=orange; deltaAngle=5; angle=55.0; deltaX=3; x=306.0]
  MovingRotatingSquare[color=green; deltaAngle=-1; angle=28.0; deltaX=-2; x=496.0])

(MovingRotatingSquares[#sprites=2; step#=3; Sprites:
  MovingRotatingSquare[color=orange; deltaAngle=5; angle=60.0; deltaX=3; x=309.0]
  MovingRotatingSquare[color=green; deltaAngle=-1; angle=27.0; deltaX=-2; x=494.0])

New state string info shown in magenta)
Try it out: Rotating moving squares that bounce

Run the file `bouncingRotatingSquaresAnimation.py` from Canopy.

```python
from Animation import *
from BouncingRotatingSquare import *

bouncingRotatingSquares = Animation(800, 600, 'skyblue', 'BouncingRotatingSquares')
bouncingRotatingSquares.addSprite(BouncingRotatingSquare(300, 400, 300, 'orange', 45, 5, 3))
bouncingRotatingSquares.addSprite(BouncingRotatingSquare(500, 150, 250, 'green', 30, -1, -2))
bouncingRotatingSquares.start()
```

Sprite Inheritance: BouncingRotatingSquare

```python
class BouncingRotatingSquare(MovingRotatingSquare):
    
    # __init__ method inherited
    def __init__(self, canvas):
        super().__init__(canvas)

    # Inherited behavior
    def addToCanvas(self, canvas):
        super().addToCanvas(canvas)
        self.maxX = canvas.getWidth()

    # Additional behavior for this class
    def step(self):
        super().step()
        # Rotate & move square
        pos = self.square.getReferencePoint()
        centerX = pos.getX()
        if centerX < 0 or centerX > self.maxX:
            self.deltaX = -self.deltaX  # Change direction

    def stateString(self):
        return super().stateString() + ';' + 'maxX={mx}'.format(mx=self.maxX)
```

Anatomy of the first bounce of the orange square

(New state string info shown in magenta)

```python
BouncingRotatingSquares[#sprites=2; step#=165; Sprites:
    BouncingRotatingSquare<color=orange; deltaAngle=5; angle=150.0;
    deltaX=3; x=795.0; maxX=800>
  ...]
BouncingRotatingSquares[#sprites=2; step#=166; Sprites:
    BouncingRotatingSquare<color=orange; deltaAngle=5; angle=155.0;
    deltaX=3; x=798.0; maxX=800>
  ...]
BouncingRotatingSquares[#sprites=2; step#=167; Sprites:
    BouncingRotatingSquare<color=orange; deltaAngle=5; angle=160.0;
    deltaX=-3; x=801.0; maxX=800>
  ...]
BouncingRotatingSquares[#sprites=2; step#=168; Sprites:
    BouncingRotatingSquare<color=orange; deltaAngle=5; angle=165.0;
    deltaX=-3; x=798.0; maxX=800>
  ...]
BouncingRotatingSquares[#sprites=2; step#=169; Sprites:
    BouncingRotatingSquare<color=orange; deltaAngle=5; angle=170.0;
    deltaX=-3; x=795.0; maxX=800>
  ...]
```

Pulsing Circles

Flesh out the `step` method for the Pulsar class in the `PulsarSkeleton.py` file (on the next slide) in two stages.

1. A growing circle should stop growing at `maxRadius` (and stay there), while a shrinking circle should stop shrinking at `minRadius` (and stay there).

2. A growing circle should start shrinking when it reaches `maxRadius`, while a shrinking circle should start growing when it reaches `minRadius`.

Try these out by running `pulsingCirclesAnimation.py`.

---

**Concepts in this slide:**
Yet a new animation, where the moving sprite bounces off the Canvas edges.

**Additional behavior for this class**

**Inherited behavior**
Try it out: bouncing images

Run the file
bouncingImagesAnimation.py
from Canopy.

from Animation import *
from BouncingImage import *

bouncingImages = Animation(800, 600, 'skyblue', 'BouncingImages')
bouncingImages.addSprite(BouncingImage(250, 100, 'glasses.gif', 7))
bouncingImages.addSprite(BouncingImage(700, 400, 'monkey.gif', -3))
bouncingImages.start()
**BouncingImage Sprite [2]**

```python
def step(self):
    # Override Sprite’s default step method
    imagePos = self.image.getReferencePoint()
    imageWidth = self.image.getWidth()
    nextX = imagePos.getX() + self.deltaX

    if nextX <= imageWidth/2.0:
        nextX = imageWidth/2.0  # Never let image go off left edge
        self.deltaX = -self.deltaX  # change deltaX from - to +
    
    elif nextX >= self.maxX:
        nextX = self.maxX  # Never let image go off right edge
        self.deltaX = -self.deltaX  # # change deltaX from + to -

    self.image.moveTo(nextX, imagePos.getY())  # Move to new pos

def stateString(self):
    return ImageSprite.stateString(self) + \
    ": deltaX={dx}; maxX={mx}; x={x}".format(
        dx = self.deltaX,
        mx = self.maxX,
        x = self.image.getReferencePoint().getX()
    )
```

---

**Concepts in this slide:** A summary hierarchy of all classes in this lecture with info where to find them.