Programming Proverbs

- 13 Do not recompute constants within a loop
- 14 Avoid implementation-dependent features
- Henry F. Ledgard, ‘‘Programming Proverbs: Principles of Good Programming with Numerous Examples to Improve Programming Style and Proficiency’’, (Hayden Computer Programming Series), Hayden Book Company, 1st edition, ISBN-13: 978-0810455221, December 1975.

Know your tools

- “a bad workman blames his tools”,

  Cambridge Idioms Dictionary

- we will examine:
  - `emacs`, `etags`, `grep`, `diff`, `patch`, `gcc`, `gm2`, `cvs`, `gdb`, `git`

emacs customisation ($HOME/.emacs) for ioquake

```
;; F5 loads in the .gdbinit file ready for debugging ioquake
(defun my-find-file-debug ()
  "load a file"
  (interactive)
  (find-file (concat (getenv "HOME") "/Sandpit/ioquake-latest/ioquake3/.gdbinit")))
(global-set-key [f5] ’my-find-file-debug)

;; F12 compiles ioquake
(setq compile-command "make")
(defun my-compile ()
  "run the compile command after moving to the correct directory"
  (interactive)
  (find-file (concat (getenv "HOME") "/Sandpit/ioquake-latest/ioquake3/.gdbinit"))
  (compile compile-command))

;; compile ioquake by pressing
(global-set-key [f12] ’my-compile)

;; F8 moves to the next error in the source file.
(global-set-key [f8] ’next-error)
```

emacs

- in conclusion you *need* to use an editor which is capable of running `gdb`, `etags` and `grep`.
- there are other candidates `codeblocks`, `kdevelop` and `elipse`.
- however `emacs` is the most mature and also it can be used for many other activities (other than programming).

etags

- the etags program is used to create a tag table file, in a format understood by emacs

- it understands the syntax for the following languages:
    - C, Objective C, C++, Java, Fortran, Ada, Cobol, Erlang, LaTeX, Emacs Lisp/Common Lisp, makefiles, Pascal, Perl, Postscript, Python, Prolog, Scheme and most assembler

Obtaining the modified ioquake tarball

- you will need this to undertake the following tutorial and also the coursework
- open up a terminal and type:

-
  ```
  $ cd $HOME
  $ mkdir -p Sandpit
  $ cd Sandpit
  $ wget http://floppsie.comp.glam.ac.uk/download/c/ioquake-latest.tar.gz
  $ tar zxf ioquake-latest.tar.gz
  ```

Running etags

- open up a terminal and change directory to the ioquake-latest directory

-
  ```
  $ cd $HOME/Sandpit/ioquake-latest
  $ cd ioquake3
  $ etags */*.[ch] */*/*.[ch]
  ```

- the last command requests etag scan each C `.c` and `.h` file in all directories one level below the current

- `etags` builds an index for every type, function, variable declaration
  - it also builds indices for `#define`’s

Using etags

- now start `emacs` and press `<F5>`
- now load in a new file: `^x^f code/server/sv_game.c`
- cursor down to the declaration of the function `SV_GameSystemCalls`
- now continue further and find the call to function `SV_CheckRPC`
  - place the cursor on top of this function
  - now press `<Esc> . <enter>`
  - `emacs` will request you enter the `TAGS` file, you can delete the last path entry. So the path will match the location where you invoked the `etags` program
  - ie the path must match `~/Sandpit/ioquake-latest/ioquake3`
  - now `emacs` will now take you to the declaration of

SV_CheckRPC

Using etags

- now split the screen into two and continue to follow the function calls

- `updatePy` -> `testFor` -> `execFunction`

- now use etags to find the function `putPacket`

grep

- or how to find where functions are called
- read the manual page for `grep`
  - this can be done from within `emacs` via `M-x` `man` `<enter>` `grep` `<enter>`
- return back to the buffer `sv_game.c`
- now type: `M-x` `grep` `<enter>` `putPacket ../*/*.[ch]` `<enter>`
- now you can use the `<F8>` key to move to the next occurrence of `putBuffer`

grep

- the previous grep command was instructed to find all occurrences of `putPacket` in all C `.c` and `.h` files and output these findings in an `emacs` compatible format

diff and patch

- read the manual pages for `diff` and `patch`
- see if you can do this from within `emacs`

diff and patch

- now switch to the `sv_game.c` buffer and split the screen into two
- move to the top of the `sv_game.c` and mark this position `<ctrl><space>`
- now move to the end of the buffer and copy the whole file into the yank buffer `<esc>w`
- move to the other window `^xo` and open up a new file `^x^f sv_game.c-test`

