DICE – Dice Rolling with Graphics

© 2019 Valentín Albillo

Abstract

DICE is a program written in 1980 for the HP-41C programmable calculator to simulate randomly throwing two dice, which are printed graphically followed by their sum. A subroutine can be globally called to accumulate the graphics for any single die face. Both standard and synthetic programming versions are included.

Keywords: *Dice, simulation, program, graphics, printer, programmable calculator, RPN, HP-41C, synthetic programming*

1. Introduction

DICE is an *RPN* program I wrote in 1980 for the *HP-41C* programmable calculator (will also run *as-is* in the *HP-41CV/CX)*, to simulate any number of random throws of two dice, which are printed graphically (followed by their sum) using a compatible attached printer.

DICE calls a globally addressable subroutine *"$"*, which accepts as input an integer from 1 to 6 in the display (stack **X** register) and accumulates in the printer buffer the graphics for the corresponding die face having that number of pips. The graphic is just accumulated but *not* printed, so that the caller program can add more graphics or text to the printer buffer before printing the whole contents by executing **PRBUF**, **ADV** or any other operation which causes the buffer to be printed. If *"$"* is called right from the keyboard, the user can afterwards press the **ADV** button on the printer (or execute the same command form the calculator's keyboard) to print the graphics for the die face. Additionally, a much shorter and faster version of "$" using *synthetic programming* is also included, see below.

The random value for each die is produced using a simple but fast and effective *pseudo-random number generator* which requires the user to first store a *seed* in register R_{01} before calling the *DICE* program (no seed is needed when calling the subroutine *"$"*). This seed must be a positive value (see **Note 1** for other restrictions), and the user needs to store it just *once* per session, no matter how many dice throws are generated afterwards

1.1 Synthetic programming version of "$"

In addition to the normal version of *"$"*, which uses just the standard function set available right out of the box, a much shorter an faster version is also listed, which uses *synthetic programming* techniques. A discussion of synthetic programming if well out of scope for the present paper but you can consult the **References** (the second one in particular) for full information on it, how to create synthetic lines and free utilities to help creating them.

The net result is that by including 7 synthetic lines the subroutine *"$"* is much shorter (30 steps, 84 bytes vs. 58 steps, 112 bytes, almost a 50% reduction) and also much faster than the normal version. This synthetic version uses the techniques pointed out in *PPC V7 N6 pp27-28* (see **References**) to create the synthetic text lines. The required **BLDSPEC** string is previously written down using the techniques described in *V7 N5 p56* so that every 7 columns of dots are accumulated into the printer buffer as a **BLDSPEC** character by the byte-saving procedure of first creating a string representing the desired dot-pattern for the character (partial graphics for each die face), then the synthetic instruction **RCL M** is used to retrieve the data from the Alpha register, followed by executing **ACSPEC** to accumulate the special character (first 7 columns of the die representation). The remaining two columns are always the same for all six faces so they're simply accumulated using **ACCOL**.

To wit, this is a fine example of the power and convenience of using synthetic functions: the resulting version is much shorter and faster than the conventional one using standard functions and all are advantages, no *caveats*, no negative collateral effects at all. Using synthetic functions is a great way to improve programs, often drastically, and also of accomplishing tasks impossible to achieve with just standar programming. It's just a matter of understanding the straightforward concepts involved and to get and use the proper, freely available tools.

2. Program Listing

2.1 Standard programming version of both DICE and "$"

```
01 ♦LBL "DICE"
02  ADV
03  XEQ 00 ►
04  STO 00
05  XEQ "$" ►
06   6
07  SKPCOL
08  XEQ 00 ►
09  ST+ 00
10  XEQ "$" ►
11   6
12  SKPCOL
13   61
14  ACCHR
15  RCL 00
16  CF 28
17  CF 29
18  FIX 0
19  ACX
20  PRBUF
21  RTN ►
22 ♦LBL 00
23  RCL 01
24  R-D
25  FRC
26  STO 01
27   6
28   *
29   1
30   +
31  INT
32  END
```

```
01 ♦LBL "$"
02  XEQ 07 ►
03  SF 12
04   65
05  ACCOL
06  XEQ IND Z ►
07  ACCOL
08   65
09  ACCOL
10 ♦LBL 07
11  CF 12
12   127
13  ACCOL
14  RTN ►
15 ♦LBL 02
16   69
17  XEQ 00 ►
18  ACCOL
19  ACCOL
20   81
21  RTN ►
22 ♦LBL 03
23   69
24  XEQ 00 ►
25   73
26  XEQ 00 ►
27   81
28  RTN ►
29 ♦LBL 04
30   85
31  XEQ 00 ►
32  ACCOL
33  ACCOL
34  X<>Y
35  RTN ►
36 ♦LBL 05
37   85
38  XEQ 00 ►
39   73
40  XEQ 00 ►
41   85
42  RTN ►
43 ♦LBL 06
44   85
45  XEQ 00 ►
46  X<>Y
47  XEQ 00 ►
48  X<>Y
49  RTN ►
50 ♦LBL 01
51  ACCOL
52  ACCOL
53   73
54 ♦LBL 00
55  ACCOL
56  X<>Y
57  ACCOL
58  END
```

