CS202 / 302 Fall 2024

Final Project

Team 4

Abigail Blevins

David Berry

Shantel Walker

Project: Adding Features to Pong, a game written in Python using the Turtle Module.

12/8/2024

Summary

* Downloaded, studied, and added comments to the original game.
* Created 2 Variations, each with different added features
  + 1 - **Added 2nd Ball** –
    - Play with 2 balls going at once
  + 2 - **Added Paddles to top and bottom of screen** 
    - Each Player controls one side Paddle and one top or bottom paddle
* Created Several Enhancements to Original and New Features:
  + Screen Color
  + Ball Color
  + Paddle Color
  + Start by Key Tap
  + Improved key placement
  + Sound Effects

If we had more time we would enable user to stop play and get out of infinite loop in original game, allow user to choose between enhancements, Regular mode, 2 balls and 2 paddles, or 2 balls and 4 paddles. We would allow the user to choose length of game: Play to 11, Play to 21, and look into delay caused by system sound.

Original Source: Need to add web address of original

Flowchart:

A diagram of a process

Description automatically generated

A diagram of paddles and paddles

Description automatically generated

Continued on next page.

A diagram of a ball

Description automatically generated

**Details of Code** of final revision (4a), including changes we made. Changes made are discussed using blue text.

Modules Imported

import turtle

import time

This section imports the turtle (graphics) and time modules.

Turtle is a graphics module that is an implementation geometric drawing tools. Turtle graphics provides a representation of a physical “turtle” that draws on the screen using python turtle functions.

The Time module allows work with time in Python. It allows functionality like getting the current time, pausing the Program from executing, etc.

Define Screen Properties

# Create screen

sc = turtle.Screen()

sc.title("Pong game")

sc.bgcolor("blue")

sc.setup(width=1000, height=600)

Define Paddle Geometry

These are turtle functions that define the size, color and title of the window that is produced by the turtle module.

# Left paddle

left\_pad = turtle.Turtle()

left\_pad.speed(0)

left\_pad.shape("square")

left\_pad.color("red") # rev 2: changed from black to red

left\_pad.shapesize(stretch\_wid=6, stretch\_len=2)

left\_pad.penup()

left\_pad.goto(-400, 0)

# Right paddle

right\_pad = turtle.Turtle()

right\_pad.speed(0)

right\_pad.shape("square")

right\_pad.color("black")

right\_pad.shapesize(stretch\_wid=6, stretch\_len=2)

right\_pad.penup()

right\_pad.goto(400, 0)

These are turtle functions that define the size, color, location and shape of the original left and right paddles. Note that each paddle is the turtle as it moves and is initially stationary. We changed the color of the left paddle to better indicate which paddles were “on the same team” for scoring. See next section that discusses top and bottom paddles.

Define Paddle Geometry for top and bottom paddles (new feature)

# start mod to add top and bottom paddles -rev2

# top paddle: copy left\_pad and change left to top

top\_pad = turtle.Turtle() # rev 2

top\_pad.speed(0) # rev 2

top\_pad.shape("square") # rev 2

top\_pad.color("red") # rev 2: change black to red

top\_pad.shapesize(stretch\_wid=2, stretch\_len=6) # rev 2: swap wid and len

top\_pad.penup() # rev 2

top\_pad.goto(0, 200) # rev 2: from -400,0 to 0,200

# rev 2

# Bottom paddle: copy right pad and change right to bottom

bottom\_pad = turtle.Turtle() # rev 2

bottom\_pad.speed(0) # rev 2

bottom\_pad.shape("square") # rev 2

bottom\_pad.color("black") # rev 2: leave black

bottom\_pad.shapesize(stretch\_wid=2, stretch\_len=6) # rev 2: swap wid and len

bottom\_pad.penup() # rev 2

bottom\_pad.goto(0, -200) # rev 2: from 400,0 to 0,-200

# end of rev 2 paddle mod

We modified the code to add paddles on the top and bottom of the screen. The turtle functions that defined the original left and right paddles were copied and modified. The color, locations and length and width were modified as shown in the comments.

