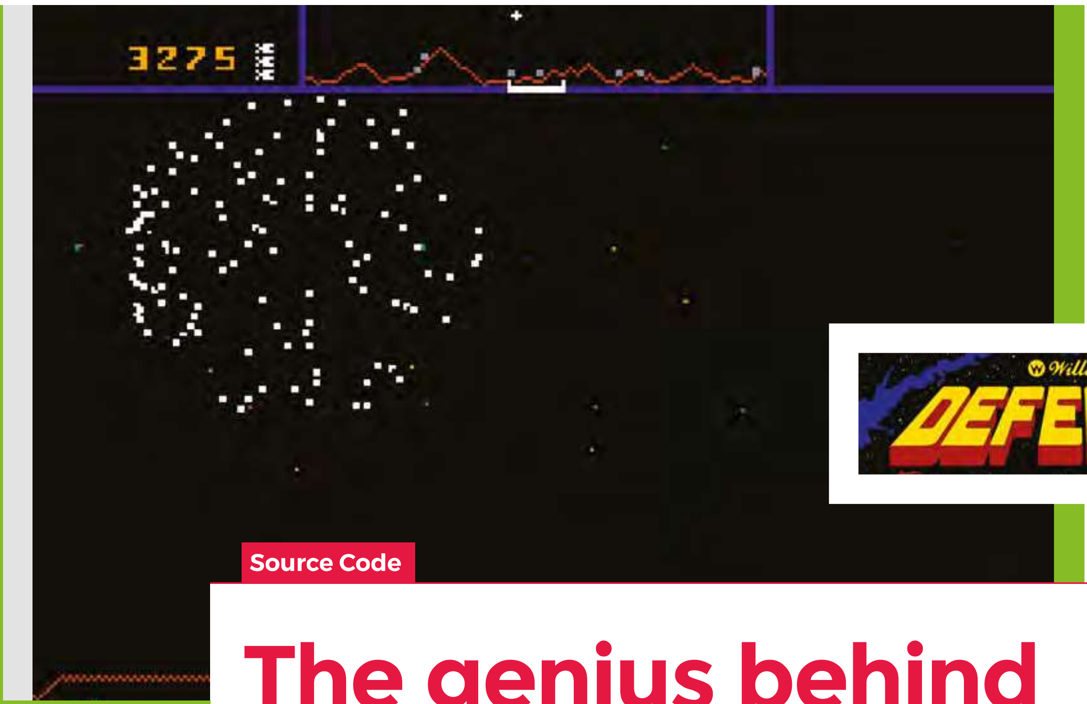
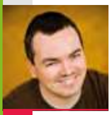


Source Code

The genius behind Defender's explosions



AUTHOR
CRAIG GRANNELL

It was one of the most celebrated games of the 1980s arcade era. We take a closer look at the secrets behind Defender's eye-catching explosions

Defender is mostly remembered for its sadistic difficulty level and dazzling pyrotechnics. In a video game era when obliterating an enemy spacecraft usually 'rewarded' you with a frame or two of animation, *Defender* instead opted for a spectacular fireworks show.

This arrived from a desire that no two games would be alike. With a background in pinball, the *Defender* team enthused about algorithms, not scripting. Also, as co-creator Eugene Jarvis remembers, ideas were "driven by us having no artists – programming was the one thing we could do".

Sam Dicker, a teen at the time, recalls wanting to impress his boss, and this heavily influenced *Defender*'s revolutionary particle effects: "Eugene needed an explosion graphic, but didn't want to overwhelm me. So he said to make something like *Galaxian*'s three-frame fireball animation. But this was my first opportunity to really contribute and show I could do something no one had seen before. So I thought: what if we took a ship's pixels and scattered them across the screen?"

This basic effect was jaw-dropping for the time, but Dicker wasn't done: "When enemies didn't have enough pixels, we'd explode a different piece of art, so we had more pixels to work with. Then I started playing with scattering pixels away from where an object was hit, further affected by how deep the bullet went. After all, the game mostly had you watch things blow up, and so why not make that as varied as possible?"

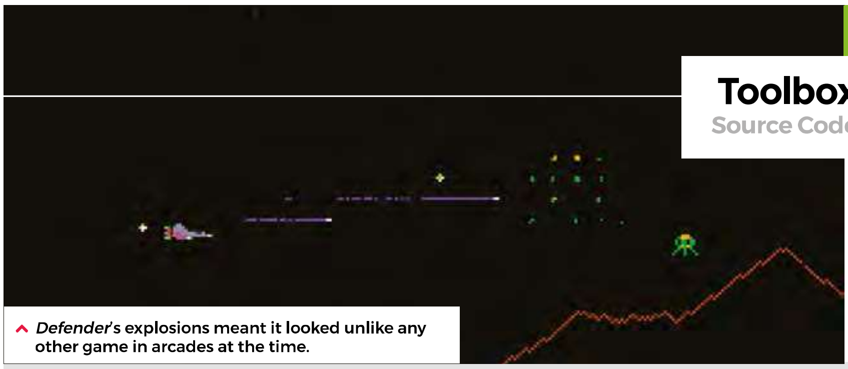
"That's the beauty of *Defender*'s algorithmic world – it became so much more than what we'd planned"

These explosions, notes Dicker, were twinned with a cleverly designed arcade cabinet: "Because we created these things simultaneously, we could use the cabinet like an amp. Through headphones, you just don't get the same effect. Back then, the whole cabinet would vibrate and rattle as you flew about, blowing things up."

