Games: Breakout

Getting Started

Up until this point in the course, you’ve been doing all of your assignments in CS50 IDE. This problem, however, cannot be completed in an online IDE as it will employ graphics that the IDE cannot render. You will need to install the following software onto your computer: python3 (<https://www.python.org/downloads>), which usually includes the tkinter library (if not included this must also be installed.) You will also need to install the graphics.py library, (<http://mcsp.wartburg.edu/zelle/python/graphics.py)> or simply place it in the folder that your python programs reside in, and download a simple python text editor like Atom ([https://atom.io](https://atom.io/)).

You can then use Atom as your text editor and your mac or windows terminal as your terminal window.

The reference for graphics.py is available at: <http://mcsp.wartburg.edu/zelle/python/graphics/graphics.pdf>

Backstory

One day in the late summer of 1975, Nolan Bushnell [founder of Atari and, um, Chuck E. Cheese’s], defying the prevailing wisdom that paddle games were over, decided to develop a single-player version of Pong; instead of competing against an opponent, the player would volley the ball into a wall that lost a brick whenever it was hit. He called [Steve] Jobs into his office, sketched it out on his little blackboard, and asked him to design it. There would be a bonus, Bushnell told him, for every chip fewer than fifty that he used. Bushnell knew that Jobs was not a great engineer, but he assumed, correctly, that he would recruit [Steve] Wozniak, who was always hanging around. "I looked at it as a two-for-one thing," Bushnell recalled. "Woz was a better engineer."

Wozniak was thrilled when Jobs asked him to help and proposed splitting the fee. "This was the most wonderful offer in my life, to actually design a game that people would use," he recalled. Jobs said it had to be done in four days and with the fewest chips possible. What he hid from Wozniak was that the deadline was one that Jobs had imposed, because he needed to get to the All One Farm to help prepare for the apple harvest. He also didn’t mention that there was a bonus tied to keeping down the number of chips.

"A game like this might take most engineers a few months," Wozniak recalled. "I thought that there was no way I could do it, but Steve made me sure that I could." So he stayed up four nights in a row and did it. During the day at HP, Wozniak would sketch out his design on paper. Then, after a fast-food meal, he would go right to Atari and stay all night. As Wozniak churned out the design, Jobs sat on a bench to his left implementing it by wire-wrapping the chips onto a breadboard. "While Steve was breadboarding, I spent time playing my favorite game ever, which was the auto racing game Gran Trak 10," Wozniak said.

Astonishingly, they were able to get the job done in four days, and Wozniak used only forty-five chips. Recollections differ, but by most accounts Jobs simply gave Wozniak half of the base fee and not the bonus Bushnell paid for saving five chips. It would be another ten years before Wozniak discovered (by being shown the tale in a book on the history of Atari titled *Zap*) that Jobs had been paid this bonus….

*Steve Jobs*

— Walter Isaacson '74

Breaking Out

Your challenge for this problem is to implement the same game that Steve and Steve did, albeit in software rather than hardware. That game is Breakout. If you completed Pong, download as described below and skip the rest of this intro to graphics.py.

Whereas all of your C programs to date have only had "command-line interfaces" (CLIs), this one will have a graphical user interface (GUI), not unlike Scratch! You’ll be building Breakout atop graphics.py, which is similar in spirit to the CS50 Library but includes a simple object oriented graphics library.

Let's take a look at what you can do with graphics.py by way of some code examples. In your terminal copy and paste the following. If you don’t have git installed, copy and paste the url into your browser.

git clone https://github.com/cs50nestm/breakout.git

Make sure graphics.py is the same the same breakout folder.

In your terminal type:  
  
 cd breakout

If you then execute ls, among the files you see should be bounce.py and window.py. In your terminal window, go ahead and execute window.py as follows:

python3 window.py

A window quite like the one below should appear and then disappear after about 5 seconds.

Neat, right? Go ahead and open window.py in Atom.

The first thing you will notice is:

from graphics import \*

which is similar to the idea of header files in c. You should already have graphic.py installed on your computer, or residing in the same folder as your python files. This program is what gives your python programs graphics capabilities.