Define Ball Geometry

# Ball of circle shape

hit\_ball = turtle.Turtle()

hit\_ball.speed(4) # Adjusted speed

hit\_ball.shape("circle")

hit\_ball.color("white")

hit\_ball.penup()

hit\_ball.goto(0, 0)

hit\_ball.dx = 0 # Set the ball speed to zero at the start of the game (Rev1b)

hit\_ball.dy = 0

Define Ball Geometry for second ball (new feature)

# Ball2 of circle shape

hit\_ball2 = turtle.Turtle()

hit\_ball2.speed(4) # Adjusted speed

hit\_ball2.shape("circle")

hit\_ball2.color("black")

hit\_ball2.penup()

hit\_ball2.goto(0, 0)

hit\_ball2.dx = 0 # Set the ball speed to zero at the start of the game (Rev3a)

hit\_ball2.dy = 0

Setting it up similarly to the first ball, but with some distinct properties. **hit\_ball2 = turtle.Turtle()**: This creates a new Turtle object named hit\_ball2….. **hit\_ball2.speed(4)**: This sets the speed of the hit\_ball2 to a value of 4…. **hit\_ball2.shape("circle")**: This sets the shape of the ball to a circle. **hit\_ball2.penup()**: This ensures that the hit\_ball2 does not draw anything as it moves around the screen.…. **hit\_ball2.goto(0, 0)**: This places the hit\_ball2 at the origin of the screen (coordinates 0, 0), which is the center of the window.…**hit\_ball2.dx = 0**: This sets the horizontal speed (movement in the X direction) of the second ball to 0 at the start of the game. This means the ball won't move horizontally initially……. **hit\_ball2.dy = 0**: This sets the vertical speed (movement in the Y direction) of the second ball to 0 at the start of the game, meaning the ball won't move vertically initially either.

Initialize and Display Scores

# Initialize the score

left\_player = 0

right\_player = 0

These two lines initialize the scores for the left player (left\_player) and the right player (right\_player) to 0. This is the starting score for both players at the beginning of the game. As the game progresses, these values will be updated based on the players' actions (e.g., when one player scores a point).

# Displays the score

sketch = turtle.Turtle()

sketch.speed(0)

sketch.color("black")

sketch.penup()

sketch.hideturtle()

sketch.goto(0, 260)

sketch.write("Left\_player : 0 Right\_player: 0",

align="center", font=("Courier", 24, "normal"))

This block of code is responsible for displaying the score on the screen using another Turtle object (sketch).

Define Functions to move left and right paddles

# Functions to move paddles

def paddleaup():

y = left\_pad.ycor()

if y < 250: # Limit paddle movement

y += 20

left\_pad.sety(y)

def paddleadown():

y = left\_pad.ycor()

if y > -240: # Limit paddle movement

y -= 20

left\_pad.sety(y)

def paddlebup():

y = right\_pad.ycor()

if y < 250: # Limit paddle movement

y += 20

right\_pad.sety(y)

def paddlebdown():

y = right\_pad.ycor()

if y > -240: # Limit paddle movement

y -= 20

right\_pad.sety(y)

When these functions are called, they each

* get the current y (vertical) coordinate of the paddle
* check to make sure the paddle has not reached the end of the screen. Note that the limits are not the same as the screen size to account for the width of the paddle.
* adds 20 pixels to the paddle y coordinate.

This code defines functions to control the movement of two paddles using the turtle module. These functions allow the left and right paddles to move up and down within a restricted range.

Each of these functions moves a paddle (either left or right) within a defined range on the screen: **paddleaup()** moves the left paddle up…..**paddleadown()** moves the left paddle down…**paddlebup()** moves the right paddle up….**paddlebdown()** moves the right paddle down.

