IMGD 1001 - The Game Development Process: Debugging

by

Robert W. Lindeman (gogo@wpi.edu)
Kent Quirk (kent_quirk@cognitoy.com)
(with lots of input from Mark Claypool!)

Outline

- Debugging
  - Five-step debugging process
  - Prevention
  - Game Maker specifics
  - Debugging tips
Debugging Introduction (1 of 2)

- Debugging is methodical process for removing mistakes in a program
- So important, whole set of tools to help. Called "debuggers"
  - Trace code, print values, profile
  - Integrated Development Environments (IDEs) (such as Game Maker) have one built in
- A good debugger is really useful...

Debugging Introduction (2 of 2)

- But debugging still frustrating
  - Beginners don't know how to proceed
  - Even advanced can get "stuck"
- Don't know how long it takes to find
  - Variance can be high
  - But can treat them in aggregate for predictions
- What are some tips?
- What methods can be applied?
The Scientific Method

- Evaluation
- Conjecture
- Deduction
- Test

- Lather, rinse, repeat

- Let’s do one

The Problem: Bubble Sort

- We need a routine to sort a list

- Algorithm:
  - Compare adjacent entries in the list
  - If they’re out of order, swap them
  - Move on to the next pair
  - Repeat until the list is sorted

- Yes, this is vague
  - But you might be lucky to get this much description of an algorithm in your code
Step 1: Reproduce the Problem Consistently

- Find case where always occurs
  - Things like: "Sometimes game crashes after kill boss" don’t help much

- Identify steps to get to bug
  - Ex: start single player, room 2, jump to top platform, attack left, ...
  - Produces systematic way to reproduce

- Consider record/playback
  - Console developers use videotape!
Step 2: Collect Clues

- Collect clues as to bug
  - Clues suggest where problem might be
  - Ex: if crash using projectile, what about that code that handles projectile creation and shooting?

- And beware that some clues are false
  - Ex: if bug follows explosion, may think they are related, but may be from something else

- Don't spend too long - get in and observe
  - Ex: crash when shooting arrow. See reference pointer from arrow to unit that shot arrow should get experience points, but it is NULL
  - That's the bug, but why is it NULL?

Step 3: Pinpoint Error

1) Propose a hypothesis and prove or disprove
   - Ex: suppose arrow pointer corrupted during flight.
   - Add code to print out values of arrow in air.
   - But equals same value that crashes.
   - -> Hypothesis is wrong. But now have new clue!
   - Ex: suppose unit deleted before experience points added. Print out values of all units before fire and after all deleted.
   - -> Yep, that's it!

2) Binary-search method (note, can use in conjunction with hypothesis test above, too)
   - Sherlock Holmes: "when you have eliminated the impossible, whatever remains, however improbably, must be the truth."
   - Setting breakpoints, look at all values, until discover bug
   - The "divide" part means break it into smaller sections
     - Ex: if crash, put breakpoint ½ way. Is it before or after? Repeat.
   - Look for anomalies, NULL or NaN values
Step 4: Repair the Problem

- Propose solution. Exact solution depends upon stage of problem.
  - Ex: late in code cannot change data structures. Too many other parts use it!
  - Worry about "ripple" effects.

- Ideally, want original coder to fix.
  - If not possible, at least try to talk with original coder for insights.

- Consider other similar cases, even if not yet reported
  - Ex: other projectiles may cause same problem as arrows did

Step 5: Test Solution

- Obvious, but can be overlooked if programmer is "sure" they have fix
  - Programmer can be wrong!

- So, test that solution repairs bug
  - Best by independent tester

- Test if other bugs introduced
  - Beware "ripple" effect
Debugging Prevention

- Use consistent style, variable names
- Indent code, use comments
- Always initialize variables when declared
- Avoid hard-coded constants
  - They make code brittle
- Add infrastructure, tools to assist
  - Alter game variables on fly (speed up)
  - Visual diagnostics (maybe on avatars)
  - Log data (events, units, code, time stamps)
- Avoid identical code
  - Harder to fix if bug found
  - Use a script
- Verify coverage (test all code) when testing

Game Maker: Print Messages

- Display a Message
  - object → main2 → info

- Or, in code
  - show_message(‘Executed this code’)
  - show_message(‘num:’ + string(number_here))