- *this version of global subroutine "$" uses just **standard** programming techniques, no synthetic lines required.*
- *32 steps (60 bytes) + 58 steps (112 bytes), will fit in a basic HP-41C with no memory modules, printer required*
- *clears flags 28 and 29 and sets display mode FIX 0.*
- *to get * press* **x** *, to get "text" press* **ALPHA**
- *the symbols ♦ and ► are purely cosmetic, to visually indicate branching, don't try to key them in.*

2.2 Synthetic programming version of "$"

```
01 ♦LBL "$"
02  SF 12
03  see side notes
04  XEQ IND X ►
05  RCL M
06  ACSPEC
07   65
08  ACCOL
09   127
10  ACCOL
11  CF 12
12  RTN ►
13 ♦LBL 01
14  see side notes
15  RTN ►
16 ♦LBL 02
17  see side notes
18  RTN ►
19 ♦LBL 03
20  see side notes
21  RTN ►
22 ♦LBL 04
23  see side notes
24  RTN ►
25 ♦LBL 05
26  see side notes
27  RTN ►
28 ♦LBL 06
29  see side notes
30  END
```

***Notes** on the synthetic text lines used:*

- *Line **03** is:* **F2 11 FE**
- *Line **14** is:* **F6 7F 0C 18 32 60 C1**
- *Line **17** is:* **F6 7F 0C 58 30 60 D1**
- *Line **20** is:* **F6 7F 0C 58 32 60 D1**
- *Line **23** is:* **F6 7F 0D 58 30 60 D5**
- *Line **26** is:* **F6 7F 0D 58 32 60 D5**
- *Line **29** is:* **F6 7F 0D 58 35 60 D5**

- *this version of global subroutine "$" uses **synthetic programming** techniques.*
- *30 steps (84 bytes), will fit on a single side of a magnetic card.*
- *sets flag 12 on entry and clears it before returning.*

3. Usage Instructions

See the following examples to understand how to use both the program *"DICE"* and the subroutine *"$"*.

4. Examples

The following examples can be useful to check that the program is correctly entered and to understand its usage:

4.1 Example 1

Using *0.5301* as a seed, produce three consecutive dice throws.

0.5301 **STO** 01 *(store the seed for the RNG just once per session, no matter how many throws are generated afterwards)*

XEQ "DICE" → = 6

XEQ "DICE" → = 10 *(pressing R/S could have been used instead of repeating XEQ "DICE")*

XEQ "DICE" → = 8 *(ditto)*

4.2 Example 2

Print all individual die faces from 1 to 6 pips. *(we must press the printer's* **ADV** *button after each to cause printing)*

1 **XEQ** "$" **ADV** → 2 **XEQ** "$" **ADV** →

3 **XEQ** "$" **ADV** → 4 **XEQ** "$" **ADV** →

5 **XEQ** "$" **ADV** → 6 **XEQ** "$" **ADV** →

Notes

1. Don't use *0* or negative seeds and also avoid **PI** and its multiples or fractions, as well as very large numbers.

2. This program was submitted to ***PPC Technical Notes*** and it was published in the *September 1980* issue *(PPCTN V1N2 p64).*

References

Valentin Albillo — *Dice Rolling – With Graphics (PPC Technical Notes, V1 N2 p64, Sep 1980)*

W.C. Wickes (1980) — *Synthetic Programming on the HP-41C*

W.C. Wickes (1980) — *Understanding BLDSPEC (PPC V7 N5 p56, June 1980)*

Jake Schwartz (1980) — *Full Wand BLDSPEC Control (PPC V7 N6 pp27-28, Jul/Aug 1980)*

Copyrights

Copyright for this paper and its contents is retained by the author. Permission to use it for non-profit purposes is granted as long as the contents aren't modified in any way and the copyright is acknowledged.