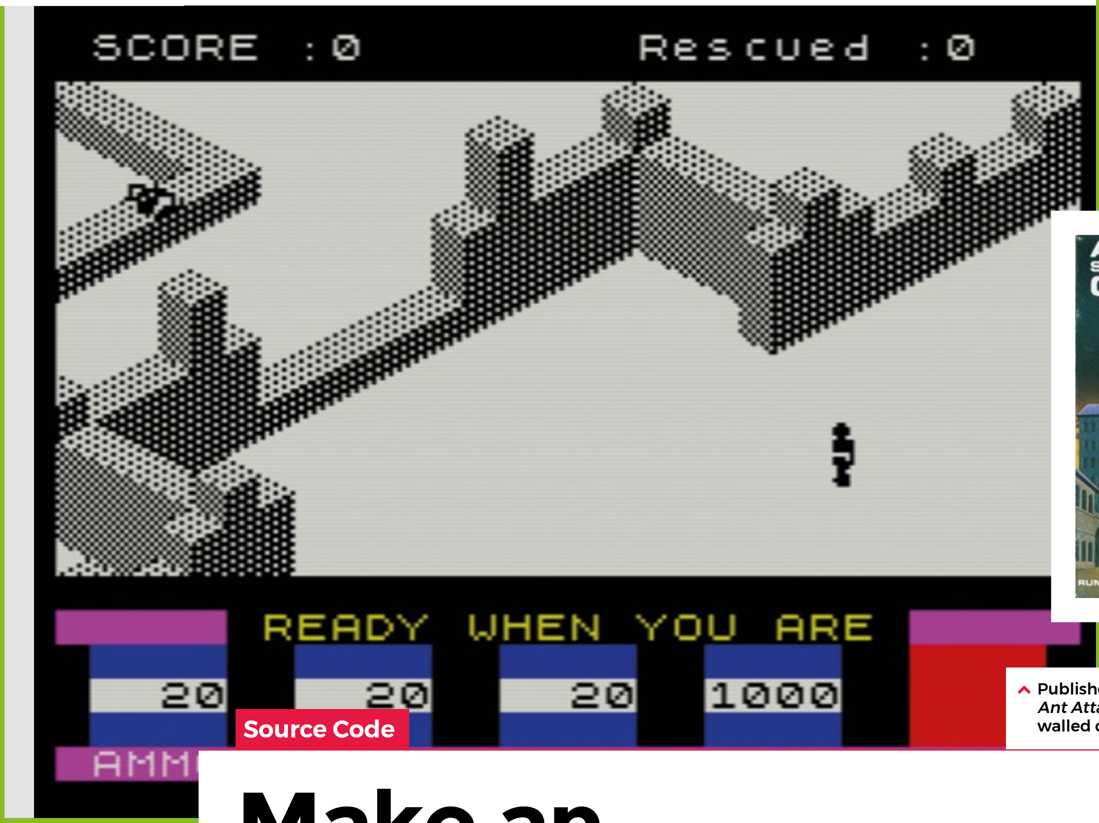


Published by Quicksilva in 1983, *Ant Attack* was set in the fictional walled city of Antescher.

Source Code

Make an isometric game map

AUTHOR
MARK VANSTONE

Here's how to make your own isometric game map worthy of Ant Attack

Most early arcade games were 2D, but in 1982, a new dimension emerged: isometric projection.

The first isometric game to hit arcades was Sega's pseudo-3D shooter, *Zaxxon*. The eye-catching format soon caught on, and other isometric titles followed: *Q*bert* came out the same year, and in 1983 the first isometric game for home computers was published – *Ant Attack*, written by Sandy White.

Ant Attack was first released on the ZX Spectrum, and the aim of the game was for the player to find and rescue a hostage in a city infested with giant ants. The isometric map has since been used by countless titles, including Ultimate Play The Game's classics, *Knight Lore* and *Alien 8*, and my own educational history series, *ArcVenture*.

Let's look at how an isometric display is created, and code a simple example of how this can be done in Pygame Zero – so let's start with the basics. The isometric view displays objects as if you're looking down at 45 degrees onto them, so the top of a cube looks like a diamond shape. The scene is made by drawing cubes on a diagonal grid so that the cubes overlap and create solid-looking structures. Additional layers can be used above them to create the illusion of height.

"To make things easier on the processor, we only need to draw visible cubes"

The cubes are actually two-dimensional bitmaps, which we start printing at the top of the display and move along a diagonal line, drawing cubes as we go. The map is defined by a three-dimensional list (or array). The list is the width of the map by the height of the map, and has as many layers as we want to represent in the upward direction. In our example, we'll represent the floor as the value 0 and a block as value 1. We'll make a border around the map and create some arches and pyramids, but you could use any method you like – such as a map editor – to create the map data.