Next you see:

win = GraphWin("window", 320, 240)

which creates or "instantiates" a new window, with a width of 320 and a height of 240.

Finally, you will see:

time.sleep(5)

win.close()

exit(0)

which is why your window stays open for 5 seconds and then closes. Finally, your program ends with:

exit(0)

which is the equivalent of return 0 from main in c.

Now let's look at a program with a window and a circle with the circle bouncing back and forth. Go ahead and open bounce.py.

First you'll see that a 320 by 240-pixel window is instantiated. Then a point is defined with (x, y) coordinates of (160, 120) which is used as the center of a circle with a radius of 20 pixels, which is instantiated on the next line.

center = Point(160, 120)

ball = Circle(center, 20)

ball.setFill("BLACK")

ball.draw(win)

Finally the fill of the ball is set to black, and it is drawn in the window.

Next, a variable velocity, which will represent the speed the ball moves, is set to 2 pixels per unit of time.

Now to make the circle bounce back and forth, we'll create an infinite loop, and inside the loop for every unit of time, we're going to move the circle 2 pixels. Then we're going to have to check if the ball touches the right edge of the window or the left edge of the window, because if it does we need to make it bounce.

So first we make the ball move along the x-axis, but not at all along the y-axis.

ball.move(velocity, 0)

Then we get the x-coordinate of the center of the circle so we know where it is after it moves.

centerBall = ball.getCenter()

xBall = centerBall.getX()

We add the radius of the ball to the x value of the center to find the coordinates of the right edge. If this value exceeds the width of the window, we know the ball has reached the right edge. We can make the ball "bounce" so to speak, by reversing the velocity when it reaches the right edge.

if xBall + 20 > 320:

velocity = -velocity

We do the same to check if the ball reaches the left edge of the window:

if xBall - 20 < 0:

velocity = -velocity

Finally, we check for a mouse click, which tells us to close the window, break out of the loop, and end the program.

if win.checkMouse():

win.close()

break

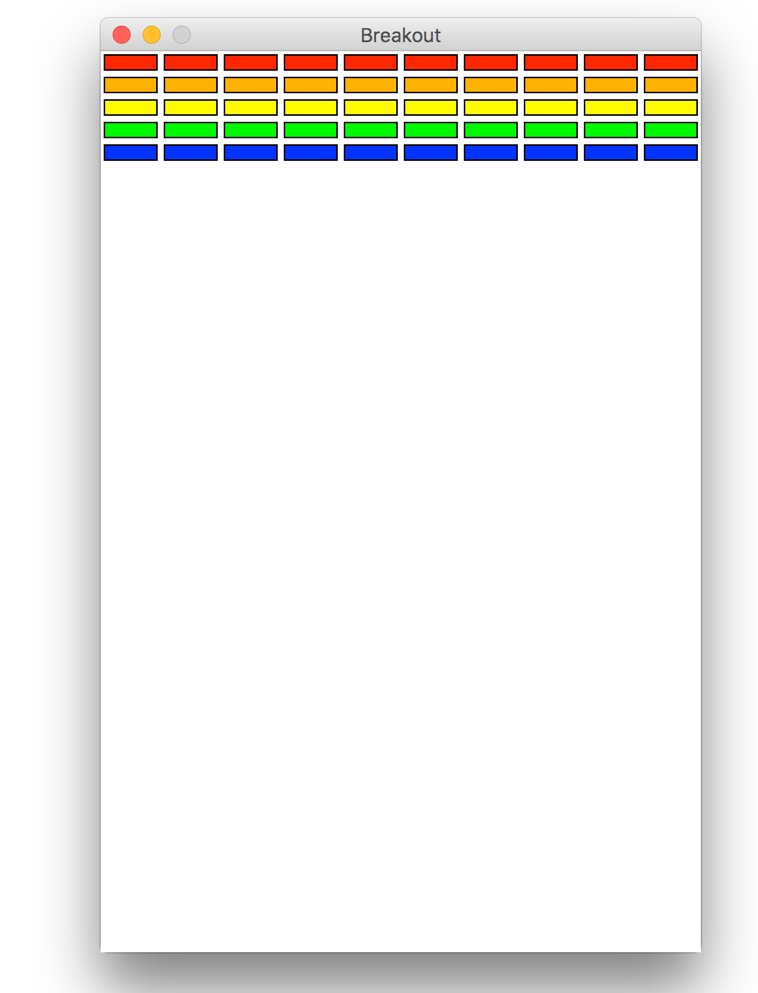
exit(0)

