Tutorial 2: Debugging

Anton Gerdelan <gerdela@scss.tcd.ie>

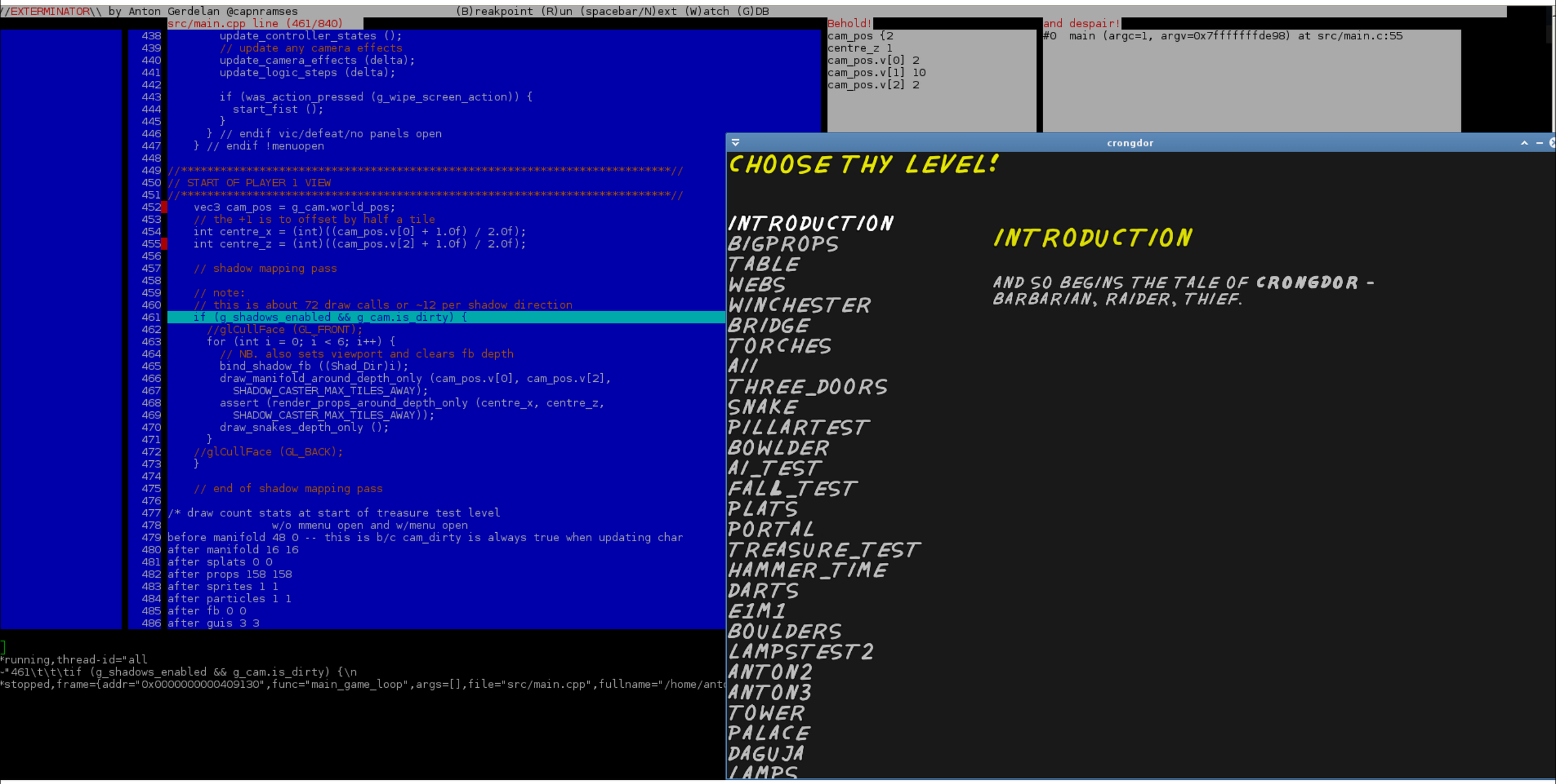
Debugging Tools

- Pencil and paper (diagrams of …., manual step-through)
- printf(), assert()
- Text editor, terminal
- Eyes, brain
- Debugger program - gdb, lldb, the one in visual studio
- A good front-end is nice

The Problem with Linux

- Debuggers are convenient for following the path code takes - functions, recursion, templates…
- Linux has a major dearth of visual debuggers
- gdb and lldb have a command line interface (okay for getting a **backtrace** after crash … clunky otherwise)
- DDD, Eclipse, Code::Blocks, CGDB - none great

“Alright I’ll Just Make My Own Then!”



“oh…*that’s* why there aren’t any decent ones…”

Enter: VS Code

- Visual Studio Code is very nice (simple, quick, visual)
- type in "**code**" in terminal to open
- install+enable the "**C/C++ extension**" (puzzle piece icon)
- you get a text editor
- **breakpoints, watch-list, stack trace,**
  git integration,
- you don't have advanced views; **memory inspect**, **asm**,
  etc.

Setup

- *File->"Open Folder"* gives you a 'project' view
- Open or create a new main.c file
- first let us compile the program (you can use the built-in terminal or an external one)
- `gcc -o my_demo main.c -g -std=c99` (Linux)
- `clang -o my_demo main.c -g -std=c99` (OS X or Linux)
- `mingw32-gcc my_demo main.c -g -std=c99` (Windows with mingw gcc)
- **-g** means "include debugging symbols in my program"
- you won't be able to debug without them

Start Debugging

- View->Debug
- Click in the margin (left of line numbers) to set a breakpoint (red circle)
- the *cog* button takes you to settings file .vscode/launch.json
    - which debugger to use, **enter your program name** to run, etc.

Stepping

- *Green arrow* starts program in debug mode
- Program pauses on breakpoint line
- Use **step over** (next line)
  step in (next line but also go *into* functions)
  step out (run until we leave the function)
  continue (green arrow)
  stop (red square)
- local variables' values (in scope)
- hover mouse over a local variable
- right click variable to **add to watch list**
- **call stack** of functions open with line that call the next function

Useful For

- Checking why variables or output do not have the value you expected - “what am I missing?”
- Check the flow of execution (esp. other peoples spaghetti code)
- Why lead up to it crashing?
    - “binary search” your breakpoints around

The command-line version

- After compiling with debug symbols
- gdb ./my_demo (or lldb ./my_demo for OS X)
- This enters a gdb terminal session
- type **run**
- after it crashes type **bt** to get a backtrace
    - which line caused the segmentation fault and what leads up to that
- type **q** to quit gdb

Alternatives

- On Apple install Xcode in Appstore to get all the programming tools (clang, lldb, etc.)
- Xcode and Visual Studio are pretty handy (but very bloaty with all the project settings)
- I still prefer gcc on Windows (but that's me)
- Some people like Code::Blocks (in lab) and QT Creator on Linux

Other Useful Tools

- **Profilers** {gprof (linux), Instruments (Xcode), VTune}

- **Static Analysis** - very handy code mistake finder - try Clang's static analysis tool on your code files.

- **Memory** leak checker - Valgrind

- **Friends** - code review / tips / sanity check