Coding Tips

Andrew Haydn Grant
Technical Director
MIT Game Lab
September 22, 2014

Iterate Everything

Experiment, analyze, repeat

- Paper prototypes
- Digital prototypes
- Team communication
- Baseball pitches
- Scientific Theories
- Romantic relationships
- Lasagna recipes
- Coding style

Quick Review

All Software Sucks

But we still use it.

All Software Sucks

Including yours.

NOT Writing Code

- Coding Is *Slow*
  - think
  - implement
  - debug
  - integrate
  - debug
  - debug
  - debug

NOT Debugging

- Coding Is *Slow*
  - think
  - implement
  - debug
  - integrate
  - debug
  - debug
  - debug
- Debugging Is ***Slower***

How To Debug Code

- Figure out the problem
- Change the code

Playtest Your Code

"I know that man page is clear! I wrote it."

- Your gameplay is harder than you think it is.
- Your puzzles are harder than you think that are.
- Your instructions aren't as clear as you think they are.
- Your documentation isn't as clear as you think it is.
- Your code is not as comprehensible as you think it is.

It's harder to read code than to write it.

Make It Easier

- Write your code to require as little knowledge as possible.
  - of the project
  - of the function
  - of the computer language
  - of the subject matter

Psychology

Miller's Law:

The number of objects an average human can hold in working memory is 7 ± 2.

Psychology

Decision Fatigue:

The deteriorating quality of decisions made by an individual, after a long session of decision making

Simplicity

- Simplicity means easier to read.
- Simplicity means fewer bugs.
- Simple rarely means “fewer characters”
- Each step is simple. When there are too many steps to hold in your head, clump them.
- Some language functionality is awesome, but still dangerous.
  - In particular, be wary of “write once read never” code.

Write Once, Read Never

Comments

- Write comments to explain WHY we are doing something.
- Write comments to explain WHAT we are doing ***only*** when the briefest glance at the code is not enough.
- Name variables and functions so that they remove the need for comments.
- If an algorithm is long enough that it's easy to lose track of the steps, write an overview of the algorithm in plain language.
- When in doubt, add a comment.

Comments

```
// Is the enemy off the screen?
if (x < 0 || x >= width)
{
    // Yes!
    return null;
}
```

Code

```
if ((x0-x1)*(x0-x1)+(y0-y1)*(y0-y1) < 900)
      return 50;
```

Code

```
if ((x0-x1)*(x0-x1)+(y0-1)*(y0-y1) < 30*30)
      return 50;
```

Code

```
if ((v0-v1).getLength() < 30)
      return 50;
```

Code

```
// Did the player get a bulls eye?
if ((v0-v1).getLength() < 30)
{
     // Yes!  Return the bulls-eye score.
     return 50;
}
```

Code

```
Vector2 offset_from_center = arrow_location - center_of_target;

bool is_bulls_eye = offset_from_center.getLength() < bulls_eye_radius;

if (is_bulls_eye)
{
        return bulls_eye_score;
}
```

Shorter Functions

Make Wrong Code Look Wrong

```
float track_length = angle * radius;

float expected_race_duration = track_length / speed;
```

Make Wrong Code Look Wrong

```
float cm_track_length = angle_in_degrees * radius_in_inches;

float ms_expected_race_duration = km_track_length / mph_speed;
```

Variables, not Constants

Present options to game designer
A constant you can tweak is only useful IF the tweaker can figure out which constant does what
UNITY tips-
- Expose constants in the editor so people can play with them
  - But then make them private if you find a global value that works

Variable Names

- Should be longer than you think
- Should be pronouncable
- Should look different: ClientRecs ClientReps
- Should be spelled correctly
- Should include the units of measurement
- Should not be reused

Function Names

- Should be longer than you think
- Should be pronouncable
- Should look different
- Should be spelled correctly
- Should include the units of measurement

- Are the primary way you teach other programmers about your API

Variable Scope & Names

```
void FeedThePigs(int num_truffles)
{
   this.numHungryPigs -= num_truffles;
   float total_cost = num_truffles * gCostInDollarsPerTruffle;
   total_cost += CalculateOverhead(total_cost);
   gCash -= total_cost;
}
```

Parallel Arrays

```
string[] PlayerNames;
int[] PlayerCash;
Vector2[] PlayerLocation;
```

AVOID.

```
Player[] Players;
```

Order Of Operations

```
float y = 35 * x + (int) --fever / 4.0f + x ^ 2 % snarkle++;
```

I use parenthesis.