These functions use the ycor() method to get the current vertical position of the paddle and the sety() method to update its position. The if conditions prevent the paddles from moving out of the screen boundaries (defined by the y-coordinate limits of 250 for the top and -240 for the bottom).

Define functions to move top and bottom paddles (new features)

# rev 2 copy paddle movements and change to move top and bottom paddles

#

# changes: y to x; left to top; right to bottom; 250 to 450; 240 to 440

# rev 2

def paddlecright(): # rev 2

x = top\_pad.xcor() # rev 2

if x < 450: # Limit paddle movement # rev 2

x += 20 # rev 2

top\_pad.setx(x) # rev 2

# rev 2

# rev 2

def paddlecleft(): # rev 2

x = top\_pad.xcor() # rev 2

if x > -440: # Limit paddle movement # rev 2

x -= 20 # rev 2

top\_pad.setx(x) # rev 2

# rev 2

# rev 2

def paddledright(): # rev 2

x = bottom\_pad.xcor() # rev 2

if x < 450: # Limit paddle movement # rev 2

x += 20 # rev 2

bottom\_pad.setx(x) # rev 2

# rev 2

# rev 2

def paddledleft(): # rev 2

x = bottom\_pad.xcor() # rev 2

if x > -440: # Limit paddle movement # rev 2

x -= 20 # rev 2

bottom\_pad.setx(x) # rev 2

# rev 2

# end of rev 2: paddle movements for top and bottom paddles

These functions copy the original functions with the coordinate directions and values modified. The functions:

* get the current x (horizontal) coordinate of the paddle
* check to make sure the paddle has not reached the end of the screen (horizontally)
* adds 20 pixels to the paddle x coordinate.

Define functions to delay start (new feature)

# Functions that add a start delay (Rev1b/Rev3a)

def startplay():

hit\_ball.dx = 5

hit\_ball.dy = -5

hit\_ball2.dx = -5

hit\_ball2.dy = -5

def stopplay():

hit\_ball.dx = 0

hit\_ball.dy = 0

hit\_ball2.dx = 0

hit\_ball2.dy = 0

#def Quit(): # quit not working

# Quit()

Startplay() is used to make it so that the game doesn’t start immediately when you start the code. Quit() was removed.

Define actions that are taken when keys are pressed for left and right paddles

# Keyboard bindings

sc.listen()

sc.onkeypress(paddleaup, "w") # Changed to 'w'

sc.onkeypress(paddleadown, "s") # Changed to 's'

sc.onkeypress(paddlebup, "Up")

sc.onkeypress(paddlebdown, "Down")

This allows the player to control the paddles, you would likely bind these functions to keyboard keys. This way, when the player presses the w or s keys, the left paddle moves up or down, and when the player presses the arrow keys, the right paddle moves up or down accordingly.

Define actions that are taken when keys are pressed to start play

# Rev1b

sc.onkeypress(startplay, "e")

#sc.onkeypress(Quit, "q") rev 3d

The game starts when you press e; Quit() was removed.

This code is responsible for starting the game when a specific key is pressed (in this case, the **"e"** key). It also hints at a potential feature for quitting the game with the **"q"** key (although the line is commented out).

Define actions that are taken when keys are pressed for top and bottom paddles (new feature)

# rev 2: copy and modify key bindings to add top and bottom paddle movements

# changes: a to c; b to d; up to left; down to right

# rev 2: "1" key moves top paddle left

sc.onkeypress(paddlecleft, "a") # Changed to 'w'(original comment) (Rev3a) (Switched '1' to 'a')

# rev 2: "2" key moves top paddle right

sc.onkeypress(paddlecright, "d") # Changed to 's' (original comment)(Rev3a) (Switched '2' to 'd')

# rev 2: "8" key moves bottom paddle left

# rev 3: Changed movement from '8' to 'Left'

sc.onkeypress(paddledleft, "Left")

# rev 2: "9" key moves bottom paddle right

# rev 3: Changed movement from '9' to 'Right'

sc.onkeypress(paddledright, "Right")

#

# rev 2: end of top and bottom key bindings

The new key bindings allow the player to control **top** and **bottom** paddles in addition to the left and right paddles.

