CISC 120: Lab 4
Pong

Name: Andrei Suslov

Date: Nov 18 2023 **Section:** Module 4

Contents

1 Introduction

This programming project is designed to test your understanding of graphics and animation in Python, as well as defining and using small functions. It is not based directly on a textbook exercise.

> You will write a program which implements the game of Pong. In this two player game, each player controls a paddle located on the left and right edges of the screen. A ball starts in the center of the screen and moves off in a random direction. This ball will bounce off of the top and bottom edges of the window, as well as off of the paddles.
>
> If the ball strikes the left or right edges of the screen, then the player opposite that side wins the round. Thus, the goal of the game is to move the paddles to bounce the ball back at the opponent, without letting it slip past.
>
> You will implement this game using the stddraw module. For details, use the `help` function, or see pages 158-170 in the textbook [1].

2 Initial Design

High Level View

Although the problem sounds pretty complex at first blush, we will see that creating a game is not actually as hard as it seems. You have already created animations of balls bouncing off of obstacles in the previous lab–all that this problem adds is a user-input component.

To begin with, you will need to have a basic idea of what it is that you will need to do to make the game work. Then, we can work on implementing each of these pieces, one at a time.

At a high level, the steps that our program must do are,

1. Draw the paddles and ball
2. Accept user input from the keyboard to determine how to move the paddles.
3. Move the paddles and ball
4. Check if the ball has collided with something,
    - if the ball has collided with a paddle or the top or bottom of the screen, bounce it.
    - if the ball has collided with the left or right of the screen, declare the player opposite that side the winner and stop.
5. Repeat from step 1.

Step 4 will be potentially tricky in comparison to the similar checks that you implemented in the previous lab, because the positions of the paddles are not static. The location of the edge of the paddle will vary with user input, and so the check must be a bit more complicated. But for the moment that is ahead of us, we will reach that complication in due time.

To write the program that implements the above process, we will use a "bottom-up" approach. We will immediately dive into coding small components of the problem, and move onto the next as we get each step working. This is a very good approach for tackling complex problems, because it keeps the amount of code that you write at any one time small. This makes testing and debugging of the code easy.

A common technique for doing this is to start by defining several empty functions, one for each step in the program. Then you can fill these in one by one. These functions are not fixed, static things. As you work through the problem, you may find that some of your initial functions are unnecessary, or that others would be best split up into several smaller functions. This is *okay*–the odds of your first glance decomposition of the problem into sub-components being optimal are identically equal to 0. Treat this first-pass design as more of a guideline than as a set of hard and fast rules.

For each section in this document, you will implement one of, or part of, these steps. As you are going this, **save a different Python file for each step.** You will need to turn in your code specifically for each step at the end. What you should do is, at the conclusion of a section, create a copy of your code, and then use the copy as the basis for your code for the next section.

Begin using the following template,

```
import stddraw
import stdio
import math
import random

def main():
    pass


if __name__ == '__main__':
    main()
```

For each of the steps listed above, define a function above `main` with a descriptive name. Then, place calls to these functions within `main`, in the appropriate order. You should find that you need a loop. Simply use an infinite while loop (`while True:`) for this, and place it within the `main` function itself. Also include the appropriate calls to `stddraw.show` and `stddraw.clear` within this while loop, inside of `main`, as well. If you're not sure where exactly to put them just yet, give it your best shot. You can always move them later if you need to! Call this file `pong_template.py`. **10pts.**

Leave these functions parameterless for the moment. As you tackle each one, you will determine what, if any, parameters are needed by that function. For now we are operating with a very high-level view of the problem.

The body of each function should contain `pass`, so that you can execute your program without syntax errors. As an example, see the following,

```
def first_function():
    pass

def second_function():
    pass

def main():
    while True:
        first_function()
        section_function()
```

```
if __name__ == '__main__':
    main()
```

This would also be a good time to make any decisions about canvas size and scaling. If you would like to use a canvas that has a different size than the default 500-by-500 pixels, or different scales than $[0, 1]$ on each axis, it'll be easiest to choose those values now. It is fine to leave them default, but if you would like to change them, feel free. If you do change them, place your calls to `setCanvasSize`, `setXscale`, and `setYscale`, just before your while statement, within `main`.

WARNING 1 *Be very careful with* ***`setCanvasSize`****. This function can only be called* once *in a given program. If you call this function a second time, your program will throw an error and crash.*

3 Implementing the Game

Drawing the Paddles and Ball

A good first step will be to simply draw the paddles and the ball on the canvas in their starting locations. This should be the task of your first function.

The stddraw module is designed such that it can be easily used inside of functions. The background and foreground canvases used by the module are **global**. This means that all calls to stddraw within a program are drawing upon the same canvas. So we can use calls to stddraw within all of our functions without needing to use any parameter passing to get variables within scope.[1]

Modify the function that you defined above for drawing the paddles and ball by placing within it calls to stddraw's functions to draw these objects. Call this file `pong_setup.py`. **10pts.**

Figure 1 contains and example of what this should look like.

