Teaching Computational Thinking

An Integrative Approach for Middle and High School Learning

By: Maureen D. Neumann, Lisa Dion

Citation:

Teaching Computational Thinking: An Integrative Approach for Middle and High School Learning
By: Maureen D. Neumann, Lisa Dion
DOI: 10.7551/mitpress/11209.001.0001
ISBN (electronic): 9780262366144
Publisher: The MIT Press
Published: 2021

OA Funding Provided By:

The open access edition of this book was made possible by generous funding from Arcadia—a charitable fund of Lisbet Rausing and Peter Baldwin.

The MIT Press

Appendix: Computer Programs

This chapter contains full versions of programs that appear throughout the text. Programs are presented in Processing (see www.processing.org), Codesters, and Python.

A.1 Drawing Cretan Labyrinths

The following Processing program draws a seven-circuit Cretan labyrinth on a computer screen, as shown in figure 4.7.

```
// file: labyrinth.pde
// Draws a seven-circuit Cretan labyrinth in two colors.
// Author: Robert R. Snapp

float t=0;
float dt=0.5;
Pvector last1;
Pvector last2;
Pvector pv1;
Pvector pv2;

void drawCross() {
    line(10, -80, 10, 0);
    line(-30,-40, 50, -40);
    arc(50,0,40,40, radians(180), radians(360));
    arc(-30,0,40,40, radians(180), radians(360));
    arc(-30,-80,40,40, radians(-90), radians(90));
    arc(50,-80,40,40, radians(90), radians(270));
}

void drawUpperArc(float r) {
    arc(0,0,r,r,0, radians(180));
}

void drawSWQuarter(float r) {
    arc(-30, 0, r, r, radians(180), radians(270));
}
```

```
void drawSEQuarter(float r){
    arc(50, 0, r, r, radians(270), radians(360));
}

void vbranch(float t, Pvector pv) {
    float rad=radians(t);
    if (t <= 0) {
        pv.x=50;
        pv.y=0;
    } else if (t <= 180) {
        pv.x=50*cos(rad);
        pv.y=50*sin(rad);
    } else if (t <= 360) {
        pv.x=20*cos(rad)-30;
        pv.y=20*sin(rad);
    } else if (t <= 540) {
        pv.x=-10*cos(rad);
        pv.y=10*sin(rad);
    } else if (t <= 630) {
        pv.x=10.0;
        pv.y=-40.0*(t-540)/90.;
    } else if (t <= 720) {
        pv.x=10.0;
        pv.y=-40*(t-630)/90.-40;
    } else if (t <= 810) {
        pv.x=-30+40*cos(rad);
        pv.y=-80-40*sin(rad);
    } else if (t <= 900) {
        pv.x=-30+120*cos(rad);
        pv.y=-120*sin(rad);
    } else if (t <= 1080) {
        pv.x=150*cos(rad);
        pv.y=-150*sin(rad);
    } else if (t <= 1170) {
        pv.x=50+100*cos(rad);
        pv.y=-100*sin(rad);
    } else if (t <= 1350) {
        pv.x=50+20*cos(rad);
        pv.y=-80-20*sin(rad);
    } else if (t <= 1440) {
        pv.x=50+60*cos(rad);
        pv.y=60*sin(rad);
    } else if (t <= 1620) {
        pv.x=110*cos(rad);
        pv.y=110*sin(rad);
    } else if (t <= 1710) {
        pv.x=-30+80*cos(rad);
        pv.y=80*sin(rad);
    } else {
```

```
        pv.x=-30;
        pv.y=-80;
    }
}

void hbranch(float t, Pvector pv) {
    float rad=radians(t);
    if (t <= 0) {
        pv.x=-30;
        pv.y=0;
    } else if (t <= 180) {
        pv.x=-30*cos(rad);
        pv.y=30*sin(rad);
    } else if (t <= 360) {
        pv.x=20*cos(rad)+50;
        pv.y=20*sin(rad);
    } else if (t <= 540) {
        pv.x=70*cos(rad);
        pv.y=70*sin(rad);
    } else if (t <= 630) {
        pv.x=40*cos(rad)-30;
        pv.y=40*sin(rad);
    } else if (t <= 720) {
        pv.x=40*(t-630)/90.-30;
        pv.y=-40;
    } else if (t <= 810) {
        pv.x=40*(t-720)/90.+10;
        pv.y=-40;
    } else if (t <= 900) {
        pv.x=50-40*cos(rad);
        pv.y =-40*sin(rad);
    } else if (t <= 1080) {
        pv.x=-90*cos(rad);
        pv.y=-90*sin(rad);
    } else if (t <= 1170) {
        pv.x=-30-60*cos(rad);
        pv.y=-60*sin(rad);
    } else if (t <= 1350) {
        pv.x=-30-20*cos(rad);
        pv.y=-80+20*sin(rad);
    } else if (t <= 1440) {
        pv.x=-30-100*cos(rad);
        pv.y=100*sin(rad);
    } else if (t <= 1620) {
        pv.x=-130*cos(rad);
        pv.y=130*sin(rad);
    } else if (t <= 1710) {
        pv.x=50-80*cos(rad);
        pv.y=80*sin(rad);
```

```
    } else {
        pv.x=50;
        pv.y=-80;
    }
}

void setup(){
    size(900,900);
    frameRate(480);
    smooth();
    background(240);

    last1=new PVector(0,0);
    last2=new PVector(0,0);
    pv1=new PVector(0,0);
    pv2=new PVector(0,0);
    vbranch(1710, last1);
    hbranch(1710, last2);
    vbranch(1710, pv1);
    hbranch(1710, pv2);
}

void draw() {
    translate(width/2, height/2);
    scale(2.5,-2.5);

    vbranch(1710-t, pv1);
    hbranch(1710-t, pv2);

    strokeWeight(3);
    stroke(200,0,0);
    line(pv1.x, pv1.y, last1.x, last1.y);
    last1=pv1;

    stroke(150, 150, 255);
    line(pv2.x, pv2.y, last2.x, last2.y);
    last2=pv2;

    t += dt;
}
```

