blood.nw – the Thurber "blood game" as a Perl-CGI

Edward McGuire

March 14, 2024

"There are lips in pistol
And mist in times,
Cats in crystal,
And mice in chimes."

–James Thurber, "Here Come The Tigers"

Abstract

As described by Thurber, the game is to find as many words as you can within a starter word. Described below is a method in software to play the game, with a browser-based implementation.

Contents

1 Preface

Thurber's story characterizes the game as finding as many words as you can within a single word. That is the game that this paper addresses –

cats: acts, cast, scat, act, cat, sac, sat, tas, as, at, st, ta

But he actually describes several related processes in his story. There is also finding a single word in as many larger words as you can –

tiger: aigret, begirt, gaiter, girted, goiter, goitre, grivet, engirt, regilt, tigers, tigery, triage

And there is finding a sentence in a word, perhaps one that matches the word in mood and tone –

chimes: She ices his mice.

2 Method

The set of characters in the starting word is broken down into subsets.

For example, `the` yields these subsets: `(e,h)`, `(e,t)`, `(t,e)`, `(e)`, `(h)`, and `(t)`.

Each set occupies one entry in the table `subwords`.

5a ⟨*blood.pl globals* 5a⟩≡ (9)

```
my %subwords ;
```

The set of letters in the original word is added to the table.

5b ⟨*blood.pl explodeword* 5b⟩≡ (9) 5c▷

```
my $seed = join( '' , sort split( '' , lc( parameter::value($word) ) ) ) ;
$subwords{ $seed } = undef ;
```

Then the set is broken down into subsets.

5c ⟨*blood.pl explodeword* 5b⟩+≡ (9) ◁5b

```
explode( $seed ) ;
```

A subroutine, defined recursively, performs the breakdown.

5d ⟨*blood.pl subroutines* 5d⟩≡ (9) 11a▷

```
sub explode($) ;
sub explode($)
{
        if( length $_[0] > 1 )
        {
                for ( my $cursor = 0 ; $cursor < length $_[0] ; $cursor++ )
                {
                        my $subword =
                                substr( $_[0] , 0 , $cursor )
                                .
                                substr( $_[0] , $cursor + 1 )
                        ;
                        if( not defined $subwords{ $subword } )
                        {
                                $subwords{ $subword } = undef ;
                                explode( $subword ) ;
                        }
                }
        }
}
```

The sets are then filtered by reference to a dictionary. Sets which correspond to a word within the dictionary are marked as valid. Each valid set occupies one entry in the table `valid`.

6a ⟨*blood.pl filter* 6a⟩≡ (9) 6b▷

```
my %valid ;
```

Validation entails reading the dictionary sequentially, transforming each entry the same way that the sets are transformed, and looking up the entry in the table of sets.

6b ⟨*blood.pl filter* 6a⟩+≡ (9) ◁6a

```
#open( DICT , '<', $dictpath{ parameter::value( $dictionary ) } )
open( DICT , $dictpath{ parameter::value( $dictionary ) } )
        or die "blood.pl: no dictionary" ;
while( <DICT> )
{
        chomp ;
        if( /^[A-Z]/ )
        {
                next unless parameter::value( $proper ) eq 'on' ;
        }
        $valid{ lc( $_ ) } = $_
                if exists $subwords{ join( '' , sort split( '' , lc( $_ ) ) ) } ;
}
close( DICT ) ;
```

3 Makefile

This section compiles and installs the application.

7a ⟨*makefile* 7a⟩≡ 7b▷

```
usage :: ; @echo 'usage: make { all | install | commit | clean }'
all ::
install :: all
commit ::
clean ::
```

This code is written to file `makefile`.

7b ⟨*makefile* 7a⟩+≡ ◁7a 7c▷

```
commit :: ; git commit -av -uno
clean :: ; rm -f *~ .*~
```

7c ⟨*makefile* 7a⟩+≡ ◁7b 7d▷

```
blood.nw.sentinel : blood.nw ; noweb -t $< && touch $@
clean :: ; rm -f blood.nw.sentinel
```

7d ⟨*makefile* 7a⟩+≡ ◁7c 7e▷

```
all :: blood.pl
blood.pl : blood.nw.sentinel ;
install :: /var/www/metaed.com/cgi-bin/blood.cgi
/var/www/metaed.com/cgi-bin/blood.cgi : blood.pl ; install $< $@
clean :: ; rm -f blood.pl
```

7e ⟨*makefile* 7a⟩+≡ ◁7d 7f▷

```
all :: makefile
Makefile : blood.nw.sentinel ;
clean :: ; rm -f makefile
```

7f ⟨*makefile* 7a⟩+≡ ◁7e 8a▷

```
all :: blood.pdf
blood.pdf : blood.tex ; latexmk -pdf blood.tex
PDF = /var/www/metaed.com/root/papers/blood.pdf
install :: $(PDF)
$(PDF) : blood.pdf ; cp $< $@
clean ::
        latexmk -c blood.tex
        rm -f blood.pdf
```

8a ⟨*makefile* 7a⟩+≡ ◁7f 8b▷

```
blood.tex : blood.nw ; noweb -o $<
clean :: ; rm -f blood.tex
```

8b ⟨*makefile* 7a⟩+≡ ◁8a

```
all :: blood.html
blood.html : blood.nw
        noweave -html -filter l2h -index -autodefs c $< >blood.tmp
        htmltoc <blood.tmp >$@
clean :: ; rm -f blood.html blood.tmp
```

4 blood.pl

The outline of the program.