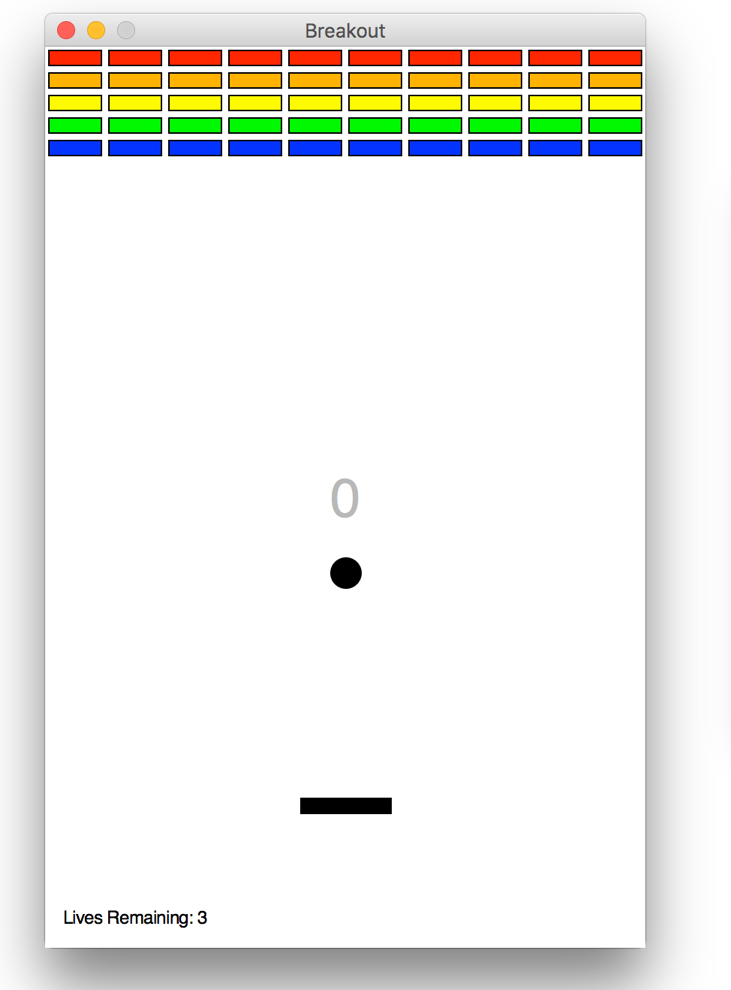
breakout

Okay, let’s see what the distribution code for breakout itself does. Go ahead and execute:

python3 breakout.py

A window like the window below should appear.



You can click anywhere inside the window to close it. Not much of a game. Yet! Ultimately your game should look something like this:

The goal, quite simply, is to bounce the ball off of the paddle, moving the paddle with your arrow keys, so as to hit bricks with it. If you break all the bricks, you win! But if you miss the ball three times, you lose!

Nice. Let’s make your implementation look more like that one. But, first, a tour!

Open up breakout.py with Atom and take a moment to scroll through it to get a sense of what lies ahead.

The first thing you may want to do is to remove lines 42, 107, 127 and 205 which commented out major sections of the code so you could execute it above.