Obviously, there is a lot of room for personal decision making here. You as the programmer get to decide exactly how far from the edge the paddles are located, how tall they are, how wide they are, etc. However they should be equally distant from their respective edge of the screen, and should start vertically centered. The ball should start in the exact center of the screen.

For the moment, leave the function parameterless. This will need to change, because eventually we will need to update the positions of the paddles and ball, but for now draw a static image.

With that said, you should also avoid "magic numbers" in code. This means that you

[1] This is a common technique used in something called **procedural programming**, which is a way of structuring programs that we have been using up to this point. An alternative paradigm, **object-oriented programming**, takes a dim view on this sort of shared global state. We will discuss OOP towards the end of the semester.

generally want to avoid simply writing numerical literals into formulae and function calls. Instead, create a variable. For example, rather than using (numbers nonsensical),

```
stddraw.filledRectangle(1.5, 1.2, .5, 5)
```

do the following, [2]

```
p1_pdl_x = 1.5
p1_pdl_y = 1.2
p1_pdl_w = .5
p1_pdl_h = 5

stddraw.filledRectangle(p1_pdl_x, p1_pdl_y, p1_pdl_w, p1_pdl_h)
```

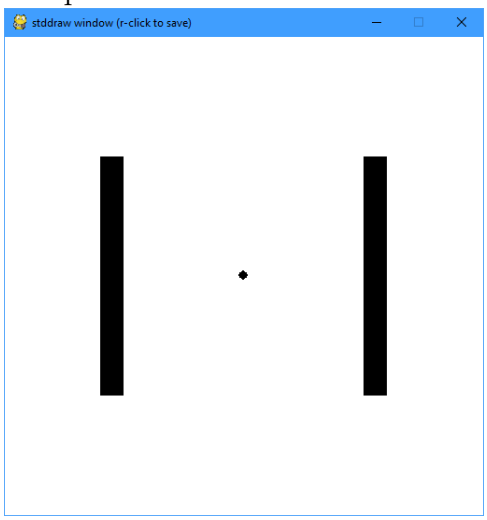
This is useful for two reasons. First, it will make your code a lot easier to read. For example, we now know that this call will draw the paddle for player 1 (just from the variable names!) and what the different arguments mean. Additionally, it will make this function much easier to parameterize later, because we can simply bump one of more of these variables up to the parameter list to accept them as input.

Note that you can re-use variables for both paddles. For example, it seems reasonable that both paddles will have the same height and width. So rather than having,

```
p1_pdl_w = .5
p1_pdl_h = 5
```

[2] Variable names can get tricky. You want the name to be clear as to what the variable is, but not so clear that is is annoying to type. For example, `px1` is probably a bad name, but so is `player1_paddle_x_coordinate`. There are a lot of other considerations here too. Check out pages 238-242 of *The Pragmatic Programmer* for other details here [2].

Figure 1: An example of the initial state of the canvas for Pong.



```
p2_pdl_w = .5
p2_pwl_h = 5
```

It might make more sense to just have a single variable for each height and width that is used to draw both paddles,

```
pdl_w = .5
pdl_h = 5
```

Accepting User Input

The stddraw module also supports accepting input from the keyboard! This is a feature that is not described in the textbook; the functions used are described in the help text for the module. The system is set up fairly similarly to reading input tokens from stdin, except it reads individual key presses instead.

The functions used are,

```
stddraw.hasNextKeyTyped()
        Return True if the queue of keys the user typed is not
        empty. Otherwise return False.

stddraw.nextKeyTyped()
        Remove the first key from the queue of keys that the
        user typed, and return that key.
```

This setup is fairly limited, and will only work for rudimentary games. But, as luck would have it, Pong is a rudimentary game.

To accept key presses as input, stddraw uses a technique called **polling**. Polling consists of having the program repeatedly ask if there is a key that has been typed, and then responding to the key press if the answer is yes. When polling, it is the program's main loop that responds to key presses. A key that has been pressed will be stored until the program asks for it, and only then returned.[3]

As a simple example, here is a program that will simply echo pressed keys back to the user. Run this code, and focus the blank stddraw window that appears. While this window is selected, any keys you press will be written to the terminal in which you are running the program.

```
import stdio
import stddraw

while True:
    stddraw.show(0)
    if stddraw.hasNextKeyTyped():
```

[3]This is in contrast to a system based around **interrupts**, where the key press event immediately and directly triggers code within the program to execute. In such a system, when a key is pressed the program will drop what it is doing immediately to respond to the event.

```
key = stddraw.nextKeyTyped()
stdio.write(key)
```

Not every key press will result in something being written on the terminal. This module is limited in the key presses that it can accept–stick to the alphabetic keys. The `nexKeyTyped` function will return a string containing the letter of the key pressed. If you press the `k` key on your keyboard, it will return `'k'`, for example.

Your program will need to accept 4 commands from the keyboard. These are,

- player 1 paddle up
- player 1 paddle down
- player 2 paddle up
- player 2 paddle down

Decide upon which key you would like to map to each of these operations, and then implement the second function in your template. This function should check to see if a key has been pressed, and return something that tells the program what action to take. You could simply return the string representing the key press itself, or map the key pressed onto something else (maybe an integer), which you then return. In any case, from the return of this function, your program should have the information it needs to determine what to do next.