Warnings Are Errors

- Warnings are often useful clues about a flaw in your code.
- When you let warnings stick around, you won't notice the **new** one that actually matters.

Backwards Conditionals

```
if (player_spaceship == null)  {}
if (null == player_spaceship) {}
```

Backwards Conditionals

```
if (player_spaceship == null)  {}
if (null == player_spaceship) {}

// Because
if (player_spaceship = null)  {} // valid code (in some languages)
if (null = player_spaceship)  {} // NOT valid code

//Similarly,
if (3 = num_players) {}
if (“ferdinand” = player_name) {}
```

Split Up The Math

```
float foo = (x^2 - sqrt( visibility_threshhold - invisibility_factor / ECM_tech ));
```

vs

```
float adjusted_invisibility = invisibility_factor / ECM_tech;
float visibility = visibility_threshhold - adjusted_invisibility;
float foo = x^2 - sqrt(visibility);
```

Split Up The Math

```
x = foo + bar--;
```

vs

```
x = foo + bar;
bar--;
```

Booleans

A boolean name should ask a question.
That question should be unambiguously answered by TRUE or FALSE.

Booleans
- IsHungry
- HasFood
- WasEaten

Done (yay)
Status (boo)
IsSquare >> Square

Mysterious Constants

Never use string or numeric constants when you can use a variable

```
// Bad
   if (status == “closed”)
     OR
   if (status == 5)


//  Good
  if (status == Status.Closed)
```

foo ? bar : baz

goto

- goto isn't ALWAYS evil.
  - Just most of the time.
- Sometimes, it can make code MORE readable.
  - A convoluted pile of nested IF statements can be really hard to unravel.

Proximity

Keep related actions together:

- Allows reader to mentally clump code.
- Reduces frequency of stealth code
- Declare a variable right before you use it, not 30 lines earlier.

```
Prepare(x);
Prepare(y);
Calculate(x);
Calculate(y);
Print(x);
Print(y);
```

Proximity

Keep related actions together:

- Allows reader to mentally clump code.
- Reduces frequency of stealth code
- Declare a variable right before you use it, not 30 lines earlier.

```
Prepare(x);
Calculate(x);
Print(x);

Prepare(y);
Calculate(y);
Print(y);
```

KISS

Keep It Simple, Stupid.
Keep It Stupidly Simple.

- Do it the easy way first
- Then do it the cool/elegant/efficient way WHEN THE EASY WAY FAILS.
  - And run it by someone else on your team first.

Systems

Resist the temptation to build a system on day 1.

Recursion

- Recursion is really fun
- Recursion is really hard to debug
  - Never recurse when iterating will work just as well.
  - Never have two functions recursively calling each other.
    - It might be “elegant,” but you will tear your hair out getting it to work.
    - And you will forever be terrified of changing the code (rightly so)

Optimizing

Wait.

Only optimize your code if it is actually, observably slow.

Address Bugs ASAP

- Fix bugs before writing new code
  - It will only get harder to fix them
  - You might build on top of the bugs
  - You cannot estimate bug fixes
  - You are always ready to ship
- Treat warnings as errors

Won't Fix

Fixing bugs is only important when the value of having the bug fixed exceeds the cost of the fixing it.