**Top paddle**:

* + - **"a"** moves it left.
    - **"d"** moves it right.

**Bottom paddle**:

* + - **Left Arrow** moves it left.
    - **Right Arrow** moves it right.

These key bindings give the player full control over all four paddles (top, bottom, left, and right).

Main Game Loop

while True:

This starts an infinite loop of game play. The rest of the program statements are inside this loop. A desired future enhancement would be to include a conditional statement such as an if statement to stop the game with a key press.

Screen Refresh

sc.update()

time.sleep(0.01) # Add delay to make game smoother

This causes the screen to be refreshed at the start of each loop and adds a delay to make the game smoother.

Ball Movement

hit\_ball.setx(hit\_ball.xcor() + hit\_ball.dx)

hit\_ball.sety(hit\_ball.ycor() + hit\_ball.dy)

# Ball 2

hit\_ball2.setx(hit\_ball2.xcor() + hit\_ball2.dx)

hit\_ball2.sety(hit\_ball2.ycor() + hit\_ball2.dy)

This causes the balls to move by incrementing the x and y coordinates by a predefined amount.

Cause Balls to Bounce when they encounter sides without paddles

# Checking borders

# rev 2: with top, bottom paddles active will comment out 2 if statements for y coordinates

# rev 2: these made balls bounce off of top and bottom sides

# if hit\_ball.ycor() > 280: # rev 2: commented out bounce so can score on top and bottom

# hit\_ball.sety(280) # rev 2: commented out bounce so can score on top and bottom

# hit\_ball.dy \*= -1 # rev 2: commented out bounce so can score on top and bottom

# if hit\_ball.ycor() < -280: # rev 2: commented out bounce so can score on top and bottom

# hit\_ball.sety(-280) # rev 2: commented out bounce so can score on top and bottom

# hit\_ball.dy \*= -1 # rev 2: commented out bounce so can score on top and bottom

# rev 2: add "or" statements for y coordinate (top and bottom) scores for passed balls

# rev 2: (was) if hit\_ball.xcor() > 500:

# Ball 2 boarders

# rev 2: for ball 2

# rev 2: with top, bottom paddles active will comment out 2 if statements for y coordinates

# rev 2: these made balls bounce off of top and bottom sides

# if hit\_ball2.ycor() > 280: # rev 2: commented out bounce so can score on top and bottom

# hit\_ball2.sety(280) # rev 2: commented out bounce so can score on top and bottom

# hit\_ball2.dy \*= -1 # rev 2: commented out bounce so can score on top and bottom

# if hit\_ball2.ycor() < -280: # rev 2: commented out bounce so can score on top and bottom

# hit\_ball2.sety(-280) # rev 2: commented out bounce so can score on top and bottom

# hit\_ball2.dy \*= -1 # rev 2: commented out bounce so can score on top and bottom

Note that this was commented out when paddles were added to the top and bottom. Before adding the paddles to top and bottom and allowing a score when the ball hit the top and bottom, the ball had to bounce off the top and bottom sides. This is done by an “if “statement. If the ball’s y coordinate has passed +/- 280, set y coordinate is set to 280 and the ball position y increment is multiplied by -1, thus reversing the ball’s direction.

Ball 2 boarders were added by us when we added a second ball (before adding a second set of paddles). These were commented out only when the second set of paddles were added.

This code is related to handling the **ball's behavior when it encounters the top and bottom edges** of the game area (borders) in the game. Specifically, this section of code is designed to make the ball **bounce off** the top and bottom boundaries unless a certain condition (like scoring) is met.