A.2 Bumper Rocks

The following program is coded in Codesters to simulate elastic collision, as seen in figure 5.8.

```
# Author: Lisa Dion
# set the background image
stage.set_background("jupiter")
```

```
# create the rock with initial size and speed
rock=codesters.Sprite("rock")
rock.set_size(0.5)
rock.set_x_speed(2)
rock.set_y_speed(-3)

# create the asteroid with initial size, position, and speed
asteroid=codesters.Sprite("asteroid")
asteroid.set_size(0.5)
asteroid.go_to(-200, 0)
asteroid.set_x_speed(-1)
asteroid.set_y_speed(4)

# create a function for the "s" key to make the rock smaller
def s_key():
    # make the rock smaller
    rock.set_size(0.8)
# this line makes the code listen for the "s" key
stage.event_key("s", s_key)

# create a function for the "w" key to make the rock larger
def w_key():
    # make the rock bigger
    rock.set_size(1 / 0.8)
# this line makes the code listen for the "w" key
stage.event_key("w", w_key)

# create a function for collision of the objects
def collision(sprite, hit_sprite):
    # set variables
    size_sum=rock.get_size()+asteroid.get_size()
    x_vel_1=sprite.get_x_speed()
    x_vel_2=hit_sprite.get_x_speed()
    y_vel_1=sprite.get_y_speed()
    y_vel_2=hit_sprite.get_y_speed()
    rock_x=rock.get_x()
    rock_y=rock.get_y()
    asteroid_x=asteroid.get_x()
    asteroid_y=asteroid.get_y()
   # calculate the new velocities for 2-dimensional elastic collisions
    # the following 4 command lines are each on their own line
    sprite.set_x_speed(x_vel_1-((2*asteroid.get_size())/size_sum) *
((x_vel_1-x_vel_2)*(rock_x-asteroid_x)+(y_vel_1-y_vel_2)*( rock_y-
asteroid_y))  /  ((rock_x-asteroid_x)*(rock_x-asteroid_x)+(rock_y-
asteroid_y)*(rock_y-asteroid_y)) * (rock_x-asteroid_x))
    sprite.set_y_speed(y_vel_1-((2*asteroid.get_size())/size_sum) *
((x_vel_1-x_vel_2)*(rock_x-asteroid_x)+(y_vel_1- y_vel_2)*(rock_y-
asteroid_y))  /  ((rock_x-asteroid_x)*(rock_x-asteroid_x)+(rock_y-
asteroid_y)*(rock_y-asteroid_y)) * (rock_y- asteroid_y))
```

```
    hit_sprite.set_x_speed(x_vel_2-((2*rock.get_size())/size_sum) *
((x_vel_2—x_vel_1)*(asteroid_x—rock_x)+(y_vel_2— y_vel_1)*(asteroid
_y—rock_y))   /   ((rock_x—asteroid_x)*(rock_x—asteroid_x)+(rock_y—
asteroid_y)*(rock_y—asteroid_y)) * (asteroid_x—rock_x))
    hit_sprite.set_y_speed(y_vel_2-((2*rock.get_size())/size_sum)*((x
_vel_2—x_vel_1)*(asteroid_x—rock_x)+(y_vel_2—y_vel_1)*(asteroid_y—
rock_y))/((rock_x—asteroid_x)*(rock_x—asteroid_x)+(rock_y—asteroid
_y)*(rock_y—asteroid_y)) * (asteroid_y—rock_y))

# this line makes the program detect collision and
# call the function above
rock.event_collision(collision)

# create a function to move the objects and bounce off walls
def move(sprite):
    sprite.move_forward(1)
    if sprite.get_x() >= 250 or sprite.get_x() <= -250:
        sprite.set_x_speed(-sprite.get_x_speed())
    if sprite.get_y() >= 250 or sprite.get_y() <= -250:
        sprite.set_y_speed(-sprite.get_y_speed())

# have the program run forever
while True:
    move(rock)
    move(asteroid)
```

