CSCI 362: Final Report

Testing NetHack

TeamName:

Daniel Hwang, David Rust, Kyle Stewart

TeamName:
David Rust
Kyle Stewart
Daniel Hwang

Table of Contents:

Introduction

Our group, aka: TeamName, performed testing on NetHack - which is a singleplayer Rogue-like video game released in 1987, featuring ASCII graphics.

Chapter 1: Process to Compile Nethack for Ubuntu:

Nice resource: https://nethackwiki.com/wiki/Compiling

1. Clone from https://github.com/NetHack/NetHack.git. We'll say you save it in a folder named "Nethack".
2. Install dependencies. In the Terminal, run "sudo apt-get install bison flex libncurses5-dev" then type your password if any.
3. Instructions can be found in Nethack/sys/unix/NewInstall.unx. Gist of it is: there are "hints" files that automate the compilation process for certain systems, and in this case we can use the "linux" hints file.
4. Once you've followed the instructions in the NewInstall.unx file, navigate to your new NETHACKDIR (probably /usr/games/lib/nethackdir) and create a file called "sysconf" with the command "touch sysconf". Now you're compiled and installed! Run "nethack" in the terminal to play.

How did we get to this point?

Nethack is over 30 years old, and has had significant support from the developers. They have made it pretty straightforward for anyone to compile and install the game on numerous machines. By reading the README in the top directory from the github, it seemed like installing on Linux would be as easy as following the instructions in the "$top/sys/unix" folder, but just by looking in that folder, you can see there are actually something like 4 instruction files! After reading through these, the "NewInstall.unx" file was the newest, and seemed to simplify the process as long as our system isn't out-of-the-ordinary. We are using a fresh Ubuntu install, so just using the Linux hints file works fine. I had some trouble with the dependencies, but that was because I did not read all the resources available to me (specifically, the wiki page linked above is very clear about the dependencies). A bit of trial-and-error there had me compiled in short order. Running the game was more difficult, because I couldn't figure out where the "sysconf" file was supposed to go. Once again, that wiki page is very clear about this, so maybe the moral of the story is: read the wiki page on compiling, when that is an option to you! Lastly, there were no included test cases, and after discussion with the developers, their tests are not recorded in easy-to-distribute files.

TeamName:
David Rust
Kyle Stewart
Daniel Hwang

Chapter 2: "NetHack Test Plan" by TeamName

The Testing Process: First, we will focus testing on the Wish Parser and Object Naming functions found in objnam.c. Later tests may include areas such as Level Generation.

Tested Items:
objnam.c:

makesingular(*oldstr*) - Function receives a word as a string and returns the singular version of the word (e.g. "man" to "men," "sword" to "swords").

makeplural(*oldstr*) - Function receives a word as a string and returns the plural version of the word (e.g. "harpy" to "harpies," "dwarf" to "dwarves").

fruitname(*juice*) - Function receives a player-specified fruit name and converts the string into the name of the corresponding fruit juice.

badman(*basestr*, *to-plural*) - Function receives a prefix as a string which would be followed by "man" or "men" and returns a boolean representing whether the corresponding singular or plural counterpart exists.

cxname(*obj*) - Function receives an object struct and returns the object's name as a string.

Testing Schedule:

Deliverable 3 (Nov 9): By this date 5 test cases must be detailed and completed. For keeping with schedule in following Deliverables, 15 completed test cases would be preferred.
Deliverable 4 (Nov 19): By this date 25 test cases must be detailed and completed.
Deliverable 5 (Nov 28): Final report containing complete documentation of all test cases and procedures.

Test Recording Procedures:

All tests will include some kind of output, either to text files (preferred) or printed to the command line. Output will include timestamp records, scenario explanation, log of parameters passed and values returned, and result of the test: Pass, Fail, or another more detailed metric, if called for. Hardware and Software Requirements: Ubuntu 16.04 LTS Distribution Please follow the build instructions (see pg.1)

TeamName:
David Rust
Kyle Stewart
Daniel Hwang

Chapter 5: Fault Injection

In the *objnam.c* file that we are testing adding faults was relatively simple. To do this we simply commented out sections of conditional statements that handled special word cases in the makeplural() and makesingular() functions we are testing.

In our particular case, we commented out the if-statement sections under the following comments in the program:

- algae, larvae, hyphae (another fungus part)
- man/men ("Wiped out all cavemen.")
- ium/ia (mycelia, baluchitheria)
- matzoh/matzot, possible food name
- Ends in z, x, s, ch, sh; add an "es"

After making these modifications we rebuilt NetHack, like so:

```
cd $Top/sys/unix      ($Top - NetHack main directory, the dir that contains the readme)
sh setup.sh hints/linux
cd ../..
make all
```

Add the test cases to exampleTests.txt in the framework folder:

```
makeplural larva      larvae
makeplural hypha      hyphae
makeplural amoeba     amoebae
makeplural man  men
makeplural matzoh matzot
makeplural mycelia  mycelium
makeplural balactheria  balactherium
makeplural mutex  mutexes
```

Run the frame.py script from your framework folder after first running junk.sh:

```
sh junk.sh
python3 frame.py
```

After running the script all of these cases should result in a FAIL result output.

Chapter 6: Conclusions