Bunny Numerics
A Number Theory Microworld

Craig Graci
Jack Narayan
Randy Odendahl

State University of New York, College at Oswego

Abstract. *A microworld designed for use in number theoretic investigations is described. This microworld, bunny numerics, is being used to complement the workhorse turtle geometry microworld in a Logo based problem solving course that we have recently initiated at SUNY Oswego. The microworld is defined, examples of its use are provided, suggestions for its use are offered, and a few notes on its implementation are made.*

Contents

1. Introduction

The bunny numerics micorworld was inspired largely by Seymour Papert's conception of how to create a curriculum, which is "to create a network of microworlds, each one focussing on different areas of knowledge."[1]

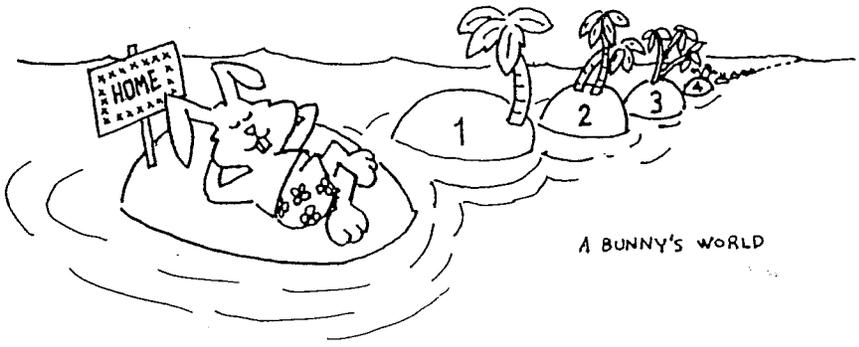
At SUNY Oswego we have recently introduced a two course sequence designed to satisfy a general education requirement in the area of mathematics and computation. The two courses, Elements of Problem Solving, Mathematics, and Computation, I and II, are grounded in Logo . They are intended to address mathematics in the broadest sense. That is, they aim to provide students with an understanding of the sorts of thought processes employed by mathematicians and computer scientists in their problem solving endeavors. In support of this course we have crafted a small number of microworlds which serve to complement the workhorse turtle geometry microworld.

This paper describes one of these microworlds, bunny numerics, which was designed for use in number theoretic investigations. Like the turtle geometry microworld, the bunny numerics microworld is embedded in Logo. However, since bunny numerics is an "add on," by contrast with turtle graphics which is inherent in Logo, the bunny numerics code must be explicitly loaded into the Logo system before it may be used.

Specifically, Sections 2, 3, 4, and 7 present the essential features of the bunny numerics microworld. Sections 5, 6, and 8 are intended to provide perspective. Section 9 contains brief remarks on the conceptual model underlying the bunny numerics microworld, and also on the use of Coral Software's ObjectLogo as the implementation language.

2. The Bunny World

The world of the bunnies may be thought of as an ocean dotted with a never ending "line" of islands. The islands are called home, 1, 2, 3, and so on.



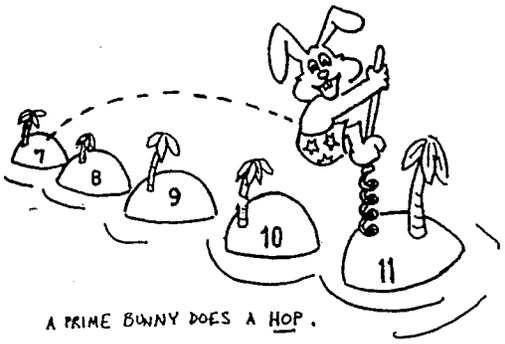
There are various *breeds* of bunnies, corresponding, in the main, to kinds of numbers. For example, there are odd bunnies, even bunnies, Fibonacci bunnies, and prime bunnies. A given breed of bunny is generally limited in terms of the islands that it can visit. A prime bunny, for example, can land only on the prime islands, the islands numbered 2, 3, 5, 7, 11, etc., and also on the Home island. A bunny knows never to set foot on an island which is not suited to its kind. All bunnies are comfortable at Home, which is also the birth place of all bunnies.

