Programming Aesthetics learned from making independent games

April 1, 2011

Braid

Braid source code

```
http://cloc.sourceforge.net v 1.53
-------------------------------------------------------------------
Language          files        blank        comment        code
-------------------------------------------------------------------
C++                 221        25736           8312       73223
C/C++ Header        239         3655           2029        9467
C                     3         1046           2249        5986
HLSL                 20          354            143        1287
...
-------------------------------------------------------------------
SUM:                488        30907          12746       90347
-------------------------------------------------------------------
```

The Witness source code

```
http://cloc.sourceforge.net v 1.53
-------------------------------------------------------------------
Language              files          blank        comment          code
-------------------------------------------------------------------
C++                     329          37708          14900         96822
C/C++ Header            331           5303           2221         12889
C                         2            583            647          3883
HLSL                     51            986            311          2458
...
-------------------------------------------------------------------
SUM:                    720          44659          18095        116204
-------------------------------------------------------------------
```

"Industry Average": 3250 lines/year

(90000 lines) / (3250 lines/year) ≈ 28 years

512 MB RAM (you can't use it all), no VM
3 *slow* in-order cores
Slow file access

Certification

Program can't crash, even with antagonistic user

Loading time is capped

Certification

“Soak Test”

3 days * 86400 sec/day * 60 frames/sec =
15,552,000 frames

If you leak 4 bytes per frame, you’ll fail.

game design
level design
art direction
audio direction
business development
marketing / PR
financial management

be

extremely effective

at

getting things done

… by the way …

Impulses to optimize

are

usually premature.

Most code is

not

performance-sensitive.

Optimization is

usually bad!

(Unless practiced very carefully!)

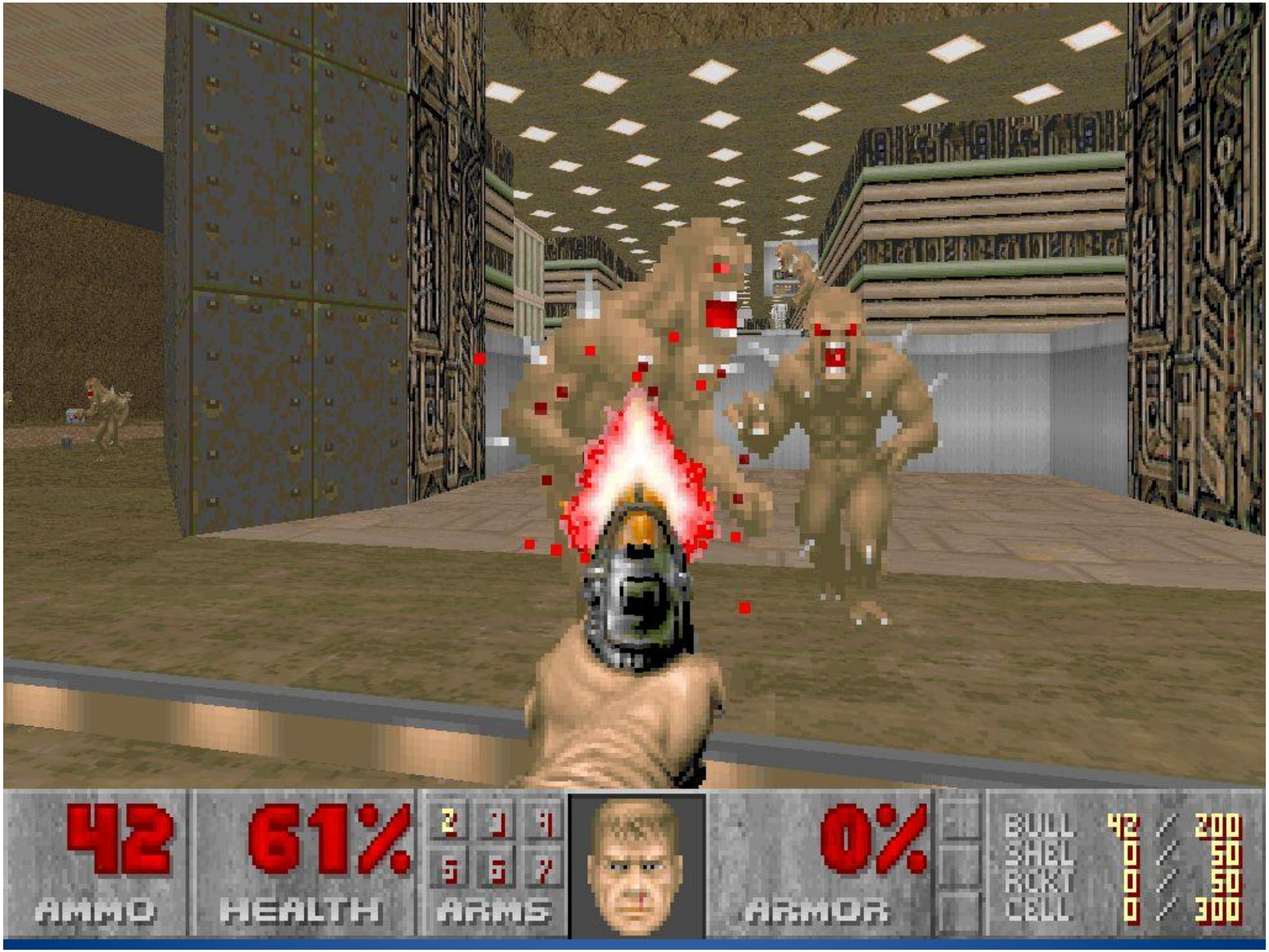
Data Structures

Data structures are about optimization.

"using the right data structure"

is usually bad!

(Because it is premature optimization!)


42
AMMO
61%
HEALTH
2 3 4
5 6 7
ARMS
0%
ARMOR
BULL 42 / 200
SHEL 0 / 50
RCKT 0 / 50
CELL 0 / 300

Now I use arrays of records for almost everything.

Things you might optimize

seconds per program execution (speed)
bytes per program execution (space)

Instead, optimizing

years of my life
per program implementation
(life)

This is a valid optimization parameter you can consider just like those others!

Data structures are about memory or speed optimization.

They are not about life optimization

(unless you absolutely need that speed or memory).

Complicated Algorithms

are not good!

Almost all applied CS research papers are bad

propose adding a lot of complexity
for a very marginal benefit

doesn't work in all cases (limited inputs, robustness)
"supported" by bogus numbers, unfair comparisons

This isn't fooling anyone any more...

A generalized system

is usually *worse*

than a specific / hardcoded one!

Adding new systems
is bad!

This should only ever be
a last resort.

deleting code >>> adding code

Straight-line code preferred over function calls

```
{
    a = b + c;
    a *= f(k);
    a /= g(a);

    print(a);
}
```

```
float
recompute () {
    float a = b + c;
    a *= f(k);
    a /= g(a);

    return a;
}

{
    recompute(a);
    print(a);
}
```

```
{
    {  // Update a.
        a = b + c;
        a *= f(k);
        a /= g(a);
    }

    print(a);
}
```

What is a good programmer, then?

(in this context)

gets things done quickly

gets things done robustly

makes things simple

finishes what he writes (for real)

broad knowledge of advanced
ideas and techniques

(but only uses them when genuinely helpful) *it's easy to see benefits of an idea developed for benefit's sake!*

very hard to measure subtle negatives chained to this idea

(which often outweigh the benefits)

“knowing”

vs.

deeply, intuitively understanding