For Jarvis, *Defender*'s explosions instantly gave the game an "otherworldly and amazing feel", captivating players who would be "mesmerised by the ballet of destruction". It was a world away from anything else at the time, and provides a useful lesson for modern-day game creators: "Back then, it was all canned animation, and you still see that. It's so boring, because whenever you kill something, you always see the same thing."

VISUAL OVERLOAD

He suggests there's no way an artist could have created pre-canned artwork along these lines; indeed, it would still be impossible. And yet through working with algorithms, this group of non-artists could create something rich and beautiful, offering interaction and feedback to the player. "The explosion choreography was almost like watching the surf," says Jarvis, "and it


Defender's explosions meant it looked unlike any other game in arcades at the time.

DEFENDER 101

Released in 1981, *Defender* was the brainchild of Eugene Jarvis and Larry DeMar. The basic premise is to protect humans on a planet, under threat from being snatched up by aliens. Should a Lander UFO grab and consume a human, it becomes a rabid Mutant, hell-bent on hunting you to destruction. Should all humans be killed, the entire planet explodes, hurling you into a manic survival challenge against waves of ferocious enemies.

If that sounds tricky, it was. *Defender* was a world away from the clockwork simplicity of *Space Invaders* – more a contained chaos, not least when you factor in the controls. The joystick was used only for elevation – buttons were needed to reverse the ship, thrust, fire, set off a smart bomb, or trigger hyperspace.

For dedicated gamers, though, this meant nuance and optimisation, rather than shoe-horning the game into a 'joystick and fire button' setup. So, despite the stern challenge, *Defender* became one of the top-grossing arcade games in history.

really allowed us to leverage our lack of artistry. We used physics as art. It *became* art. And the great thing is, it became greater than what we could have conceived of. But that's the beauty of *Defender*'s algorithmic world – it became so much more than what we'd planned, and much more than what we put into it, because it was a living, breathing thing."

Dicker adds that the effects offered one final twist – when your own ship exploded, you got a massive spherical particle effect that was so eye-popping, it almost removed the frustration at having lost a life. "This was the big one. Because the rest of the animation had stopped at that point, I effectively had a dedicated machine to create that one effect," he says. "So you got to see this massive firework, with dozens of particles."

The very last thing you saw in any game was therefore a dazzling piece of visual overload – and it made you want to go back for more. "Compare that to your death in *Space Invaders*," says Dicker. "That game just kind of stops. It's not very satisfying!"

EXPLOSIONS in PYTHON

Here's a code snippet that shows a *Defender*-style particle explosion working 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/XVIleD**

Download the code from GitHub: **wfmag.cc/wfmag1**

```
import random
import math
WIDTH = 800
HEIGHT = 600  # the size of the screen
DRAG = 0.8 # how much a particle slows down by each second
# the colour of each particle in R, G, B values
PARTICLE_COLOR = 255, 230, 128
MAX_AGE = 3  # the time in seconds for which a particle is displayed
particles = []  # an array to hold the details of the explosion particles
def explode(x, y, speed=300): # Creates a new explosion at co-ordinates
    age = 0     # these are new particles, so set their age to zero
    for _ in range(100):    # generate 100 particles per explosion
        # for each particle, generate a random angle and distance
        angle = random.uniform (0, 2 * math.pi)
        radius = random.uniform(0, 1) ** 0.5
        # convert angle and distance from the explosion point into x and y velocity:
        vx = speed * radius * math.sin(angle)
        vy = speed * radius * math.cos(angle)
        # add the particle's position, velocity and age to the array
        particles.append((x, y, vx, vy, age))
def draw():
# This function redraws the screen by plotting each particle in the array
    screen.clear()  # clear the screen
    for x, y, *_ in particles: # loop through all the particles in the array
        # for each particle in the array, plot its position on the screen:
        screen.surface.set_at((int(x), int(y)), PARTICLE_COLOR)
def update(dt):  # This function updates the array of particles
    new_particles = []  # to update the particle array, create a new empty array
    # loop through the existing particle array:
    for (x, y, vx, vy, age) in particles:
         # if a particle was created more than a certain time ago, it can be removed:
         if age + dt > MAX_AGE:
            continue
        drag = DRAG ** dt  # update particle's velocity - they slow down over time
        vx *= drag
        vy *= drag
        x += vx * dt
        y += vy * dt  # update the particle's position according to its velocity
        age += dt   # update the particle's age
        # add the particle's new position, velocity and age to the new array:
        new_particles.append((x, y, vx, vy, age))
    particles[:] = new_particles  # replace current array with the new one
def explode_random(): # creates an explosion at random location
    x = random.randrange(WIDTH)
    y = random.randrange(HEIGHT)  # select a random position on the screen
    explode(x, y)  # call the explosion function for that position
# call the random explosion function every 1.5 seconds:
clock.schedule_interval(explode_random, 1.5)
```