This code is meant to **detect collisions** between the balls (hit\_ball and hit\_ball2) and the top or bottom edges of the screen. When a ball reaches the top or bottom, the code would normally **reverse the vertical direction (dy) of the ball**, making it bounce back. However, in the updated version of the code, the bounce is **commented out** because the game is designed to allow balls to **pass through the top and bottom borders to score**. This allows the player to score points if the ball goes past the top or bottom borders, instead of bouncing back.

Defining Scoring

# Define scoring for ball 1

if hit\_ball.xcor() > 500 or hit\_ball.ycor() < -300: # rev 2: added "or hit\_ball.ycor() < -300"

hit\_ball.goto(0, 0)

hit\_ball.dy \*= -1

left\_player += 1

# rev 4 - add sound

# os.system('afplay /System/Library/Sounds/Sosumi.aiff') # all rev 4 sound commented out in rev 4a

sketch.clear()

sketch.write("Left\_player : {} Right\_player: {}".format(

left\_player, right\_player), align="center",

font=("Courier", 24, "normal"))

# rev 2: (was) if hit\_ball.xcor() < -500:

if hit\_ball.xcor() < -500 or hit\_ball.ycor() > 300: #rev 2: added "or hit\_ball.ycor() > 300"

hit\_ball.goto(0, 0)

hit\_ball.dy \*= -1

right\_player += 1

# os.system('afplay /System/Library/Sounds/Sosumi.aiff') # all rev 4 sound commented out in rev 4a

sketch.clear()

sketch.write("Left\_player : {} Right\_player: {}".format(

left\_player, right\_player), align="center",

font=("Courier", 24, "normal"))

# Define scoring for ball 2

if hit\_ball2.xcor() > 500 or hit\_ball2.ycor() < -300: #rev 2: added "or hit\_ball2.ycor() < -300"

hit\_ball2.goto(0, 0)

# stopplay() # rev3d removed stopplay - game freezes too often; also, if implement, need to do on both balls

hit\_ball2.dy \*= -1

left\_player += 1

# os.system('afplay /System/Library/Sounds/Sosumi.aiff') # all rev 4 sound commented out in rev 4a

sketch.clear()

sketch.write("Left\_player : {} Right\_player: {}".format(

left\_player, right\_player), align="center",

font=("Courier", 24, "normal"))

if hit\_ball2.xcor() < -500 or hit\_ball2.ycor() > 300: #rev 2: added "or hit\_ball2.ycor() > 300"

hit\_ball2.goto(0, 0)

# stopplay() # rev3d removed stopplay - game freezes too often; also, if implement, need to do on both balls

hit\_ball2.dy \*= -1

right\_player += 1

# os.system('afplay /System/Library/Sounds/Sosumi.aiff') # all rev 4 sound commented out in rev 4a

sketch.clear()

sketch.write("Left\_player : {} Right\_player: {}".format(

left\_player, right\_player), align="center",

font=("Courier", 24, "normal"))

This does several things related to scoring:

* The IF statement defines the condition for scoring as when the ball coordinates go past left, right, top, and bottom limits.
* Moves the ball back to center
* Increments the proper player’s score
* Changes the initial vertical direction

Define Ball / Paddle Collision

# Paddle ball collision

#right paddle

if (hit\_ball.xcor() > 360 and hit\_ball.xcor() < 370) and \

(hit\_ball.ycor() < right\_pad.ycor() + 50 and hit\_ball.ycor() > right\_pad.ycor() - 50):

hit\_ball.setx(360)

hit\_ball.dx \*= -1

# os.system('afplay /System/Library/Sounds/Sosumi.aiff')

# # os.system("afplay bounce.wav&")# Adding sound rev 4

# left paddle

if (hit\_ball.xcor() < -360 and hit\_ball.xcor() > -370) and \

(hit\_ball.ycor() < left\_pad.ycor() + 50 and hit\_ball.ycor() > left\_pad.ycor() - 50):

hit\_ball.setx(-360)

