

John Romero Programming Proverbs

- 2. “It’s incredibly important that your game can always be run by your team. Bulletproof your engine by providing defaults (for input data) upon load failure.”
- John Romero, “The Early Days of Id Software - John Romero @ WeAreDevelopers Conference 2017”

Game Engine Design

- in this module we will examine:
 - tools necessary to develop game engines: `gdb`, `emacs` and friends
- examine how one can integrate Python into a game engine `doom3`
 - exploit parallelism
- examine and extend a small physics game engine and expose its API to Python
- learn how to debug shared libraries and exploit remote debugging
 - should a highly useful transferable skill
 - both `doom3` and `pge` use shared libraries

GNU/Linux lab

- iMacs
 - GNU/Linux treats each as having eight cores
- the lab contains custom software highly tailored for our academic purpose
 - customised doom3
 - pge
 - chisel
 - darkradiant

GNU/Linux lab

- we will be looking at:
 - chisel
 - overview of game engines
 - an understanding of tools (gdb, emacs)
 - doom3
 - bsp and map structure
 - pge and game physics

Chisel

- consists of a number of programs
 - `txt2pen` converts a txt file into a pen file
 - recall the txt file is created in a text editor (emacs, gedit etc)
 - a pen file is the format used by penguin-tower
 - `pen2map` converts a penguin tower map into a doom3 map

chisel/map/doors.txt

```
define 1 room 1
define 2 room 2
define 3 room 3
define s worldspawn
define o monster monster_demon_imp
define n monster monster_demon_hellknight
define i light
define a ammo ammo_shells_large 16
```

```
#####
# 1          . 2          #
#          i  #          i  #
#  s          .          #
#          #          #
#          i  .          #
#          #          i  o  #
#          .          #
#####
```

txt2pen

```
$ cd $HOME/Sandpit/chisel/python  
$ txt2pen -o doors.pen ../maps/doors.txt
```

■ generates a `doors.pen` file from the `../maps/doors.txt` file

doors.pen

```
ROOM 1
  WALL
    1 9   18 9
    18 9   18 1
    18 1   1 1
    1 1   1 9
  DOOR 18 8 18 8 STATUS OPEN LEADS TO 2
  DOOR 18 6 18 6 STATUS OPEN LEADS TO 2
  DOOR 18 4 18 4 STATUS OPEN LEADS TO 2
  DOOR 18 2 18 2 STATUS OPEN LEADS TO 2
  LIGHT AT 15 7
  LIGHT AT 12 4
  SPAWN PLAYER AT 4 6
END
```


doors.pen

```
ROOM 2
  WALL
    18 9   34 9
    34 9   34 1
    34 1   18 1
    18 1   18 9
  DOOR 18 2 18 2 STATUS OPEN LEADS TO 1
  DOOR 18 4 18 4 STATUS OPEN LEADS TO 1
  DOOR 18 6 18 6 STATUS OPEN LEADS TO 1
  DOOR 18 8 18 8 STATUS OPEN LEADS TO 1
  MONSTER monster_demon_imp AT 32 3
  LIGHT AT 26 7
  LIGHT AT 24 3
END

END.
```

Obtaining chisel



```
$ cd  
$ mkdir Sandpit  
$ cd Sandpit  
$ git clone https://github.com/gaiusm/chisel
```



chisel is installed globally on all iMacs, however you have to extend the chisel source code as part of your coursework

Running: your copy of txt2pen

```
$ cd $HOME/Sandpit/chisel/python
$ python3 txt2pen.py -h
Usage: txt2pen [-dhvV] [-o outputfile] inputfile
  -d debugging
  -h help
  -V verbose
  -v version
  -o outputfile name
```

```
$ python3 txt2pen.py -o doors.pen ../maps/doors.txt
```

Operating system concepts!

