

Quality Assurance and the Programmer

Eric Raue

About

In this talk I present some ideas of how to improve the interaction between quality assurance (QA) and programmers.

QA

- Finds your mistakes -- you have to fix them.
- Will often write up difficult to understand bug reports.
- Usually cannot provide enough information.
- You sometimes need them to reproduce bugs.
- Basically slow you down.

So ...

- You're smart and should be able to think of some way to help them out.
- Your goals are
 - make them independent
 - increase their quality of bug reports
 - allow them to find more bugs
 - make it easy for you to figure out where the bug is in the code

Some Ideas

- Debug menu
- Automated testing
- Network testing

Debug Menu

Debug Menu (1)

- Idea: Provide a debug menu in the game that allows useful functionality for QA.
- Remember that QA are usually not programmers. Make it simple but powerful.

Debug Menu (2)

- Provide cheats
 - Why: Often verification testing is tedious because getting to the place to test involves a great amount of skill or time.
 - Some good cheats:
 - Unlock everything
 - Instant win
 - Invincible
 - Level skip
 - Show specific screen

Debug Menu (3)

- Screenshots
 - Why: A picture of a bug is a lot easier to figure out than a medium to long non-technical description of it.
 - Tip: Save it over the network to avoid security risks of allowing QA to use memory sticks.
 - Some development kits have this feature already, but most test kits do not.

Debug Menu (4)

- Free camera
 - Why: Allow QA to look for bugs from usually inaccessible camera angles.
 - Can get a closer shot of a bug.

Debug Menu (5)

- Captions
 - Why: Often replay camera angles and cut scenes contain bugs but it is difficult for QA to name them.
 - Also a good idea to log these names.

Debug Menu (6)

- Logging
 - Crashes
 - Why: Instead of just displaying crash dump to screen, save a crash log if possible.
 - Try to include useful information such as PC, RA and other registers.
 - User input
 - The developer could replay the user input to reproduce the bug.

Debug Menu (7)

- Develop a good framework once and use it for all games.
- Build two versions of the game: debug, release
 - Bugs can occur on one but not the other.
 - The debug system will eat up memory.
- Testing for at least the last month before release should be done mostly without the debug menu.

Automated Testing

Automated Testing (1)

- Idea: Save many man hours for verification and coverage testing.
- Remember
 - The only bugs it can find are crash bugs and if the script it is running fails.
 - Not good at finding bugs along the way.

Automated Testing (2)

- Use a scripting language for automation.
- It should be able to
 - Simulate user input
 - Get the current game/screen state
 - Bonus: Primitive AI
 - Take screenshot

Automated Testing (3)

- Use: Verification testing
 - Take screenshots of every <object> in the game. A QA tester can then verify each screenshot without any delay.
 - Depending on the game this is not always possible.

Automated Testing (4)

- Use: Coverage Testing
 - Try every combination of game settings, teams or items.
 - Doing important tasks (e.g a match) many times builds confidence in shipping the game.

Network Testing

Network Testing (1)

- Idea: Simulate network conditions to test the game under real world conditions.

Network Testing (2)

- You can buy hardware for this which is expensive.
- Or you could use netem running on a linux box.
- Can simulate hundreds of users online using server tools.

Network Testing (3)

- Tests
 - The game should be reasonably playable under $700\text{ms} \pm 100\text{ms}$ latency with 10% packet loss.
 - Should be smooth at $350\text{ms} \pm 75\text{ms}$ latency with 10% packet loss.
- The Internet is actually more brutal it seems.

Conclusion

- When working on a game project keep in mind that quality assurance can slow you down unless you make them autonomous by providing them with powerful and simple tools.

Questions?