hit\_ball.dx \*= -1

# os.system('afplay /System/Library/Sounds/Sosumi.aiff') # all rev 4 sound commented out in rev 4a

# rev 2 add for top and bottom paddles; swap x and y; change 360 to 160 and 370 to 170

#top paddle # rev 2

if (hit\_ball.ycor() > 160 and hit\_ball.ycor() < 170) and \

(hit\_ball.xcor() < top\_pad.xcor() + 50 and hit\_ball.xcor() > top\_pad.xcor() - 50): # rev 2

hit\_ball.sety(160) # rev 2

hit\_ball.dy \*= -1 # rev 2

# os.system('afplay /System/Library/Sounds/Sosumi.aiff') # all rev 4 sound commented out in rev 4a

# bottom paddle # rev 2

if (hit\_ball.ycor() < -160 and hit\_ball.ycor() > -170) and \

(hit\_ball.xcor() < bottom\_pad.xcor() + 50 and hit\_ball.xcor() > bottom\_pad.xcor() - 50): # rev 2

hit\_ball.sety(-160) # rev 2

hit\_ball.dy \*= -1 # rev 2

# os.system('afplay /System/Library/Sounds/Sosumi.aiff') # all rev 4 sound commented out in rev 4a

# end of rev 2 changes for ball 1

# Paddle ball collision for Ball 2

if (hit\_ball2.xcor() > 360 and hit\_ball2.xcor() < 370) and \

(hit\_ball2.ycor() < right\_pad.ycor() + 50 and hit\_ball2.ycor() > right\_pad.ycor() - 50):

hit\_ball2.setx(360)

hit\_ball2.dx \*= -1

# os.system('afplay /System/Library/Sounds/Sosumi.aiff') # all rev 4 sound commented out in rev 4a

if (hit\_ball2.xcor() < -360 and hit\_ball2.xcor() > -370) and \

(hit\_ball2.ycor() < left\_pad.ycor() + 50 and hit\_ball2.ycor() > left\_pad.ycor() - 50):

hit\_ball2.setx(-360)

hit\_ball2.dx \*= -1

# os.system('afplay /System/Library/Sounds/Sosumi.aiff') # all rev 4 sound commented out in rev 4a

# rev 2 add for top and bottom paddles; use previous rev 2 changes and change ball1 to ball2

#top paddle # rev 2

if (hit\_ball2.ycor() > 160 and hit\_ball2.ycor() < 170) and \

(hit\_ball2.xcor() < top\_pad.xcor() + 50 and hit\_ball2.xcor() > top\_pad.xcor() - 50): # rev 2

hit\_ball2.sety(160) # rev 2

hit\_ball2.dy \*= -1 # rev 2

# os.system('afplay /System/Library/Sounds/Sosumi.aiff') # all rev 4 sound commented out in rev 4a

# bottom paddle # rev 2

if (hit\_ball2.ycor() < -160 and hit\_ball2.ycor() > -170) and \

(hit\_ball2.xcor() < bottom\_pad.xcor() + 50 and hit\_ball2.xcor() > bottom\_pad.xcor() - 50): # rev 2

hit\_ball2.sety(-160) # rev 2

hit\_ball2.dy \*= -1 # rev 2

# os.system('afplay /System/Library/Sounds/Sosumi.aiff') # all rev 4 sound commented out in rev 4a

# end of rev 2 changes for ball 1

**Ball / Paddle Collision**: The collision detection is based on comparing the ball's position to the paddles' positions (both in the x and y directions). If a collision is detected, the ball's position is adjusted to prevent overlapping, and its movement direction (dx for horizontal or dy for vertical) is reversed to create the bounce effect. The logic is used for collisions with **left**, **right**, **top**, and **bottom paddles**, with special adjustments for top and bottom paddles introduced in **rev 2**. **Ball 2**, the same collision logic applies but for the second ball object (hit\_ball2).

*.*