A.3 Basketball

The following program is coded in Codesters to simulate the physics involved in shooting a basketball, as seen in figure 5.9.

```
# Author: Lisa Dion
# set gravity's strength
stage.set_gravity(10)
stage.set_background("halfcourt")
net=codesters.Sprite("basketballnet", 200, 100)
# we do not want the net to fall
net.set_physics_off()
net.set_gravity_off()
player=codesters.Sprite("player3", -180, -130)
# we do not want the player to fall
player.set_physics_off()
player.set_gravity_off()
ball=codesters.Sprite("basketball", -150, -130)
# we do not want the ball to fall yet
ball.set_physics_off()
ball.set_gravity_off()
# create a rectangle along the floor
```

```
floor=codesters.Rectangle(0, -250, 500, 10, "red")
# create a function called shoot
def shoot(x_speed, y_speed):
    # set the ball's speed to make it move
    ball.set_x_speed(x_speed)
    ball.set_y_speed(y_speed)

# create a function for collisions
def collision(sprite, hit_sprite):
    # check for a score
    # the next two lines should be on one line of code
     if sprite.get_x() >= 165 and sprite.get_x() <= 215 and sprite.
get_y() >= 75 and sprite.get_y() <= 100 and sprite.get_y_speed() < 0:
        # stop the ball
        sprite.set_x_speed(0)
        sprite.set_y_speed(0)
        sprite.set_physics_off()
        sprite.set_gravity_off()
        # create a win message
        message=codesters.Text("You scored!", 0, 0, "white")
    # check if the ball hit the red floor
    elif hit_sprite.get_color() == "red":
        # stop the ball
        sprite.set_x_speed(0)
        sprite.set_y_speed(0)
        sprite.set_physics_off()
        sprite.set_gravity_off()
        # create a lose message
        message=codesters.Text("Try again!", 0, 0, "white")

ball.event_collision(collision)

# prompt the player for inputs
x_vel=int(player.ask("Enter an x-velocity:"))
y_vel=int(player.ask("Enter a y-velocity:"))
# now we want the ball to fall
ball.set_physics_on()
ball.set_gravity_on()
# this calls the function above
shoot(x_vel, y_vel)
```

A.4 Ping-pong

The following program is written in Codesters to demonstrate reflection, as discussed in section 5.3.

```
# Author: Lisa Dion
# import the random number generator library module
import random
```

```
# set a list of speeds that we will choose from
speeds=[-4, -3, 3, 4]
# create the ball and set initial values
ball=codesters.Circle(0, 0, 50, "green")
ball.set_size(0.5)
# set the ball velocity in x and y components
ball.set_x_speed(speeds[random.randint(0, len(speeds)-1)])
ball.set_y_speed(speeds[random.randint(0, len(speeds)-1)])
# create rectangles for the paddles
# sprite=codesters.Rectangle(x, y, width, height, "color")
player1=codesters.Rectangle(-230, 0, 10, 50, "yellow")
player2=codesters.Rectangle(230, 0, 10, 50, "gray")
# create rectangles against the left and right walls
left_wall=codesters.Rectangle(-250, 0, 10, 500, "red")
right_wall=codesters.Rectangle(250, 0, 10, 500, "red")

# create a function for when "w" is pressed
def w_key():
    # make sure there is room to move up
    if player1.get_y() < 220:
        player1.move_up(20)

# create a function for when "s" is pressed
def s_key():
    # make sure there is room to move down
    if player1.get_y() > -220:
        player1.move_down(20)

# create a function for when up is pressed
def up_key():
    # make sure there is room to move up
    if player2.get_y() < 220:
        player2.move_up(20)

# create a function for when down is pressed
def down_key():
    # make sure there is room to move down
    if player2.get_y() > -220:
        player2.move_down(20)

# make the program listen for the keys
stage.event_key("s", s_key)
stage.event_key("w", w_key)
stage.event_key("up", up_key)
stage.event_key("down", down_key)

# create a function to detect collision
def collision(sprite, hit_sprite):
    # make ball bounce off paddles by changing its x-direction
```

```
    ball.set_x_speed(-ball.get_x_speed())
    # check if the ball hit the left or right wall
    if hit_sprite.get_color() == "red":
        # stop the ball
        ball.set_x_speed(0)
        ball.set_y_speed(0)
        if hit_sprite.get_x() < 0:
            # ball hit left wall.
            msg=codesters.Text("Player 2 wins!", 0, 0, "red")
        else:
            # ball hit right wall.
            msg=codesters.Text("Player 1 wins!", 0, 0, "red")
# make the program listen for collisions
ball.event_collision(collision)
```

