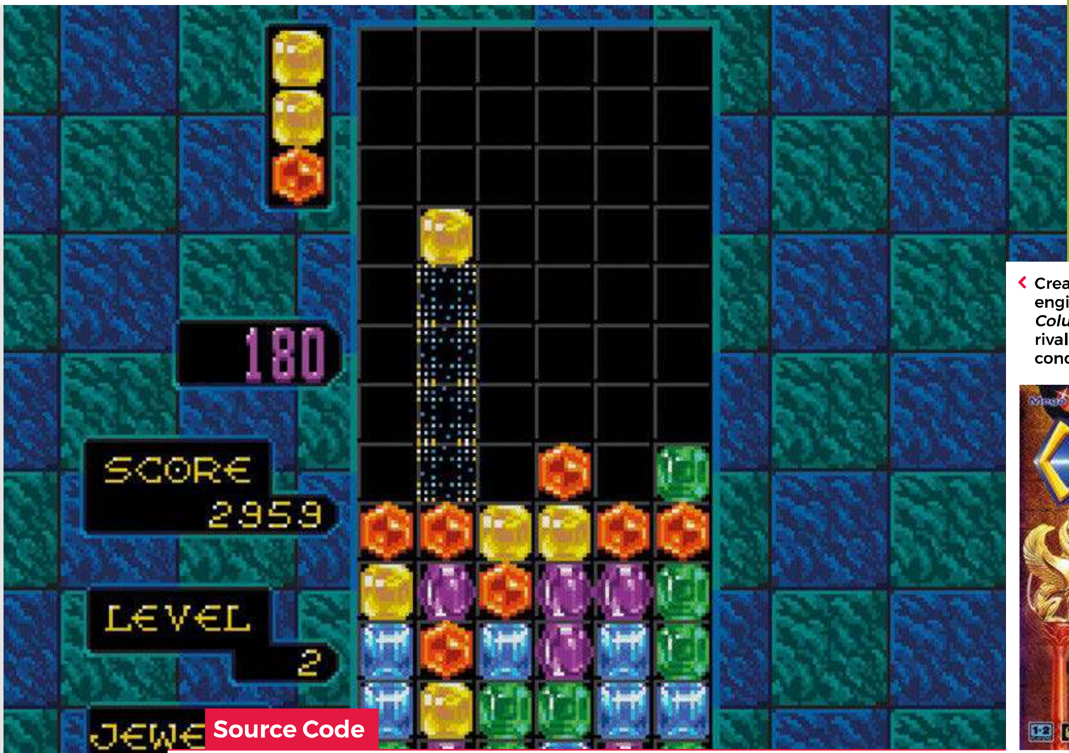
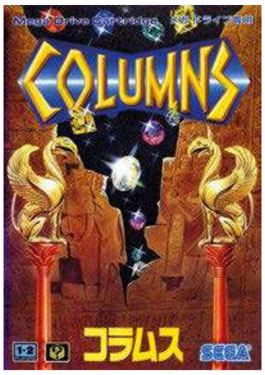


Created by Hewlett-Packard engineer Jay Geertsen, *Columns* was Sega's sparkly rival to Nintendo's all-conquering *Tetris*.



Source Code

Make a tile-matching game

AUTHOR
RIK CROSS

Rik shows you how to code your own Columns-style tile-matching puzzler

Tile-matching games began with *Tetris* in 1984 and the less famous *Chain Shot!* the following year. The genre gradually evolved through games like *Dr. Mario*, *Columns*, *Puyo Puyo*, and *Candy Crush Saga*. Although their mechanics differ, the goals are the same: to organise a board of different-coloured tiles by moving them around until they match.

Here, I'll show how you can create a simple tile-matching game using Python and Pygame. In it, any tile can be swapped with the tile to its right, with the aim being to make matches of three or more tiles of the same colour. Making a match causes the tiles to disappear from the board, with tiles dropping down to fill in the gaps.

At the start of a new game, a board of randomly generated tiles is created. This is made as an (initially empty) two-dimensional array, whose size is determined by the values of **rows** and **columns**. A specific tile on the board is referenced by its row and column number.

We want to start with a truly random board, but we also want to avoid having any matching tiles. Random tiles are added to each board position, therefore, but replaced if a tile is the same as the one above or to its left (if such a tile exists).

In our game, two tiles are 'selected' at any one time, with the player pressing the arrow keys to change those tiles. A **selected** variable keeps track of the row and column of the left-most selected tile, with the other tile being one column to the right of the left-most tile. Pressing **SPACE** swaps the two selected tiles, checks for matches, clears any matched tiles, and fills any gaps with new tiles.