* Atop the file you will find two import statements.
* Next up are some constants, values that you don’t need to change, but because the code we’ve written (and that you’ll write) needs to know these values in a few places, we’ve factored them out as constants so that we or you could, theoretically, change them in one convenient location. By contrast, hard-coding the same number (pejoratively known as a "magic number") into your code in multiple places is considered bad practice, since you’d have to remember to change it, potentially, in all of those places.
* After the constants (which are really just variables in Python), bricks is declared as a list, with global scope, so that we can access the bricks from any function.
* Next, we instantiate a window.
* Then, you'll see a main function, which calls initBricks(), a function already written for you, that instantiates a grid of bricks atop the game’s window.
* After this, main calls initBall() which instantiates the ball that will be used to play Breakout.
* Called by main next is, initPaddle(), which instantiates the paddle, and returns the rectangle as the object paddle.
* Main then calls functions to instantiate the scoreboard, and the "lives" scorekeeper, using functions which have already been defined for you.
* The Python random() function is called which generates a pseudorandom number in the range [0.0, 1.0). This is then used to set the xvelocity.
* The function win.getMouse() waits for a mouse click to start the game.
* After initializing a few variables, we wait for a click and then start the game using a loop. Of course there's not much code in that loop now!
* Below the loop is a call to win.getMouse(), a function that waits for a mouse click so that the window doesn’t close until the user intends.
* Not too bad, right? Let's take a closer look at the functions which you are going to complete.
* In initBall(), you will write code that instantiates and then returns a ball whose radius is RADIUS, which is positioned in the center of the window, just below the scoreboard.
* In initPaddle(), you'll write code that instantiates a paddle (a rectangle), that's somehow centered in the bottom-middle of the game's window.
* Finally in checkEdge(xBall), you will write a function that takes as it's parameter the center point of the ball, xBall, returns True if it touches the left or right edge of the window, and False if it does not.
* The other functions that you may want to use are already written for you. The function paddleMove(paddle) takes as an input parameter the object paddle, and moves the paddle left or right when you press on the left or right arrow keyboard keys.
* The function padHit, takes the object paddle, and the x and y-coordinates of the center point of the ball as input parameters, and returns true if the paddle hits the ball and false if it does not.
* The function initLives, initializes the "Lives Remaining:" text on the bottom left of the scoreboard, updateLives, can update the lives remaining text in the window for you, initScoreboard initializes the scoreboard with 0 points, checkCollision returns true if the ball collides with a brick, and false if it does not, and updateScoreboard, gameOver and youWin are pretty self explanatory.

Alright, ready to break out Breakout?

If you’re like me, odds are you’ll find it easiest to implement Breakout via some baby steps, each of which will get you closer and closer to a great outcome. You may want to comment out sections of code until you are ready to utilize them. That way you can try to execute a little bit at a time. Rather than try to implement the whole game at once, allow us to suggest that you proceed as follows:

1. Now initPaddle()'s only purpose is to instantiate and return a paddle. It shouldn't handle any of the paddle's movement. To construct a rectangle using graphics.py, you provide points which represent the diagonally opposite corners, as in   
     
    paddle = Rectangle(Point(1, 3), Point(4, 7))  
     
   In the graphics.py module, Point(x, y) is an object you can define with the coordinates (x, y).
2. Implement initBall(). The radius is already defined for you in RADIUS. Don't forget to return ball.
3. Complete the function checkEdge, so that it checks if the ball touches the left or right sides of   
   the window. If it does, checkEdge returns True, if it does not, it returns False.
4. Then back in the main function, you will write code to complete the while loop. You'll have to write some code that compels the ball to move. Look to bounce.py to get ideas on how to implement this.
5. Refer to the graphics.py reference to figure out how to get the coordinates of the center of the ball (you may want to set a variable equal to this value) and then get the x and y coordinates of this center point (xBall and yBall).
6. Use the function checkEdge to bounce the ball off of the left and right sides of the window.
7. Now decide how to determine whether the ball has zoomed past the paddle and struck the window’s bottom edge, in which case the user should lose a life. If you have no lives left, call gameOver. If you lives remaining, gameplay should probably sleep for 2 or 3 seconds, after which you call initBall to instantiate a new ball.
8. Call paddleMove, to allow the right and left arrow keys to move the paddle.
9. Call padHit to determine if the paddle hit the ball. If true, reverse the yvelocity of the ball.
10. Finally, for each brick in your list (a.k.a. array) of bricks, use checkCollision, to determine if the ball hit that brick. If this function returns true, remove the brick from the list, reverse the yvelocity, decrease the number of bricks by 1, increase the score by 1, and update the scoreboard. If there are no more bricks remaining, call youWin!

When you've completed the steps above, you are done! Feel free to personalize your implementation to make it uniquely yours!

Because this game expects a human to play, no check50 for this one! Best to invite some friends to find bugs!