3. Basic Bunny Talk

Bunny talk is the set of Logo procedures one uses to communicate with bunnies. The most fundamental bunny talk procedures are: Hop, Location, Distance, HopAge, and HopHome. A brief description of each follows.

- Hop *bunny* (command)

  The specified bunny moves to the next highest numbered island to which its kind is suited.

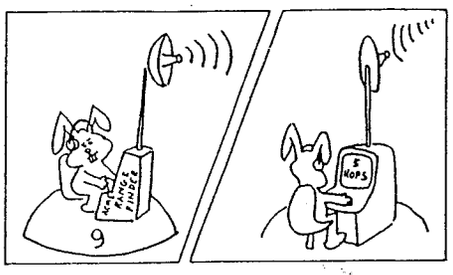


- Location *bunny* (operator)
  Loc *bunny*

  The "name" of the island on which the specified bunny is presently resting is returned.

- Distance *bunny* (operator)
  Dis *bunny*

  The number of hops that the specified bunny is from Home is returned.



- HopHome *bunny* (command)

  The specified bunny hops Home.

- HopAge *bunny* (operator)
  Age *bunny*

  The number of hops that the specified bunny has taken since its birth is returned.

4. Standard Bunnies, Breeds, and Birthing Operators

Standard bunnies are bunnies who are born when bunny numerics is loaded. Initially, they are found lounging at home. Standard bunnies were included with young children in mind, and will generally be ignored by "grown ups." A good way to begin thinking about number theory is to simply generate some sequences of numbers, and look for patterns. The reader may wish to refer to Table 1 when reading the following examples. Note, particularly, the *variables* that are used to denote standard bunnies.

```
? ;;view some squares, assuming Sammy is at home
? REPEAT 10 [ Hop :Sammy Type ( Loc :Sammy ) Type "| | ]
1 4 9 16 25 36 49 64 81 100

? ;;view some multiples of 4, assuming Mark4 is at home
? REPEAT 10 [ Hop :Mark4 Type ( Loc :Mark4 ) Type "| | ]
4 8 12 16 20 24 28 32 36 40
```

A generalization of this idea is in order. The following procedure takes a bunny as input and displays the "names" of the first few islands on which it comes to rest.

```
TO Sequence :b :n
  HopHome ;note that this is not a "bunny invariant" procedure
  REPEAT :n [ Hop :b Type ( Loc :b ) Type "| | ]
END

? ;;display the first 10 Fibonacci numbers
? Sequence :Flo 10
1 1 2 3 5 8 13 21 34 55

? ;;display the first 10 prime numbers
? Sequence :Pierre 10
2 3 5 7 11 13 17 19 21 23
```

Standard breeds are breeds of bunnies that exists when the bunny numerics system is loaded. One can create virtually any number of bunnies of a particular bunny breed through application of appropriate *birthing operators*. A simple illustration employing two bunnies of like breed in a harmonious way is given by the following procedure for displaying pairs of *twin primes*,

i.e., prime numbers which differ from one another by two. This procedure also typifies the representational independence characteristic of many solutions to number theory problems expressed in bunny talk.

```
TO DisplayTwinPrimes
  Make "b1 PrimeBunny
  Make "b2 PrimeBunny
  Hop :b2
  FOREVER
_ [ Hop :b1 Hop :b2
_   IF ( ((Loc :b2) - (Loc :b1)) = 2 )  [ Pr List ( Loc :b1 ) ( Loc :b2 ) ]
_ ]
END

? DisplayTwinPrimes
3   5
5   7
11   13
17   19
29   31
...
```

The table below identifies a *sampling* of the standard bunnies, breeds, and birthing operators.

Standard Breeds	Birthing Operators	Standard Bunnies
Even Bunny	EvenBunny	Ed
Square Bunny	SquareBunny	Sammy
Factorial Bunny	FactorialBunny	Fred
Fibonacci Bunny	FibonacciBunny	Flo
Multiple of i Bunny	MultipleBunny < *i* >	Mark1, Mark2, .. , Mark12
Divisors of n Bunny	DivisorBunny < *n* >	Di1, Di2, .., Di20
Prime Bunny	PrimeBunny	Pierre
Perfect Bunny	PerfectBunny	Pearl

Table 1: Some Standard Bunnies, Breeds, and Birthing Operators

The standard breeds were rather arbitrarily chosen, and are merely a small fraction of the interesting bunny breeds. For a complete listing of the standard bunny numerics entities see the *Bunny Numerics Report* [2]. The definition of nonstandard breeds is discussed in Section 7.

