Player 1

ˇ **An early version of** *Artillery* **was programmed for the Apple II computer in the 1970s.**
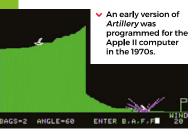
BAGS=2   ANGLE=60   ENTER B,A,F,F■   WIND 20

^ *Artillery Duel* **was an early example of the genre, and appeared on such systems as the Bally Astrocade and Commodore 64 (pictured).**

‹ **Our homage to the artillery game genre. Fire away at your opponent, and hope they don't hit back first.**

Code your own
Artillery-style tank game

Fire artillery shells to blow up the enemy with our take
on a classic two-player artillery game

AUTHOR
MARK VANSTONE

To pick just one artillery game is difficult since it's a genre in its own right. Artillery simulations and games have been around for almost as long as computers, and most commonly see two players take turns to adjust the trajectory of their tank's turret and fire a projectile at their opponent. The earliest versions for microcomputers appeared in the mid-seventies, and the genre continued to develop; increasingly complex scenarios appeared involving historical settings or, as we saw on page six, even mad ideas like battles between factions of worms.

To code the basics of an artillery game, we'll need two tanks with turrets, a landscape, and some code to work out who shot what, in which direction, and where said shot landed. Let's start with the landscape. If we create a landscape in two parts – a backdrop and foreground – we can make the foreground destructible so that when a missile explodes it damages part of the landscape. This is a common

effect used in artillery games, and sometimes makes the gameplay more complicated as the battle progresses. In our example, we have a grass foreground overlaid on a mountain scene. We then need a cannon for each player. In this case, we've used a two-part image, one for the base and one for the turret, which means the latter can be rotated using the up and down keys.

For this code example, we can use the Python dictionary to store several bits of data about the game objects, including the Actor objects. This makes the data handling tidy and is quite similar to the way that JSON is used in JavaScript. We can use this method for the two cannons, the projectile, and an explosion object. As this is a two-player game, we'll alternate between the two guns, allowing the arrow keys to change the angle of the cannon. When the **SPACE** bar is pressed, we call the firing sequence, which places the projectile at the same position as the gun firing it. We then move the missile through the air, reducing the

speed as it goes and allowing the effects of gravity to pull it towards the ground.

We can work out whether the bullet has hit anything with two checks. The first is to do a pixel check with the foreground. If this comes back as not transparent, then it has hit the ground, and we can start an explosion. To create a hole in the foreground, we can write transparent pixels randomly around the point of contact and then set off an explosion animation. If we test for a collision with a gun, we may find that the bullet has hit the other player and after blowing up the tank, the game ends. If the impact only hit the landscape, though, we can switch control over to the other player and let them have a go.

So that's your basic artillery game. But rest assured there are plenty of things to add – for example, wind direction, power of the shot, variable damage depending on proximity, or making the tanks fall into holes left by the explosions. You could even change the guns into little wiggly creatures and make your own homage to *Worms*. ⬡