-Joel Spolsky

Debugging

- Use a debugger
  - Learn it. Love it.
- Talk to a team mate.
  - They probably will have no idea what you are talking about.
    - That doesn't matter.
- Take a walk.
- Binary search
  - Perform a test that will cut out as many possibilities as you can.
- If the bug magically vanishes, you are not done.
  - But now you have a clue.

Source Control

- Check EVERYTHING into source control
  - Especially 3rd party libs
- Anyone should be able to do a get and build
  - The build should be one human step.
- Check in often

Daily Builds

- The checked-in build must never remain broken.
  - Fix it immediately.
- Do Check Builds
  - Have a second, clean checkout of the code on your machine
  - When you check in, immediately check out and build there!
- Daily builds force a code test
- Daily deliverables force a project status check

Coding Standards

Why?

- We like tidy code
- It is an objective truth that opening braces belong on the same line as the code

Coding Standards

Why?

- ~~We like tidy code~~
- ~~It is an objective truth that opening braces belong on the same line as the code~~
- Uniform code can reduce the sense of “MY code”
- Rules reduce decision fatigue
- Encourages code that is easier to read
- Encourages code that is easier to debug

Sources

Code Complete by Steve McConnell

Joel is easy to read in blog-sized chunks. He talks as much about management and startups as about code. His writing style is fun and engaging.

Steve has gigantic books that look really daunting. I've never read one from cover to cover. However, that's because he has a lot to say and he says it well.

http://www.joelonsoftware.com/

http://www.stevemcconnell.com/books.htm

Coding Tips

Andrew Haydn Grant
Technical Director
MIT Game Lab
October 2, 2013

Team Coding

Pair Programming

- Two programmers, one computer
  - The **driver** writes code
  - The **observer** or **navigator** reviews each line of code in real time**.**
  - Switch roles frequently!
- The driver considers tactical issues, focusing on the current task.
- The navigator considers larger issues, acting as a safety net.

???

Art
- Placeholder art! Awesome.
- Timeboxing
- Photoshop pngs: save for web!!
- Sprite strips

Audio
- Can kill your download size.
- Compress!
- Mono!
  - mp3 is a pain, but ogg isn't supported everywhere

???

Slide from BBN presentation- these things are hard
(in the slow sense)
Synchronizing music to user inputs

Late Changes

Trespasser Sorting

The Iceberg Secret

- Very little of your software is visible to the player
- People judge software by how it looks
- So make sure that it looks about as done as it is

  ed;hjsdf;kj

Beware Algorithms

Especially ones you create yourself.

Wait, what?

Cool ways of solving problems are one of the reasons that we all learned to program. An Algorithm, in the textbook sense, is a particularly clever way of solving a problem. We want to make some of our own.

Okay, but only make one when you need to.

This goes back to performance. Most algorithms are about solving some problem QUICKLY. The first step you should take is solving the problem in the simplest, most readable, least bug-prone way possible.

Then run it. Computers are fast. This may be good enough.

After the simple thing fails, then think about optimizations. After you've done the easy ones, then consider creating an algorithm. No, wait, don't! Look online first. It's be faster and less bug prone to benefit from some poor shmuck who has solved (and debugged) this problem before.

Intelligent Design

Evolved code vs designed code.

It's stunning, really, how much the code of a long-running prject resembles DNA. There are huge swathes of it that are useless, or appear to be. But when you take them out, the program stops working.

Rewriting is dangerous, but only when you are rewriting old, evolved code.

???

- Keep functions short.
- Declare your variables as close as possible to the place where you will use them.
- Don't use macros to create your own personal programming language.
- Don't use goto.
- Don't put closing braces more than one screen away from the matching opening brace.

???

But also resist the temptation to cut&paste code everywhere. It's already a good idea to split your code up into functions just for readablilty.

It's also a good idea to put common code in functions to save debugging time.

So make it a function call instead of a cut&paste, but wait for AT LEAST the second instance of the code. I like to wait for the third instance, but at that point I'm risking forgetting one of the first two.

CMS.611J / 6.073 Creating Video Games
Fall 2014