Source Code

Code an explosive homage to Bomb Jack

Help Jack collect the bombs in a remake of the arcade hit

AUTHOR
MARK VANSTONE

In 1984, a new hero flew into arcades. Developed by Tehkan (later Tecmo), *Bomb Jack* saw its caped protagonist leap around platforms, collecting bombs while avoiding the patrolling enemies. Jack gained extra points for collecting bombs with burning fuses, and once the screen was cleared, it was onto the next globe-trotting location. *Bomb Jack* was soon ported to home computers, and received several sequels and re-releases, including one for the Nintendo Switch.

For our Python remake, we'll focus on the unique way Jack moves around the screen. He can jump very high and float back down, while his cape can be used to guide him left and right and glide at a slower pace. Using Pygame Zero, we'll start with the Egyptian background from the original game and then add some platforms over the top. We'll use a different image for each platform and create an Actor for each one. We can put these Actors in a list, which will make it easier to check for collisions later. We also make an Actor for Jack and add a couple of extra properties so that we can track his direction and thrust. Once this is done, we can create Actors for our bombs. We define a list of coordinates to represent the position of each bomb, and then loop through that list creating a bomb Actor in each location and then adding the Actor to a list of bombs.

Each of *Bomb Jack*'s levels had different backgrounds and platform layouts to keep things fresh.

Our `draw()` function will first draw the background followed by the platforms (from the platform list we created earlier), then the bombs from the bomb list, and then Jack. We can also add the score at the top of the screen. You'll see that the bomb Actors also have a property called 'state', which we'll use later when checking to see if Jack has collected them or not. A `gameState` variable can also be set when all the bombs have been collected to trigger the display of a message saying that the level has been completed.

In our `update()` function, we'll handle Jack's movement. We can assume that Jack is facing forward to start with, and then we can adjust his direction depending on what keys are being pressed or if he's jumping up or floating down. We need to do two collision checks because we don't want Jack to jump up through the bottom of a platform, but at the same time, we want him to land on top of a platform if he's falling down. If there are no collisions, then we increase Jack's Y coordinate to simulate gravity, and to counter that, subtract any thrust value that should be applied. The thrust value is reduced each update until it gets to zero. If the left or right keys are pressed and Jack's falling, we move his X coordinate accordingly and give him a little boost upwards to stop him falling so quickly. If he's on a platform, we display animation frames to make him run left or right. We capture the jump key press in the `on_key_down()` function and make the thrust property equal to 20, which will send Jack up in the air.

Collect all the bombs in our homage to *Bomb Jack*.

The other part of the `update()` function checks the state of the bombs. We run through the list of bombs, checking to see if Jack has collided with them. If he has, we set the bomb state to a value higher than 1. This sets off an animation cycle through a few frames until the bomb disappears. In this loop, we count how many bombs have reached the invisible state – when that number reaches the total number of bombs, we set `gameState` to 1 and the level is completed.

To expand the project, you could put in some baddies, and don't forget that lit bomb fuses can give Jack a bonus if he collects them while they're fizzing. As ever, we'll leave those for you to add.

Bomb Jack in Python

Here's Mark's code for a *Bomb Jack*-style platformer. To get it working on your system, you'll first need to install Pygame Zero. Full instructions can be found at **wfmag.cc/pgzero**.

```
# Bomb Jack
import pgzrun
WIDTH = 600
HEIGHT = 650
jack = Actor('jackf',(300,300))
ground = Actor('ground',(300,640))
roof = Actor('roof',(300,61))
platform1 = Actor('platform1',(400,180))
platform2 = Actor('platform2',(420,580))
platform3 = Actor('platform3',(320,440))
platform4 = Actor('platform4',(180,250))
platform5 = Actor('platform5',(120,510))
platformList =
[roof,ground,platform1,platform2,platform3,platform4,platform5]
jack.thrust = gameState = count = frame = score = 0
jack.dir = "l"
bombs = []
bombXY = [(110,95),(170,95),(230,95),(430,95),(490,95),(550,95),(40,290)
,(40,350),(40,410),(40,470),(560,290),(560,350),(560,410),(560,470),(110
,605),(170,605),(230,605),(360,545),(420,545),(480,545)]
for b in bombXY:
    bombs.append(Actor('bomb1', center=(b[0], b[1])))
    bombs[len(bombs)-1].state = 1

def draw():
    screen.blit("background",(0,0))
    for p in platformList: p.draw()
    for b in bombs :
        if b.state > 0 :
            b.image = "bomb"+str(int(b.state))
            b.draw()
    jack.draw()
    screen.draw.text("SCORE:"+str(score), center= (300, 28), owidth=0.5,
ocolor=(255,255,255), color=(255,0,0) , fontsize=40)
    if gameState == 1: screen.draw.text("LEVEL CLEARED", center = (300,
300), owidth=0.5, ocolor=(255,255,255), color=(0,255,255) , fontsize=50)

def update():
    global count, frame
    if gameState == 0:
        jack.dir = "f"
        ytest = jack.y
        if keyboard.up:
            jack.dir = "u"
        if checkCollisions(platformList,(jack.x,jack.y+(jack.height/2)))
== False : jack.y += 4
        if checkCollisions(platformList,(jack.x,jack.y-32)) == False :
jack.y -= jack.thrust
        jack.thrust = limit(jack.thrust-0.4,0,20)
        if jack.y > ytest+1: jack.dir = "d"
        if jack.y < ytest-1: jack.dir = "u"
        if keyboard.left and jack.x > 40:
            if jack.y != ytest:
                jack.dir = "lf"
                jack.y -= 2
            else:
                jack.dir = "l" + str(frame%2+1)
            jack.x -= 2
        if keyboard.right and jack.x < 560:
            if jack.y != ytest:
                jack.dir = "rf"
                jack.y -= 2
            else:
                jack.dir = "r" + str(frame%2+1)
            jack.x += 2
        jack.image = "jack" + jack.dir
        checkBombs()
        count += 1
        if(count%5 == 0) : frame += 1

def on_key_down(key):
    global gameState
    if gameState == 0:
        if key.name == "UP" and jack.dir == "f":
            jack.thrust = 20
            jack.dir = "u"

def limit(n, minn, maxn):
    return max(min(maxn, n), minn)

def checkCollisions(cList, point):
    for i in range(0, len(cList)):
        if cList[i].collidepoint(point):
            return True
    return False

def checkBombs():
    global gameState, score
    bombsCollected = 0
    for b in bombs:
        if b.state > 1: b.state += 0.4
        if b.state == 0 : bombsCollected += 1
        if b.collidepoint((jack.x,jack.y)) and b.state == 1:
            b.state = 1.4
        if int(b.state) > 4 :
            b.state = 0
            score += 100
    if bombsCollected == len(bombs): gameState = 1

pgzrun.go()
```