Writing classic “Breakout” in Python

Breakout was a game invented in 1976. The object of the game is simple: to bounce a ball in to a wall of bricks until all the bricks are destroyed. In this project, we’ll recreate that classic game in Python.

# Getting Set Up

We’re going need a new Python file, called Breakout.py, so create this in IDLE. We’re going to use the PyGame library to manage all the tricky parts like creating windows, drawing things and handling input from the mouse, but in order to do that we need to “import” PyGame in our code.

Unfortunately, we have to do a fairly boring exercise of getting set up to actually write the game. Think of it as getting your fingers warmed up! Add the following code to your Breakout.py program:

**from** pygame.locals **import** \*  
**import** pygame  
**import** pygame.key  
  
*# Initialise PyGame and some variables*pygame.init()  
scale = 1  
screen = pygame.display.set\_mode((644, 480))  
s\_width, s\_height = screen.get\_size()  
  
*# Used to set the FPS*clock = pygame.time.Clock()  
*# The state of the game (0 = running, 1 = before start, 2 = game over, 3 = quit*game\_state = 0  
  
  
**def** draw\_game():  
 screen.fill((0, 0, 0))  
 pygame.display.update()  
  
  
**def** check\_user\_input():  
 **global** game\_state  
 **for** event **in** pygame.event.get():  
 **if** event.type == QUIT:  
 *# Exit the game immediately if the close window icon is pressed.* game\_state = 3  
  
  
*# The main loop of the game***def** main\_loop():  
 **while** game\_state != 3:  
 clock.tick(60)  
 check\_user\_input()  
 draw\_game()  
  
  
*# Start the game*main\_loop()  
  
*# Terminate PyGame gracefully*pygame.quit()

Run the code and the output should just draw a window on the screen, painted black, that you can close again.

## Adding a Bat

Let’s add a bat to our breakout game. The bat is going to be controlled by the mouse.

Start by adding some colours just under the declaration of s\_width and s\_height, near the top of the program:

*# Colours*BLACK = (0, 0, 0)  
WHITE = (255, 255, 255)

We’ll be adding more later, but these will do for now.

*# Bat variables*bat\_pos = (100, 450)  
bat\_size = (100, 5)

These two variables also need to be added, just under the colours is fine. These store co-ordinates in what’s called a *tuple*. The first number is the X co-ordinate (how far across the screen something is) and Y co-ordinate (how far down something is). bat\_pos refers to where the bat is on the screen, bat\_size stores how big the bat is (width then height). Remember that the top left of the screen is (0,0) and the bottom right is (644, 480).

We’ll need two functions to manage our bat:

1. Set the position of the bat according to where the mouse is.
2. Draw the bat (it’s just a rectangle).

Add these functions under your variable declarations, just above where the draw\_game function is:

**def** draw\_bat():  
 pygame.draw.rect(screen, WHITE, (bat\_pos[0], bat\_pos[1], bat\_size[0], bat\_size[1]))  
  
  
**def** move\_bat():  
 **global** bat\_pos  
 mouse = pygame.mouse.get\_pos()  
 bat\_pos = (mouse[0], bat\_pos[1])

Once you’ve added them to the program they won’t actually do anything until we actually use them. Using a function is referred to as “calling” it. So,we need to add a call to move\_bat inside the main\_loop:

*# The main loop of the game***def** main\_loop():  
 **while** game\_state != 3:  
 clock.tick(60)  
 check\_user\_input()  
 **move\_bat()** draw\_game()

… and a call to draw\_bat inside draw\_game:

**def** draw\_game():  
 screen.fill((0, 0, 0))  
 draw\_bat()  
 pygame.display.update()

Run your code and you should have a bat that you can move with the mouse!