Interactive Media and Game Development

Debugging

Debugging Introduction

- Debugging is methodical process for removing mistakes in program
- So important, whole set of tools to help. Called “debuggers”
- Trace code, print values, profile
- New Integrated Development Environments (IDEs) (such as Game Maker) have it built in
- But debugging still frustrating
  - Beginners not know how to proceed
  - Even advanced can get “stuck”
  - Don’t know how long takes to find
  - Variance can be high
- What are some tips? What method can be applied?

Outline

- 5-step debugging process
- Game Maker specifics
- Debugging tips

Step 1: Reproduce the Problem Consistently

- Find case where always occurs
  - “Sometimes game crashes after kill boss” doesn’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

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: 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 in camp before fire and all deleted. Yep, that’s it.

And/ Or, 2) divide-and-conquer method
   - 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 1/4 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.
  - 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 (but programmer can be wrong!)
- So, test that solution repairs bug
  - Best by independent tester
- Test if other bugs introduced (beware "ripple" effect)

Debugging Prevention

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

Outline

- 5-step debugging process (done)
- Kodu specifics (next)
- Debugging tips

Debugging in Kodu (1 of 4)

- Built-in debugging aids
  - Path
  - Collision
  - Sight and sound

Debugging in Kodu (2 of 4)
Debugging in Kodu (3 of 4)

• To help see when/if events triggered, add own debug “messages”
  – Color, express …

Debugging Tips (1 of 3)

• One thing at a time
  - Fix one thing at a time – don’t try to fix multiple problems
  - Change one thing at a time – test 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

Debugging Tips (2 of 3)

• Check code recently changed – if bug appears, may be in latest code (not even yours!)
• Use debugger - breakpoints, memory watches, stack …
• 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.
• Minimize randomness –
  - Ex: can be caused by random seed or player input. Fix input (script player) so reproducible

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, …