5. Selected Examples of Programming with Bunnies

5.1 Generating Simple Lists of Numbers

The following procedure simply displays a specified number of factorials.

```
TO DisplayFactorials :n
  Make "FB FactorialBunny
  REPEAT :n [ Hop :FB Print Location :FB ]
END
```

```
? DisplayFactorials 7
1
2
6
24
120
720
5040
```

Similarly, one could write a procedure to print out the first n primes, cubes, etc. Of course we could have called upon Sequence to list the Factorials, but as they quickly become very large, the placement of each on a separate line seemed appropriate. Several students, upon seeing the bunnies in action, have asked about how various sequences of numbers are generated. This is the sort of interest that we had hoped bunny numerics would generate! We earnestly encourage interested students to investigate the generation of various number sequences in terms of the more primative Logo procedures.

Displaying the multiples of a given number may be accomplished with the following procedure:

```
TO DisplayMultiples :m
  Make "MB MultipleBunny :m
  FOREVER [ Hop :MB Print ( Loc :MB ) ]
END
```

The multiple bunny breed is partitioned into subbreeds. The parameter provided to the birthing operator is used to select the particular subbreed from which the bunny is born.

5.2 Searching for Numbers with More than One Property

The example below presents a very primative solution to computing the least common multiple of two integers. Such illustrations can help to make notions meaningful to beginners. The "leapfrogging" technique employed by the bunnies is a common idiom used in bunny talk programming.

```
TO LeastCommonMultiple :n1 :n2
  LocalMake "Jack ( MultipleBunny :n1 "Jack )
  LocalMake "Jill ( MultipleBunny :n2 "Jill )
  Hop :Jack PrintLoc :Jack
  Hop :Jill PrintLoc :Jill
  WHILE [ NOT ( ( Loc :Jack ) = ( Loc :Jill ) ) ]
_ [
_   IFELSE ( ( Loc :Jack ) < ( Loc :Jill ) )
_   [ Hop :Jack PrintLoc :Jack ]
_   [ Hop :Jill PrintLoc :Jill ]
_ ]
  PrintLines 2
  ( Display "The LCM of " :n1 "| and | :n2 "| is:  | ( Loc :Jack ) )
END
```

```
? LeastCommonMultiple 4 7
location of Jack:  4
location of Jill:  7
location of Jack:  8
location of Jill:  14
location of Jack:  12
location of Jack:  16
location of Jill:  21
location of Jack:  20
location of Jack:  24
location of Jill:  28
location of Jack:  28

The LCM of 4 and 7 is:  28
```

As may be surmised from this example, several IO utilities are included with the bunny numerics microworld, e.g., PrintLoc, Display, PrintLines, and TypeSpaces. Moreover, the use of an optional *name* parameter, which may be supplied to any birthing operator, is employed in the calls to the MultipleBunny birthing operator. This name is used by the PrintLoc command. (In ObjectLogo, the application of a procedure with some number of inputs other than the standard requires a LISP-like use of parentheses).

The following less prolix example uses the same leapfrogging technique to compute and display prime Fibonacci numbers.

```
TO DisplayFiboPrimes
  LocalMake "FB FibonacciBunny
  LocalMake "PB PrimeBunny
  Hop :FB Hop :PB
  FOREVER
_ [  IFELSE ( ( Loc :FB ) = ( Loc :PB ) )
_     [ Print ( Loc :FB ) hop :Fb hop :PB ]
_     [ IFELSE ( ( Loc :FB ) < ( Loc :PB ) ) [ Hop :FB ] [ Hop :PB ] ]
_ ]
END

? DisplayFiboPrimes
2
3
5
13
89
233
1597
...
```

5.3 Divisors

The number sequences focussed on thus far have all been infinite. In contrast, the sequences of numbers corresponding to the subbreeds of divisor bunnies are among those which are, in a sense, finite. These "sequences" are somewhat artificial, but nontheless turn out to be very useful. A "divisor 10" bunny, for example, can land on the islands 1, 2, 5, and 10, in addition to Home. Recall the procedure Sequence:

```
? Sequence ( DivisorBunny 10 ) 8
1 2 5 10 home 1 2 5
```