A.5 Reading Temperatures from a File

The following program is written in Python to read from a file and graph average January temperatures for Burlington, Vermont, as seen in figure 6.19.

```
# Author: Lisa Dion
# Import the csv library so we can read in from a csv file
import csv
# Import matplotlib so we can graph the data
import matplotlib.pyplot as plt
# Import numpy so we can fit a line to the data
from numpy.polynomial.polynomial import polyfit

# Use January as the month to graph
month_string='01'

# Open the csv file
with open('BurlingtonVTData.csv') as csv_file:
    # Use the reader from the csv library
    reader=csv.reader(csv_file, delimiter=',')
    # Declare an empty list for all the months' temps for
    # the best fit line
    all_avg_month_temps=[]
    # Declare empty lists to hold what will be the x and y
    # values to graph
    years=[]
    graph_temps=[]
    # Skip the header line
    next(reader)
    # For each line of data in the file
    for row in reader:
        # Add the average temperature to the list
        all_avg_month_temps.append(float(row[3]))
        # If the month matches the month to graph
```

```
        if row[2][-2:] == month_string:
        # Add the year and the average temperature to the
        # lists to graph
            years.append(int(row[2][0:4]))
            graph_temps.append(float(row[3]))

# Use matplotlib to graph the data
ax=plt.subplot(111)
ax.plot(years, graph_temps)
# Only print every tenth year on x-axis
ax.xaxis.set_major_locator(plt.MaxNLocator(10))
# Make sure y-axis starts at 0
ax.set_ylim(bottom=0)
ax.hlines([5, 10, 15, 20, 25, 30], 1940, 2019)
# Label graph and axes
plt.gcf().canvas.set_window_title('BVT January Temperatures')
plt.title('Average January Temperatures for Burlington, VT')
plt.xlabel('Year')
plt.ylabel('Average January Temperature (F)')

# Use numpy to calculate the best fit line to graph
b_graph, m_graph=polyfit(years, graph_temps, 1)
# Calculate coordinates from the line's y and b values
line_y_values=[]
for value in years:
    line_y_values.append(value * m_graph+b_graph)
# Plot the line on the graph
plt.plot(years, line_y_values, '-')

# Use numpy to calculate the best fit line on all the data
# The next command should be on one line
b_all, m_all=polyfit(range(len(all_avg_month_temps)),
all_avg_month_temps, 1)
# Calculate the temperatures of the first and last month
print('First month temp (F):', b_all)
# The next command should be on one line
print('Last month temp(F):', (len(all_avg_month_temps)-1)
* m_all+b_all)
print ('Slope of best fit line:', format(m_all,'.5f'))

# Display the graph in a new window
plt.show()
```

The MIT Press would like to thank the anonymous peer reviewers who provided comments on drafts of this book. The generous work of academic experts is essential for establishing the authority and quality of our publications. We acknowledge with gratitude the contributions of these otherwise uncredited readers.

This book was set in Stone Serif and Stone Sans by Westchester Publishing Services.

Library of Congress Cataloging-in-Publication Data

Names: Neumann, Maureen D., author. | Dion, Lisa (Computer scientist), author.
Title: Teaching computational thinking : an integrative approach for middle and high school learning / Maureen D. Neumann and Lisa Dion with Robert Snapp.
Description: Cambridge, Massachusetts : The MIT Press, 2021. | Includes bibliographical references.
Identifiers: LCCN 2021000766 | ISBN 9780262045056 (paperback)
Subjects: LCSH: Computer science—Study and teaching (Secondary) | Critical thinking—Study and teaching (Secondary)
Classification: LCC QA76.27 .N474 2021 | DDC 004.071—dc23
LC record available at https://lccn.loc.gov/2021000766