Source Code

Travel back in time with Braid

Code a time-reversal mechanic straight out of the 2008 indie hit

AUTHOR
MARK VANSTONE

How many times have you played a game and wish you'd done the last move differently? In 2008, *Braid* answered thanks to designer Jonathan Blow's ingenious time-reversal mechanic. If you jumped at the wrong time and fell down a hole, you could rewind time and try again.

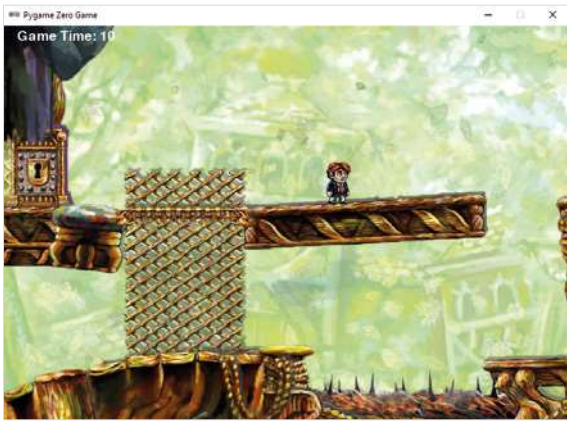
Braid's graphics were created by artist David Hellman, who's generously made the artwork available for download from his website at **wfmag.cc/hellman** – we've used these to recreate the *Braid* time mechanic with Pygame Zero. There's also a Game Developers Conference video where Jonathan Blow describes how he coded the time reversal part of the game (see **wfmag.cc/braid-blow**), so we have a good idea of how to approach this challenge. We'll just make a small section of the game here, but it could easily be expanded using materials provided by the original creators.

Before we get to the reversing time bit, we'll need to create a basic platformer game screen. *Braid* has a parallax scrolling background with organic-looking foreground elements. We can't really make the game environment out of separate blocks, as we might in other retro platformers, so we'll just have one large image for the background and one for the 'platforms' in front, and this will form the environment which protagonist Tim runs around in. We also need another invisible image for collision detection, to test where Tim's moving and whether his path is blocked. On our collision image, white pixels can be moved in, black ones are the platforms, and the blue area represents a rope ladder that Tim can climb.

In *Braid*, Tim solves puzzles and collects jigsaw pieces to save a princess – though there's more to the plot than first appears.

We'll move Tim left and right with the arrow keys. This scrolls the environment along with the background moving half the speed of the foreground to create a parallax effect. There are several frames of animation, both left-facing and right-facing, for Tim running, so we cycle round those while the arrow keys are pressed. We need to check if Tim's standing on a platform, so we check the pixel under his feet on the collision image, and if it's not black, we apply some gravity to him. There's a bit of a fiddle on the gravity to enable Tim to run up inclines. If we move Tim up a pixel before we apply gravity, then that will allow for shallow slopes during movement. We'll also stop Tim from running through walls by checking the direction he's moving in for collision too. We can get Tim to jump with the **SPACE** bar by activating a jumping countdown, with Tim moving upwards as the counter counts to zero, before the gravity calculation kicks in again.

Now we can deal with the time-shifting element. To make the game run backwards, we'll make a list of all the things that are changing on the screen each frame. In this case, there are only a few movable elements, but in the original game, there were many things that needed to be remembered, so it would've started to take up quite a bit of memory. All we need to save in our version is the position, direction, and frame number associated with Tim, and the position of the background and platforms. We put all this data into a list and keep adding to it. We can also put a count of how many seconds have passed (at 60 frames a second) at the top of the screen.



Our homage to the classic *Braid* game. Press 'backspace' to activate the time rewind mechanic.

Pressing **BACKSPACE** switches on rewind. We now look at the last entry in our list and set Tim's attributes and the environment position to reflect that data. We then delete the last entry of the list and move the time counter back. We can actually move two spaces back in the data list and have the rewind happen twice as fast as the normal game speed. And that's about it. The original game had many more complex elements for Tim to contend with, but we'll leave you to have a go at adding them to this quirky, fascinating platform puzzler.