The following procedure will neatly display the divisors of a given number.

```
TO DisplayDivisors :n
  LocalMake "Diva DivisorBunny :n
  REPEAT ( LongestTrip :Diva  ) [ Hop :Diva Type Location :Diva TypeSpace ]
  PrintLine
END

? DisplayDivisors 23
1 23
? DisplayDivisors 24
1 2 3 4 6 8 12 24
```

The LongestTrip operator is a part of bunny numerics. It computes the maximum distance from home that a bunny may find itself, and it may be applied to any bunny. The computation will terminate, however, only if the longest trip is finite.

The procedure below displays a table of divisors for the first n natural numbers.

```
TO DisplayTableOfDivisors :n
  FOR [ i 1 :n ]
   [ Type :i  TypeSpaces ( 6 - ( Count :i ) )  DisplayDivisors :i ]
END

? DisplayTableOfDivisors 6
1    1
2    1 2
3    1 3
4    1 2 4
5    1 5
6    1 2 3 6
```

Generating sequences of numbers and various tables for analysis is essential to finding patterns and making conjectures in the elementary theory of numbers. The bunnies can be extremely helpful in this regard.

6. Bunny Sets and Number Set Operators

There is provision in bunny numerics to create sets of numbers derived from number sequences. The principle bunny set constructor is *BunnySet*, which takes two inputs, a bunny, say b, and an integer, call it n. This operator returns a set corresponding to the first n elements of the sequence generated by b. There are also a variety of number set procedures included with the bunny numerics system for use with bunny sets. To illustrate:

```
? PrintSet BunnySet OddBunny 15
{ 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 }
```

```
? PrintSet BunnySet FibonacciBunny 7
{ 1 2 3 5 8 13 }
```

Notice the absence of two occurrences of "1" in the Fibonacci set. Note also that order is not significant within the braces.

This feature of bunny numerics can be used to describe many ideas cleanly. The procedure below, for example, generates primes using Eratosthenes' sieve method.

```
TO Sieve :n
  LocalMake "Numbers Diff ( BunnySet NaturalBunny :n ) ( Set 1 )
  LocalMake "Limit ( Sqrt :n )
  FOR [ i 2 :Limit ]
_ [
_   IF ( ElementOf :i :Numbers )
_   [
_     LocalMake
_       "SpecialSet  Diff ( BunnySet ( MultipleBunny :i ) :n ) ( Set :i )
_     Make
_       "Numbers ( Diff :Numbers :Specialset )
_   ]
_ ]
_ OUTPUT ( :Numbers )
END

? PrintSet Sieve 50
{ 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 }
```

The operators Set, Diff, ElementOf, and the command PrintSet are all part of the aforementioned number set procedures included with bunny numerics.

7. Nonstandard Bunnies

The procedure *NewBunnyBreed* is used to create a new breed of bunnies. For the general form of this operator, and details of its use, the *Bunny Numerics Report* [2] may be consulted. Below are two examples of the use of the NewBunnyBreed operator in defining new bunny breeds.

A breed of geometric progression bunnies could be defined as follows:

```
NewBunnyBreed "GeoBunny [ Base Mult ]
_ [ Make "CI :Base ]
_ [ Make "CI ( :CI * :Mult ) ]
```

CI stands for "Current Island." Recalling, again, Sequence from Section 4:

```
? Sequence ( GeoBunny 2 3 ) 5
2 6 18 54 162
? Sequence ( GeoBunny 5 9 ) 3
5 45 405
```

As a second example, a breed of *wonder* bunnies may be defined in order to investigate the "wondrousness" number property discussed by Achilles and the Tortoise in Douglas Hofstadter's *Aria with Diverse Variations.*[2]

```
NewBunnyBreed "WonderBunny [ i ]
_ [ Make "CI :i ]
_ [ IFELSE ( Odd :i ) [ Make "CI ( :CI * 3 ) + 1 ] [ make "CI ( :CI / 2 ) ] ]
```

The following procedure *might* then be used to verify that a given number is, indeed, wondrous.

```
TO IsWondrous :i
  Make "WB WonderBunny :i
  FOREVER [ Hop :WB  IF ( ( Loc :WB ) = 1 ) [ OP "True ] ]
END
```

8. Uses of Bunny Numerics

One can use bunny numerics in the ways alluded to thus far: to generate number sequences; to find numbers with particular properties; to test conjectures. The student should typically, perhaps with some direction from the teacher, read some of the history and lore of number theory, identify interesting questions, and explore these questions with the aid of the bunny numerics microworld.