- Beware if done every step!
  - Save code ahead of time
Game Maker: Debug Mode

- **Show Debug Messages**: 
  - Like `show_message` but in debug mode only
  - Note, doesn’t pause
  - In code: `show_debug_message` (‘Execute this code’) 
  - Need to run in debug mode
  - Debug Information → Tools → Show Messages
Game Maker: Log Messages

- Write messages to file
- Example:
  - At beginning (maybe create log object)
    - `global.log_name = "logfile";
    - `global.fid = file_text_open_write(global.log_name);
  - Then, where needed:
    - `file_text_write_string(global.fid,"Debug message here") ;`
  - Close when done (object → event other → game end):
    - `file_text_close(global.fid)`
- More file operations at:
  - Note: files also useful for save/load game, etc.

Game Maker: Script/Code Syntax

```
{ n = 1;
while (x < 10) {
  x = x + 1;
  n = 0;
}
}
```

Lindeman & Quirk (& Claypool) - WPI Dept. of Computer Science
Game Maker: Error Messages (1 of 2)

Pay attention!

Refers to:
- Object
- Event
- Line number
- Variable name

- Help pinpoint problem
  - Refer to object and method and offending code

Game Maker: Error Messages (2 of 2)

- Can write messages to log file
- Can ignore messages
  - Use "error_last" and "error_occurred" for custom handling
  - Typically, use only in release

Lindeman & Quirk (& Claypool) - WPI Dept. of Computer Science
Debugging Tips (1 of 3)

- **Fix one thing at a time**
  - Don’t try to fix multiple problems
- **Change one thing at a time**
  - Tests hypothesis. Change back if doesn’t fix problem!
- **Start with simpler case that works**
  - Then add more complex code, one thing at a time
- **Question your assumptions**
  - Don’t even assume simple stuff works, or "mature" products
  - Ex: libraries and tutorials can have bugs
- **Minimize interactions**
  - Systems can interfere, make slower so isolate the bug to avoid complications

Debugging Tips (2 of 3)

- **Minimize randomness**
  - Ex: can be caused by random seed or player input. Fix input (script player) so reproducible
- **Break complex calculations into steps**
  - May be equation that is at fault or "cast" badly
- **Check boundary conditions**
  - Classic "off by one" for loops, etc.
- **Use debugger**
  - Breakpoints, memory watches, stack ...
- **Check code recently changed**
  - If bug appears, may be in latest code (not even yours!)
Debugging Tips (3 of 3)

- **Take a break!**
  - Too close, can't see it. Remove to provide fresh prospective

- **Explain bug to someone else**
  - Helps retrace steps, and others provide alternate hypotheses

- **Debug with partner**
  - Provides new techniques
  - Same advantage with code reviews, peer programming

- **Get outside help**
  - Tech support for consoles, Web examples, libraries, ...

Tough Debugging Scenarios and Patterns (1 of 3)

- **Bug in Release but not in Debug**
  - Often in initialized code
  - Or in optimized code
  - Turn on optimizations one-by-one

- **Bug in Hardware but not in Dev Kit**
  - Usually dev kit has extra memory (for tracing, etc.).
  - Suggest memory problem (pointers), stack overflow, not checking memory allocation

- **Bug Disappears when Changing Something Innocuous**
  - Likely timing problem (race condition) or memory problem
  - Even if looks like gone, probably just moved. So keep looking

Based on Chapter 3.5, *Introduction to Game Development*
Tough Debugging Scenarios and Patterns (2 of 3)

☐ Truly Intermittent Problems
   ▪ Maybe best you can do is grab all data values (and stack, etc) and look at ("Send Error Report")

☐ Unexplainable Behavior
   ▪ Ex: values change without touching. Usually memory problem. Could be from supporting system. Retry, rebuild, reboot, re-install.

Based on Chapter 3.5, Introduction to Game Development

Lindeman & Quirk (& Claypool) - WPI Dept. of Computer Science

---

Tough Debugging Scenarios and Patterns (3 of 3)

☑ Bug in Someone Else’s Code
   ▪ 'No it is not." Be persistent with own code first.
   ▪ Find concrete support for your claim!
      ▪ Small reproduction case
   ▪ It's not in hardware
      ▪ Ok, very, very rarely, but expect it not to be, unless you are designing the hardware too!
      ▪ Download latest firmware, drivers
   ▪ If really is, best bet is to help isolate to speed others in fixing it
   ▪ Meanwhile, you probably need to find a workaround or alternative