If no key is pressed, the function should just return None. Call this file `pong_input.py`. **10pts.**

Moving the Paddles

Next up, you will need to move the paddles up and down in response to the paddle up/down commands. This will not map cleanly onto a single function, because it will require interfacing the two functions that you have just written. When the keyboard input function returns a paddle up/paddle down command for either player, the y-coordinate of the paddle will need to be raised or lifted in the paddle drawing function that you wrote initially.

Thus, in order to accomplish this, you will need to parameterize this paddle drawing function, so that the y-coordinates of the paddles are inputs to the function. If you properly defined variables instead of using magic numbers when first writing this function, then the update to the function to accept these numbers as input should be trivial.

Move the y-coordinate variable for both the paddles out into the `main` function. If a key press would result in a paddle moving, update the value of these variables accordingly. Then, pass them as input to the paddle drawing function.

Once you get this working, if you run the program, you should be able to move both of the paddles up and down by pressing on the corresponding keys. It may take some experimentation with the code to get the paddles to move smoothly. Call this file `pong_paddles.py`. **10pts.**

Collision Detection – Boundaries

Once the paddles are moving, you will also need to make the ball move. Just like in the previous lab animations, this can be done by defining an x and y velocity for the ball, and then updating the coordinates of the ball by that amount each time through the main program loop. Give both of these velocities a reasonable, random value to start (using a random number generator, it should be different each time). It may require some experimentation to get random number ranges that work relatively well here.

That is simple enough. However, you will also need to make the ball bounce around! As a start, consider only the boundaries of the screen and ignore the paddles. You will want to write code so that, when the ball strikes the top or bottom boundaries of the screen, it "bounces" off. This can be done by reversing the sign on the ball's y-velocity.

Additionally, when the ball strikes the left or right edge of the screen, that means that a player has won. Also add code to detect this case. When it happens, the game should stop, and the console running the game should display some text stating which player has won.

Implement these boundary behaviors (again, ignoring the paddles). Call this file `pong_boundaries.py`. **10pts.**

Collision Detection – Paddles

There remains only a single major feature to implement before your pong game is playable. The ball needs to actually bounce off of the paddles.

The act of making the ball bounce is not in-and-of-itself difficult. Simply negate the x-velocity of the ball. The tricky part will be figuring out when the ball has struck the paddle, given that the paddles themselves are moving around.

Implement this behavior. When the ball strikes the edge of a paddle, you should reverse its x-velocity in order to make the ball bounce. Once you've done this, you should have a fully functional (albiet very basic) game of pong to play with your friends. Call this file `pong_complete.py`. **15pts.**

Other Features

If you've made it this far, you have a working game of pong. However, it is very basic and there are a lot of other features that you may want to implement. Here's a list of a few ideas to experiment with; try these out and see how many you can get working! These are not for points/credit, only for fun.

1. Allow the program to play the game multiple times. When a player wins, have the program reset the game back to its initial state and play again.
2. Implement a difficulty/handicap system. Each player can select a difficulty level (perhaps via command-line arguments) that will control how large that player's paddle is. One player can have a large paddle, and a better player can have a small one.

3. stddraw supports writing text onto the screen. Modify your program so that it displays the win count for each player.

4. Enable a "best of x" feature. Have the game accept an integer as a command-line argument, and then run the game until one player has achieved the necessary number of wins. For example, in a Best of 7 game, the first player to win 4 times is the winner of the whole game.

If you do implement any of this, submit the file as `pong_extra.py`.

4 Submission

To earn full credit for this lab, you must submit at least six documents by the assigned deadline. These are the Python files associated with each question, `pong_template.py`, `pong_setup.py`, `pong_input.py`, `pong_paddle.py`, `pong_boundaries.py`, and `pong_complete.py`. Additionally, if you implemented any additional features, make sure to turn in `pong_extra.py` as well!. If you fail to turn in one of these files, you will forfeit any points associated with that file.

These Python files must run. If you call a program `pong_complete.py`, for example, then I will attempt to execute the following on command prompt, with my working directory set to the folder containing all of your submitted documents,

```
% python pong_complete.py
```

If the program throws an error immediately, be it a syntax error, type error, name error, or any other error, then you won't receive any points for it. If you cannot get the program to work, then implement as much of the required functionality as you can without running into any of these errors. This will maximize your opportunity to earn partial credit. It is better to turn in a half-functional program that runs, than an attempt to implement all of the functionality that does not.

References

[1] R. Sedgewick, K. Wayne, and R. Dondero, *Introduction to Programming in Python*. Addison-Wesley, 1st ed., 2015.

[2] D. Thomas and A. Hunt, *The Programmatic Programmer*. Pearson, 2nd ed., 2020.

5 Rubric

Question	Points	Score
High Level View	10	
Drawing the Paddles and Ball	10	
Accepting User Input	10	
Moving the Paddles	10	
Collision Detection – Boundaries	10	
Collision Detection – Paddles	15	
Other Features	0	
Total:	65	