Beyond this, one might exploit the bunny numerics microworld in a number of ways. For example, a tried and true induction game based on guessing the next number in a sequence can be nicely automated in the context of bunny numerics. We have written a version called INDUCE which takes a bunny as input, generally a nonstandard bunny of our own design, and then interacts with the player offering the opportunity to guess the underlying rule. We employ a very simple acceptance procedure. If the player can correctly identify the next three numbers in the sequence, we credit the player with knowing the rule. Each time the person fails to guess the rule, the next number in the sequence is divulged. The distance from Home of the bunny at the time the rule is finally guessed is displayed at the end of a game. INDUCE is quite like WHEEL OF FORTUNE, only a bit more interesting from the perspective of a mathematician - with one notable exception, perhaps.

The generation and solution of cross number puzzles are activities enhanced by the bunny numerics microworld. An interesting Artificial Intelligence project within the context of Logo, employing both the turtle graphics and the bunny numerics capabilities, would be to completely automate the generation of cross number puzzles. Good puzzles must have a certain "degree of interest" which is sufficiently difficult to describe as to, indeed, render their automatic generation a project within the domain of Artificial Intelligence. Regardless of how they are created, cross number puzzles

present very nice opportunities to apply strategies of *constrained search*, an important aspect of problem solving.

9. Some Implementation Notes

Our implementation of the bunny numerics microworld took very little time, largely because of the nature of the language that we used, namely Coral Software's ObjectLogo.

We exploited the object oriented features of ObjectLogo in modelling the bunnies. All bunnies have certain commonalities, e.g., they can all hop, determine their location, determine their distance from home, determine their "hopage," and find their way home. Thus a generic Bunny class was established as a direct subclass of the Logo class. Due to the fact that different bunny breeds hop in dramatically different ways, each breed requires its own refinement of the Hop procedure. Also, each bunny requires its own set of state variables, and thus its own birthing operator. The natural thing to do was to make each breed a subclass of the generic bunny class. This was all a straightforward exercise in object oriented programming. What was less straightforward was establishing nonstandard breeds as subclasses of the generic bunny class, "under the table" so to speak.

We exploited ObjectLogo's very direct kinship with LISP in order to achieve the undertaking just mentioned. Basically, the NewBunnyBreed procedure programmably generates ObjectLogo programs required to establish the new bunny breed classes as subclasses of the generic bunny class.

We also exploited ObjectLogo's inherent ability to operate on large integers. Multiline (hundreds of digit) factorials and perfect numbers, for example, are readily computed through bunny numerics, in a manner consistent with the computation of small factorials and perfect numbers.

10. Concluding Remarks

We are only now using the bunny numerics microworld in the first of our two new course offerings at SUNY Oswego. We hope soon to report on its successes and failures.

Citations

1. Papert, S. "MICROWORLDS: Transforming Education," in *Artificial Intelligence and Education, Volume One*, edited by R. Lawler and M. Yazdani, Ablex Publishing Co., 1987, page 60.

2. Hofstadter, D. *Godel, Escher, Bach,* Vintage Books, 1980, pp. 400 and 401.

References

[1] Coral Software: *ObjectLogo Reference Manual.* Coral Software Corp., 1986.

[2] C. Graci: *Bunny Numerics Report.* SUNY Oswego Department of Computer Science, 1989.

[3] D. Hofstadter: *Godel, Escher, Bach.* Vintage Books, 1980.

[4] R. Lawler and M. Yazdani (ed): *Artificial Intelligence and Education, Volume One.* Ablex Publishing Co, 1987.

[5] S. Papert: *Mindstorms: Children, Computers and Powerful Ideas.* Basic Books, 1980.