- we will be looking at networking in a game engine
- also looking at architectural parallelism in doom3

Architectural parallelism in doom3

- within the the doom3 modifications to introduce Python bots
- notice the calls to `fork` and `execl`

Architectural parallelism in doom3

■ `doom3/source/latest-git/dhewm3/neo/game/ai/pybot.cpp:1144`

```
char buffer[PATH_MAX];

idStr::snPrintf (buffer, sizeof (buffer), "%s/%s/%s.py",
                getHome (), getDir (), name);
gameLocal.Printf ("execl /usr/bin/python3 %s\n", buffer);
int pid = fork ();
if (pid == 0)
    /* child process. */
    {
        int r = execl ("/usr/bin/python3", "python3", buffer,
                      (char *)NULL);

        if (r != 0)
            perror ("execl");
    }
}
```

Architectural parallelism in doom3

- we notice that `doom3` and `python3` are running in parallel
 - allowing the bot to run its pathfinding and AI simultaneously as the engine

chisel: txt2pen

- source is in one file:
`$HOME/Sandpit/chisel/python/txt2pen.py`
 - 690 lines of Python
- uses the following command line options

```
$ cd $HOME/Sandpit/chisel/python
$ python3 txt2pen.py -h
  -d debugging
  -h help
  -V verbose
  -v version
  -o outputfile name
```


chisel: txt2pen

- notice the `-o` option which takes an additional argument (filename)
- it uses the `getopt` module to handle the options
 - see function `handleOptions`

chisel: txt2pen

```
def handleOptions ():
    global debugging, verbose, outputName

    outputName = None
    try:
        optlist, l = getopt.getopt(sys.argv[1:], ':dho:vV')
        for opt in optlist:
            if opt[0] == '-d':
                debugging = True
            elif opt[0] == '-h':
                usage (0)
            elif opt[0] == '-o':
                outputName = opt[1]
            elif opt[0] == '-v':
                printf ("txt2pen version " + str (versionNumber) + "\n")
                sys.exit (0)
            elif opt[0] == '-V':
                verbose = True
        if l != []:
            return (l[0], outputName)

    except getopt.GetoptError:
        usage (1)
    return (None, outputName)
```

chisel: txt2pen

- it uses a dictionary to maintain the defines
- stores the map in a 2D list (array)
 - mapGrid
- it determines the walls of a room
 - it finds the room number (location)
 - moves to the top left inside the room (`generateRoom`)
 - it then attempts to turn left as it moves around the room (the wall is always on the left)
 - examine `scanRoom` for the implementation
 - it looks the square forward and square forward left comparing the two characters: `##` or `--` or `#-`
 - `#` wall and `-` for space

Extending chisel

- one of the obvious improvements is for chisel to automatically introduce lights
 - add another option to enable automatic lighting
 - `-l`
- copy `scanRoom` into a new function `introduceLights`
- adapt this new function to add lights
 - but only if the rooms has no user defined lights

Doom3 data directories

- you will need to initialise the doom3 map directory so that darkradiant and doom3 will work
- you can do this by:

```
$ cd  
$ wget http://floppsie.comp.glam.ac.uk/download/targz/skeleton-doom3-data.tar.gz  
$ tar xzf skeleton-doom3-data.tar.gz
```

Doom3 data directories

- this will silently populate your directory
 - `.local/share/dhewm3/base`
 - with symbolic links to the real doom3 data

- your doom3 map file will always be:
 - `.local/share/dhewm3/base/maps/tiny.map`

- you will need to remember these paths and feed them into `darkradiant`
 - `.local/share/dhewm3/base` contains all the doom3 data
 - `.local/share/dhewm3/base/maps/tiny.map` is your map

darkradiant

- change directory into

- ```
$ cd
$ cd Sandpit/chisel/python
$./developer-txt2map ../maps/two.txt
```

- view your map using the tool, remember your output file will always be (tiny.map)
  - when running darkradiant you will need to configure the map directory
  - you can click on the right hand mouse button to fix/enable freelook
  - cursor keys will move you around the 3D space

# darkradiant

- ```
$ darkradiant
```
- now change the map slightly
- ```
$ gedit ../maps/two.txt
$./developer-txt2map ../maps/two.txt
```
- and view the changes using darkradiant