9 ⟨*blood.pl* 9⟩≡

```
#! /usr/bin/perl
# This file Compiled from rc.nw using Noweb by Norman Ramsey.
⟨blood.pl preamble 10⟩
⟨blood.pl globals 5a⟩
⟨blood.pl subroutines 5d⟩
⟨blood.pl parameters 12⟩
⟨blood.pl getword 13⟩
⟨blood.pl explodeword 5b⟩
⟨blood.pl findwords 14a⟩
⟨blood.pl filter 6a⟩
⟨blood.pl drawpage 17⟩
exit 0 ;
```

This code is written to file `blood.pl`.

5 Preamble

The preamble establishes the environment in which this program is executed. It checks the interpreter version, and sets interpretation options. It also adds the standard Perl CGI library to the environment.

10 ⟨*blood.pl preamble* 10⟩≡ (9)

```
die 'blood.pl: wrong Perl interpreter ' . $]
        unless $] eq '5.008008'
        or $] eq '5.010000'
        or $] eq '5.034000' ;
use strict ;
use warnings ;
use CGI qw(:standard) ;
```

6 Subroutines

A `parameter` object represents a value determined from the environment. It takes a default value if missing from the environment. The `make` method creates it; the `value` method returns its value.

11a ⟨*blood.pl subroutines* 5d⟩+≡ (9) ◁5d
⟨*blood.pl parameter::make* 11b⟩
⟨*blood.pl parameter::value* 11c⟩

The `make` method creates a `parameter` object. The required arguments are the name and default value of the parameter. It returns the object if successful; `undef` if an error occurs.

11b ⟨*blood.pl parameter::make* 11b⟩≡ (11a)
```
sub parameter::make { return @_ < 2
        ? undef
        : { VALUE => scalar defined param( $_[0] ) ? param( $_[0] ) : $_[1] } ;
}
```

The `value` method expects a `parameter` object and returns its value.

11c ⟨*blood.pl parameter::value* 11c⟩≡ (11a)
```
sub parameter::value { return @_ < 1
        ? undef
        : $_[0]->{"VALUE"} ;
}
```

7 Parameters

Here we establish parameters and their default values.

12 ⟨*blood.pl parameters* 12⟩≡ (9)

```
my $word       = parameter::make( 'word'       , 'chimes'  ) ;
my $limit      = parameter::make( 'limit'      , '50'      ) ;
my $dictionary = parameter::make( 'dictionary' , 'yawl032' ) ;
my $proper     = parameter::make( 'proper'     , 'on'      ) ;
```

8 Getword

Now we determine what word to blood-game. The word comes either from an input parameter or a hard-coded default.

13 ⟨*blood.pl getword* 13⟩≡ (9)

```
if( length parameter::value( $word ) > 10 )
{
        print
                header( -type => 'text/html' )   ,
                start_html                       ,
                h1( 'Error' )                    ,
                p( 'Word is too long.' )         ,
                end_html                         ,
        ;
        exit 0 ;
}
```

9 Findwords

Somewhere in the environment, there need to be word lists (dictionaries). These are used below to filter the exploded wordlist. Here we search for the word lists that are available and make note of them.

14a ⟨*blood.pl findwords* 14a⟩≡ (9) 14b▷

```
my %dictpath ;
my %dictdesc ;
```

Some systems keep a text dictionary at `/usr/share/dict/words`.

14b ⟨*blood.pl findwords* 14a⟩+≡ (9) ◁14a 14c▷

```
my $dictwords = '/usr/share/dict/words' ;
if ( -f $dictwords )
{
        $dictpath{ $dictwords } = $dictwords ;
        $dictdesc{ $dictwords } = $dictwords ;
}
```

When the system has `aspell`, we can pipe from its wordlist.

14c ⟨*blood.pl findwords* 14a⟩+≡ (9) ◁14b 15▷

```
{
        my $dictid = 'aspell -d en_US dump master|' ;
        $dictpath{ $dictid } = $dictid ;
        $dictdesc{ $dictid } = $dictid ;
}
```

If the environment is Debian Linux, we can use the `wordlist` table to identify dictionaries.