Artillery-style tank game in Python

Here's Mark's code for an artillery-style tank game. To get it working on your system, you'll need to install Pygame Zero – full instructions are available at **wfmag.cc/pgzero**.

```python
# Artillery
from random import randint
from pygame import Surface, image
from pygame.locals import *
import math

landSurface = Surface((800,600),SRCALPHA)
landSurface.blit(image.load('images/landscape.png') ,(0,0))
gun1 = {"name": "Player 1", "actor": Actor('gunbody1', center=(700,
300)), "turret": Actor('gunbarrel1', center=(695, 280)), "angle": 30,
"multiplier": 1, "color": (255,0,0)}
gun2 = {"name": "Player 2", "actor": Actor('gunbody2', center=(200,
400)), "turret": Actor('gunbarrel2', center=(210, 380)), "angle": 30,
"multiplier": -1, "color": (0,0,255)}
bullet = {"active": False, "actor": Actor('bullet', center=(0, 0)),
"angle": 0, "speed": 0, "count":0}
bang = {"actor": Actor('expl1', center=(0, 0)), "frame": 0}
activePlayer = gun1
gameState = 0

def draw():
    screen.blit('background',(0,0))
    screen.blit(landSurface, (0, 0))
    if bullet["active"] == True: bullet["actor"].draw()
    if gameState != 1:
        gun1["turret"].draw()
        gun1["actor"].draw()
    if gameState != 2:
        gun2["turret"].draw()
        gun2["actor"].draw()
    if gameState == 0: drawText(activePlayer["name"],
activePlayer["color"])
    if gameState == 1: drawText("Player 2 Wins!", (0,255,0))
    if gameState == 2: drawText("Player 1 Wins!", (0,255,0))
    if bang["frame"] != 0: bang["actor"].draw()

def update():
    global activePlayer, gameState
    if gameState == 0:
        if keyboard.space and bullet["active"] == False: fireBullet()
        if keyboard.up: activePlayer["angle"] =
limit(activePlayer["angle"]-1,5,90)
        if keyboard.down: activePlayer["angle"] =
limit(activePlayer["angle"]+1,5,90)
        gun1["turret"].angle = gun1["angle"]
        gun2["turret"].angle = -gun2["angle"]
        if bullet["active"] == True:
            bullet["count"] += 1
            bullet["speed"] = bullet["speed"]*0.991
            bullet["actor"].pos = getNewPos(90-bullet["angle"])
            if checkBullet(bullet["actor"].pos) :
                explosion(bullet["actor"].pos)
                bullet["active"] = False
                if activePlayer == gun1: activePlayer = gun2
                else: activePlayer = gun1
        if bullet["actor"].y > 600: bullet["active"] = False
```

```python
        if bang["frame"] > 0:
            bang["actor"].image = "expl"+str(int(bang["frame"]))
            bang["frame"] += 0.2
            if bang["frame"] > 6: bang["frame"] = 0

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

def fireBullet():
    bullet["active"] = True
    bullet["actor"].pos = activePlayer["turret"].pos
    bullet["angle"] = activePlayer["angle"] * activePlayer["multiplier"]
    bullet["speed"] = 10
    bullet["count"] = 0
    sounds.canon.play()

def getNewPos(angle):
    newX = bullet["actor"].x - (bullet["speed"]*math.cos(math.
radians(angle)))
    newY = bullet["actor"].y - (bullet["speed"]*math.sin(math.
radians(angle)))
    newY += 10-bullet["speed"]
    if bullet["count"] == 60: sounds.whine.play()
    return newX, newY

def checkBullet(pos):
    global gameState
    if pos[0]>0 and pos[0]<800 and pos[1]>0 and pos[1]<600:
        pixel = landSurface.get_at((int(pos[0]),int(pos[1])))
        if pixel[3] > 0: return True
    if gun1["actor"].collidepoint(pos):
        gameState = 1
        explosion(gun1["actor"].pos)
    if gun2["actor"].collidepoint(pos):
        gameState = 2
        explosion(gun2["actor"].pos)
    return False

def explosion(pos):
    x = int(pos[0])
    y = int(pos[1])
    sounds.explosion.play()
    bullet["active"] = False
    bang["actor"].pos = pos
    bang["frame"] = 1
    for c in range(2000):
        landSurface.set_at((x+randint(0,100)-50,y+randint(0,100)-50),
(0,0,0,0))
        if c < 1500: landSurface.set_at((x+randint(0,50)-
25,y+randint(0,40)-20), (0,0,0,0))
        if c < 1000: landSurface.set_at((x+randint(0,20)-
10,y+randint(0,30)-15), (0,0,0,0))

def drawText(t,col):
    screen.draw.text(t, center = (400, 60), owidth=0.5, ocolor=(0,0,0),
color=col , fontsize=40)
```