diff and patch

- now yank the internal buffer into this file `^y`
- now make a single line change (maybe just add your name in a comment)
  - anywhere in the file
- save the file `^xs`

diff and patch

- now make `emacs` run the bash shell via `M-x shell <enter>`
- and type:
-
```
$ diff sv_game.c sv_game.c-test
```
- note the output

diff and patch

- `diff`’s are a useful method of transferring textual changes and posting changes to co-developers
- nearly all free software/open source projects operate on this principle
    - this has had a major impact on keeping these free software/open source projects highly portable
- give the flexibility of `diff` and `patch` it is often possible to apply textual `patch`es to later releases of the same project
    - not always, and sometimes a few *hunks* require fixing

diff and patch

- in the early days of free software projects were maintained by creating patches and diffs by hand
- fortunately now there are tools which perform most of this automatically
  - and coordinate all the source files in a project
- these are content management systems such as:
  - `cvs`, `git`, `svn`

diff and patch

- svn or subversion ⟨`http://subversion.tigris.org/`⟩
- git ⟨`http://git-scm.com/`⟩
- cvs ⟨`http://www.nongnu.org/cvs/`⟩
- we will be concentrating on `git` as it is used by the `ioquake3` project

git

- open up a terminal and type in the following:

-
  ```
  $ mkdir -p $HOME/Sandpit/ioquake3-git
  $ cd $HOME/Sandpit/ioquake3-git
  $ git clone git://github.com/ioquake/ioq3.git
  ```

- these commands create a new directory `$HOME/Sandpit/ioquake3-git` and change your current working directory to this location

- finally it checks out the latest copy of the development version of `ioquake3`

git

- now type:

```
$ cd ioquake3
$ make
```

- this builds the latest `ioquake3`
- you should be able to run this by typing:

```
$ ./build/debug-linux-x86_64/ioquake3.x86_64
```

git

- run it, load up the level `pom_bots` and play against alien or legoman
- take note at the smoke and blood when using the grenade launcher and the rocket launcher

git

- now open up another terminal and type:

-
  ```
  $ cd $HOME/Sandpit/ioquake-latest/ioquake3
  $ ./compilequake
  $ ./build/debug-linux-x86_64/ioquake3.x86_64
  ```

- run it, load up the level `pom_bots` and play against alien or legoman

- take note at the smoke and blood when using the grenade launcher and the rocket launcher

git

- open up another terminal and type:

-
  ```
  $ cd $HOME/Sandpit/ioquake-latest/ioquake3
  $ svn diff
  ```

- this will yield the code changes which were made in order that the blood, smoke, gibs and Python bot work

Tutorial work

- see if you can create a patch set from the modified ioquake3 which can be applied to the ioquake3-svn tree

- hint you need to capture the `diff`'s and use patch to apply them
  - however you also need to make copies of some new files which are not in the svn tree

- the new files can be found by typing:

-
```
$ cd $HOME/Sandpit/ioquake-latest/ioquake3
$ svn status
```

Tutorial

- read through the code: `$HOME/Sandpit/ioquake-latest/python-bot/bot-legoman/botfiles/bots/legoman.py` `$HOME/Sandpit/ioquake-latest/python-bot/bot-legoman/botfiles/bots/botlib.py` `$HOME/Sandpit/ioquake-latest/ioquake3/code/botlib/be_ai_py.[ch]` `$HOME/Sandpit/ioquake-latest/ioquake3/code/botlib/be_ai_char.c`
  also read the directed reading at the end of lecture 0

gdb

- GNU debugger
  - online documentation is avilable here ⟨`http://sourceware.org/gdb/current/onlinedocs/gdb`⟩.
- read the
  - summary
  - sample session
  - GDB commands and
  - stopping and continuing

Debugging ioquake3

- you will need two computers, side by side
- on the left computer log in, open a terminal and change directory to ioquake-latest

-
  ```
  $ cd $HOME/Sandpit/ioquake-latest/ioquake3
  ```
- and start the debugging version of ioquake3 under the debugger

Debugging ioquake3

-
  ```
  $ ./debugquake
  ```

- at this point the debugger (GDB) is split into two
  - the server is manipulating ioquake3

- the GDB client will being controlled by you

Debugging ioquake3

- on the right computer login and start emacs
  - maximise it to its rightful size, full screen
  - press F5

Debugging ioquake3

- the first line requires modifying, change `localhost` for the FQDN of the computer running ioquake3 under the debugger

- you should be given a GDB prompt
  - now type

-
  ```
  (gdb) cont
  ```

- and debug the ioquake3 as you would debug an ordinary binary
  - there are a few differences
  - never type `run`
  - and always finish the debugging session by typing `quit`