Ride on time

Here's Mark's code for a time-bending rewind mechanic. To get it working on your system, you'll first need to install Pygame Zero – full instructions can be found at **wfmag.cc/pgzero**.


```
# Braid
import pgzrun
from pygame import image

tim = Actor('timr1',(220,400))
tim.dir = "r"
tim.frame = 1
tim.ystore = tim.xstore = tim.jumping = 0
collisionMap = image.load('images/backgroundcol.png')
levelx = backx = count = 0
gameData = []

def draw():
    screen.blit("background1", (backx, 0))
    screen.blit("background2", (levelx, -50))
    tim.draw()
    screen.draw.text("Game Time: "+str(int(count/60)), topleft = (20,
5),color=(255,255,255) , fontsize=28)

def update():
    global levelx,backx,count
    if keyboard.backspace:
        playGameData()
        tim.image = "tim"+tim.dir+str(tim.frame)
    else:
        rgbtop = collisionMap.get_at((int(tim.x - levelx),int(tim.y+30)))
        rgbbottom = collisionMap.get_at((int(tim.x - levelx),int(tim.y+50)))
        tim.ystore = tim.y
        tim.xstore = tim.x
        if keyboard.left:
            if levelx < 0 and checkMove(-2):
                levelx += 2
                backx += 1
                tim.x -= 2
                tim.y -= 1
                tim.dir = "l"
                if count%7 == 0:
                    tim.frame += 1
                    if tim.frame > 5: tim.frame = 2
        if keyboard.right:
            if levelx > -480 and checkMove(2):
                levelx -= 2
                backx -= 1
                tim.x += 2
                tim.y -= 1
                tim.dir = "r"
                if count%7 == 0:
                    tim.frame += 1
                    if tim.frame > 5: tim.frame = 2
        if keyboard.up:
            if rgbtop == (0,0,255) or rgbbottom == (0,0,255):
                tim.y -= 5
                if tim.frame < 9 : tim.frame = 9
                if count%7 == 0:
                    tim.frame += 1
                    if tim.frame > 10: tim.frame = 9
        if keyboard.down:
            if rgbtop == (0,0,255) or rgbbottom == (0,0,255):
                if tim.frame < 9 : tim.frame = 9
                if count%7 == 0:
                    tim.frame += 1
                    if tim.frame > 10: tim.frame = 9
                if rgbbottom != (0,0,0): tim.y += 1
        if tim.jumping == 0: doGravity()
        else:
            if rgbtop == (255,255,255): tim.y -= tim.jumping/3
            tim.frame = 7
            tim.jumping -= 1
        if tim.y > tim.ystore+2: tim.frame = 8
        if tim.x == tim.xstore and tim.y == tim.ystore: tim.frame = 1
        tim.image = "tim"+tim.dir+str(tim.frame)
        count += 1
        storeGameData()

def storeGameData():
    newData = [tim.x,tim.y,tim.dir,tim.frame,levelx,backx]
    gameData.append(newData)

def playGameData():
    global count,levelx,backx,gameData
    if count > 2:
        tim.x = gameData[-1][0]
        tim.y = gameData[-1][1]
        tim.dir = gameData[-1][2]
        tim.frame = gameData[-1][3]
        levelx = gameData[-1][4]
        backx = gameData[-1][5]
        for i in range(2): del gameData[-1]
        count = len(gameData)

def on_key_down(key):
    if key.name == "SPACE":
        rgb = collisionMap.get_at((int(tim.x - levelx),int(tim.y+100)))
        if rgb == (0,0,0):
            tim.frame = 6
            tim.jumping = 12

def doGravity():
    rgb = collisionMap.get_at((int(tim.x - levelx),int(tim.y+25+50)))
    if rgb[0] > 100 or rgb == (0,0,255): tim.y += 3

def checkMove(xinc):
    rgb = collisionMap.get_at((int(tim.x - levelx + xinc),int(tim.y + 50)))
    if rgb == (255,255,255) or rgb == (0,0,255): return True
    return False

pgzrun.go()
```