To make things a bit easier on the processor, we only need to draw cubes that are visible in the window, so we can do a check of the coordinates before we draw each cube. Once we've looped over the x, y, and z axes of the data list, we should have a 3D map displayed. The whole map doesn't fit in the window, and in a full game, the map is likely to be many times the size of the screen. To see more of the map, we can add some keyboard controls.

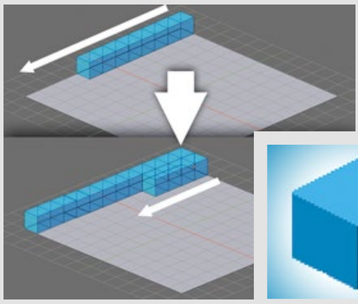
If we detect keyboard presses in the `update()` function, all we need to do to move the map is change the coordinates we start drawing the map from. If we start drawing further to the left, the right-hand side of the map emerges, and if we draw

An Ant Attack-style map in Python

Here's a code snippet that creates an Ant Attack-style map 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/wfmag15**

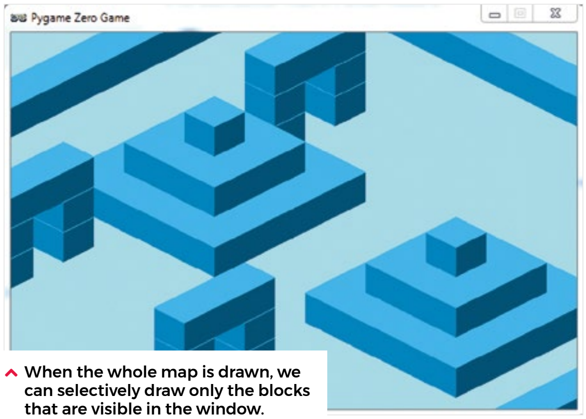
```
import numpy as np
WIDTH = 600 # Width of window
HEIGHT = 400 # Height of window
mapPositionX = 268 # start displaying the map from
mapPositionY = -100 # these window co-ordinates
mapWidth = mapHeight = 20 # the width and height of the map
# make a blank map in a 3 dimensional list
mapBlocks = np.zeros((mapWidth,mapHeight,3))
# draw the window
def draw():
  screen.fill((150, 255, 255))
  drawMap()
# Move the map with the arrow keys
def update():
  global mapPositionX, mapPositionY
  if keyboard.left: mapPositionX -= 4
  if keyboard.right: mapPositionX += 4
  if keyboard.up: mapPositionY -= 4
  if keyboard.down: mapPositionY += 4
# Draw the map to the window
def drawMap():
  for z in range(0, 3):
    for x in range(0, mapWidth):
      for y in range(0, mapHeight):
        bx = (x*32) - (y*32) + mapPositionX
        by = (y*16) + (x*16) - (z*32) + mapPositionY
        # Only display blocks that are in the window
        if -64 <= bx < WIDTH + 32 and -64 <= by < HEIGHT + 32:
            if mapBlocks[x][y][z] == 1:
                  screen.blit("block", (bx, by))
# Make a three layer arch
def makeArch(x,y):
  for z in range(0, 3):
    mapBlocks[x][y][z] = 1
    mapBlocks[x][y+2][z] = 1
  mapBlocks[x][y+1][2] = 1
# Make a three layer pyramid
def makePyramid(x,y):
  for px in range(0, 5):
    for py in range(0,5):
      mapBlocks[px+x][py+y][0] = 1
  for px in range(1, 4):
    for py in range(1,4):
      mapBlocks[px+x][py+y][1] = 1
  mapBlocks[x+2][y+2][2] = 1
# Map building section
for x in range(0, mapWidth):
  for y in range(0, mapHeight):
    if x == 0 or x == mapWidth-1 or y == 0 or y ==
mapHeight-1:
      mapBlocks[x][y][0] = 1
    if x == 5 and (y == 4 or y == 13):
      makeArch(x,y)
    if x == 12 and y == 14:
      makeArch(x,y)
    if (x == 4 or x == 12) and y == 7:
      makePyramid(x,y)
```


Blocks are drawn from the back forward, one line at a time and then one layer on top of another until the whole map is drawn.

The base image we are using is a cube-shaped block on a transparent background, so that it can overlap other cubes when it is drawn.

the map higher, the lower part of the map can be seen.

We now have a basic map made of cubes that we can move around the window. If we want to make this into a game, we can expand the way the data represents the display. We could add different shaped blocks represented by different numbers in the data, and we could include a player block which gets drawn in the `draw()` function and can be moved around the map. We could also have some enemies moving around – and before we know it, we'll have a game a bit like *Ant Attack*.



When the whole map is drawn, we can selectively draw only the blocks that are visible in the window.

TILED

When writing games with large isometric maps, an editor will come in handy. You can write your own but there are several out there that you can use. One very good one is called Tiled and can be downloaded free from **mapeditor.org**. Tiled allows you to define your own tilesets and export the data in various formats, including JSON, which can be easily read into Python.