A basic 'match-three' algorithm would simply check whether any tiles on the board have a matching colour tile on either side, horizontally or vertically. I've opted for something a little more convoluted, though, as it allows us to check for matches on any length, as well as track multiple, separate matches. A **currentmatch** list keeps track of the (x,y) positions of a set of matching tiles. Whenever this list is empty, the next tile to check is added to the list, and this process is repeated until the next tile is a different colour. If the **currentmatch** list contains three or more tiles at this point, then the list is added to the overall **matches** list (a list of lists of matches!) and the **currentmatch** list is reset. To clear matched tiles, the matched tile positions are set to **None**, which indicates the absence of a tile at that position. To fill the board, tiles in each column are moved down by one row whenever an empty board position is found, with a new tile being added to the top row of the board.

The code provided here is just a starting point, and there are lots of ways to develop the game, including a scoring system and animation to liven up your tiles.

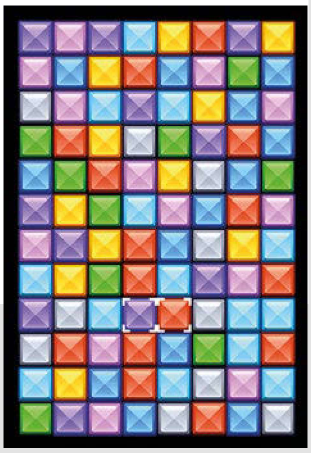
Match-three in Python

Here's Rik's code snippet, which creates a simple match-three game in Python. To get it running on your system, you'll first need to install Pygame Zero – you can find full instructions at **wfmag.cc/pgzero**

Download the code from GitHub: **wfmag.cc/wfmag25**

```
from random import randint

WHITE = 255,255,255

boardx = 40
boardy = 40
tilesize = 40
columns = 8
rows = 12
numberoftiles = 9

WIDTH = (boardx * 2) + (tilesize * columns)
HEIGHT = (boardy * 2) + (tilesize * rows)

tiles = [[1] * columns for j in range(rows)]
for r in range(rows):
  for c in range(columns):
    tiles[r][c] = randint(1, numberoftiles-1)
    while (r>0 and tiles[r][c] == tiles[r - 1][c]) or (c > 0
and tiles[r][c] == tiles[r][c - 1]):
      tiles[r][c] = randint(1, numberoftiles - 1)

selected = [0,0]

def checkmatches():
  matches = []
  for c in range(columns):
    currentmatch = []
    for r in range(rows):

      if currentmatch == [] or tiles[r][c] == tiles[r - 1][c]:
        currentmatch.append((r,c))
      else:
        if len(currentmatch) >= 3:
          matches.append(currentmatch)
        currentmatch = [(r,c)]
    if len(currentmatch) >= 3:
      matches.append(currentmatch)
  for r in range(rows):
    currentmatch = []
    for c in range(columns):
      if currentmatch == [] or tiles[r][c] == tiles[r][c - 1]:
        currentmatch.append((r,c))
      else:
        if len(currentmatch) >= 3:
          matches.append(currentmatch)
        currentmatch = [(r,c)]
    if len(currentmatch) >= 3:
      matches.append(currentmatch)

  return matches
```

```
def clearmatches(matches):
  for match in matches:
    for position in match:
      tiles[position[0]][position[1]] = None

def fillboard():
  for c in range(columns):
    for r in range(rows):
      if tiles[r][c] == None:
        for rr in range(r,0,-1):
          tiles[rr][c] = tiles[rr - 1][c]
        tiles[0][c] = randint(1, numberoftiles - 1)
        while tiles[0][c] == tiles[1][c] or (c > 0 and
tiles[0][c] == tiles[0][c-1]) or (c<columns-1 and tiles[0][c]
== tiles[0][c+1]):
          tiles[0][c] = randint(1, numberoftiles - 1)

def on_key_up(key):
  if key == keys.LEFT:
    selected[0] = max(0,selected[0] - 1)
  if key == keys.RIGHT:
    selected[0] = min(selected[0] + 1,columns - 2)
  if key == keys.UP:
    selected[1] = max(0,selected[1] - 1)
  if key == keys.DOWN:
    selected[1] = min(selected[1] + 1,rows - 1)
  if key == keys.SPACE:
    tiles[selected[1]][selected[0]], tiles[selected[1]]
[selected[0] + 1] = tiles[selected[1]][selected[0] + 1],
tiles[selected[1]][selected[0]]
    matches = checkmatches()
    clearmatches(matches)
    fillboard()

def draw():
  screen.clear()
  for r in range(rows):
    for c in range(columns):
      screen.
blit(str(tiles[r][c]),
(boardx + (c * tilesize),
boardy + (r * tilesize)))
  screen.
blit(‘selected’,(boardx+
(selected[0] * tilesize),
boardy + (selected[1] *
tilesize)))
```



A board consisting of 12 rows and 8 columns of tiles. Pressing **SPACE** will swap the 2 selected tiles (outlined in white), and in this case, create a match of red tiles vertically.