15 ⟨*blood.pl findwords* 14a⟩+≡ (9) ◁14c 16a▷

```
my $debian_wordlist_dir = '/var/lib/dictionaries-common/wordlist' ;
my $debian_wordlist_glob = $debian_wordlist_dir . '/*' ;
my $debian_dict = '/usr/share/dict' ;
if( -d $debian_wordlist_dir )
{
        map
        {
                open my $fh, '<', $_ or die $! ;
                my $dictid = ( $_ =~ m'.*/(.*)' )[0] ;
                local $/ ;
                my %dict_meta_data = map
                {
                        split /:\s*/, $_, 2 ;
                } split /\n/, <$fh> ;
                close $fh ;
                if( defined $dict_meta_data{ 'Hash-Name' } )
                {
                        $dictpath{ $dictid } = $debian_dict . '/'
                                . $dict_meta_data{ 'Hash-Name' } ;
                        $dictdesc{ $dictid } = $dict_meta_data{ 'Language' }
                                if -f $dictpath{ $dictid } ;
                }
        } glob( $debian_wordlist_glob ) ;
}
```

The dictionary “Yet Another Word List 0.3.2” might exist somewhere.

16a ⟨*blood.pl findwords* 14a⟩+≡ (9) ◁15 16b▷

```
{
        my $dictid = 'yawl032' ;
        my $yawl032 ;
        #$yawl032 = '/home/meta/newproject/toys/trunk/yawl-0.3.2/word.list' ; # on delaware
        #if( -f $yawl032 )
        #{
        #       $dictpath{ $dictid } = $yawl032 ;
        #       $dictdesc{ $dictid } = 'Yet Another Word List 0.3.2' ;
        #}
        #$yawl032 = '/cygdrive/c/Documents and Settings/duane/newproject/toys/trunk/yawl-0.3.2/word.li
        #if( -f $yawl032 )
        #{
        #       $dictpath{ $dictid } = $yawl032 ;
        #       $dictdesc{ $dictid } = 'Yet Another Word List 0.3.2' ;
        #}
        $yawl032 = '/srv/www/metaed.com/private/word.list' ; # on newjersey
        if( -f $yawl032 )
        {
                $dictpath{ $dictid } = $yawl032 ;
                $dictdesc{ $dictid } = 'Yet Another Word List 0.3.2' ;
        }
}
```

And we might have the Wordle word list.

16b ⟨*blood.pl findwords* 14a⟩+≡ (9) ◁16a

```
{
        my $dictid = 'wordle-2023-07-06' ;
        my $dict = '/home/metaed/src/wordle/wordle.list.2023-07-06' ;
        if ( -f $dict )
        {
                $dictpath{ $dictid } = $dict ;
                $dictdesc{ $dictid } = 'Wordle List 2023-07-06' ;
        }
}
```

10 Drawpage

Here we draw the form with any results on the browser canvas.

17 ⟨*blood.pl drawpage* 17⟩≡ (9)

```
my $wordPrint = escapeHTML( ucfirst( parameter::value($word) ) ) ;
my $title = $wordPrint . ': ' ;
my @sorted = sort { length $b <=> length $a || lc( $a ) cmp lc( $b ) } keys %valid ;
if( @sorted )
{
        $title .= $valid{ shift @sorted } ;
        my $n = 1 ;
        foreach( @sorted )
        {
                last if
                        parameter::value( $limit ) > 0
                        and
                        $n >= parameter::value( $limit )
                ;
                $n++ ;
                $title .= ', ' . $valid{ $_ } ;
        }
        #$title .= '.' ;
}
#$title .= ' ' . $wordPrint . '.' ;
print
        header( -type => 'text/html' )
                ,
        start_html(
                -dtd => '-//W3C//DTD HTML 4.01//EN'
                        ,
                -encoding => 'utf8'
                        ,
                -title => $title
                        ,
#               -head => Link( { -rev => 'made' , -href => uri_escape( 'mailto:blood@metaed.com' ) } )
                -head => Link( { -rev => 'made' , -href => 'mailto:blood%40metaed.com' } )
        )
                ,
        blockquote(
                'There are lips in pistol'
                . '<br>And mist in times,'
                . '<br>Cats in crystal,'
                . '<br>And mice in chimes.'
```

```
                . '<br>--James Thurber, "Here Come The Tigers"'
        )
                ,
        h1( $title )
                ,
#       p( 'Find the meaning of a name or any word within itself. ' )
#               ,
#       p( keys %subwords )
#               ,
#       p( keys %valid )
#               ,
#       p( @sorted )
#               ,
#       p( keys %dictname )
#               ,
#       p( values %dictname )
#               ,
#       p( keys %dictpath )
#               ,
#       p( values %dictpath )
#               ,
        p( 'Dictionary: ', escapeHTML( $dictdesc{ $dictionary } ) )
                ,
        start_form()
                ,
        textfield( -name => 'word' , -size => 40 , -maxlength => 40 )
                ,
        popup_menu( -name => 'dictionary' , -values => [ keys %dictpath ] )
                ,
        popup_menu( -name => 'limit' , -values => [ '', 10, 20, 30, 40, 50 ] ,
        -default => 50 )
                ,
        checkbox( -name => 'proper' , -checked => 1 )
                ,
        submit( -name => 'Enter' , -value => 'Enter' )
                ,
        end_form
                ,
        end_html
;
```

11 To-do list

- support multiple dictionaries: select 1 or more dictionaries, see which dictionary a word was found in
- search for dictionaries
- support ”proper” option
- sort and capitalize in output
- proper URI escape
- Language is a UTF-8 string