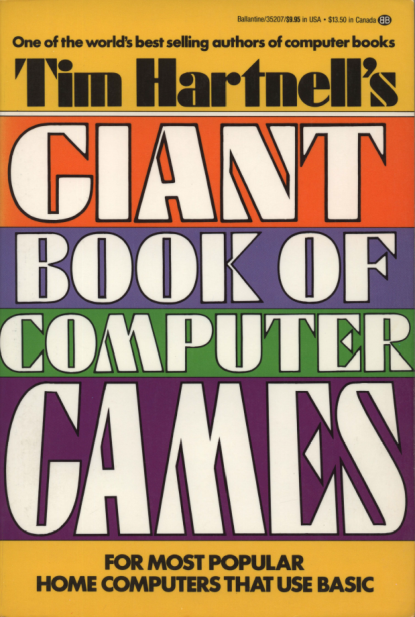
# Stronghold of The Dwarven Lords

## A Beginner / Intermediate Python Game

### Introduction

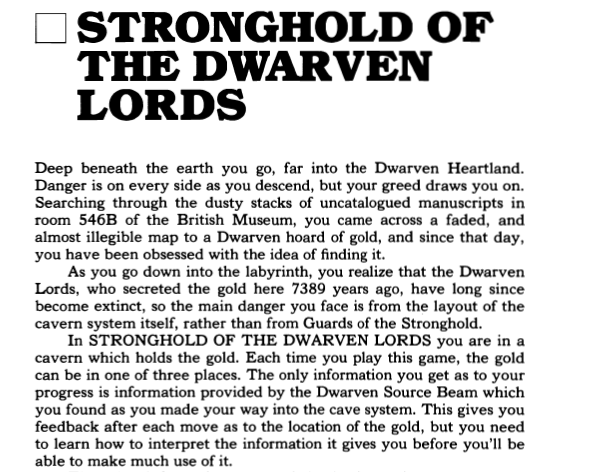
In the early 1980s, as early home computers boomed in popularity, a little programming knowledge was essential. My first machine was a Sinclair ZX81 with 1K of memory (I used to dream of being able to afford a 16K RAM expansion!). To make the ZX81 do anything at all you had to know a few BASIC commands, and the computer’s manual included a good introduction to BASIC programming. For those interested in more advanced programming, magazines like *ZX Computing* and *Creative Computing* included listings of BASIC programs – mostly games, which could be typed in, played, saved and hacked to your heart’s content. It was great fun and a fantastic way to learn how to program.

The late Tim Hartnell was a prolific author of BASIC games and published many articles and over 30 books during the early ‘80s. Second-hand copies of his books are still readily available online and a few PDFs can be found on vintage computer fan websites.

Stronghold of The Dwarven Lords is a simple adventure game, from the 1984 book *Tim Hartnell’s Giant Book of Computer Games.* In my Python translation I try to stick as closely as possible to Tim’s original BASIC version. To get going open the Python IDLE 3 Shell, then hit Ctrl+N to open a new window. I suggest typing the program in manually (just like we would have done 30 years ago) and trying to work out how it works as you go. Don’t type in the line numbers – they are included to make the listing easier to navigate.

|  |  |
| --- | --- |
| 1 | # STRONGHOLD OF THE DWARVEN LORDS v2.1 |
| 2 | # Martin Hodgson - November 2013 |
| 3 | # Translated from Tim Hartnell's original BASIC version... |
| 4 | # ...with a couple of updates. Now you can't walk through walls! |
| 5 |  |
| 6 | **import** random |
| 7 |  |
| 8 | # VARIABLES, LISTS |
| 9 | data1 **=** [2,2,2,3,2,4,2,5,2,6,2,7] |
| 10 | data2 **=** [3,7,4,7,5,7,5,6,5,5,5,4,5,3,6,3] |
| 11 | data3 **=** [7,3,7,4,7,5,7,6,7,7,7,8,7,9,9,8] |
| 12 | data4 **=** [9,9,10,8,10,7,10,6,10,5,10,4,8,8] |
| 13 | data5 **=** [10,3,11,3,12,3,13,3,14,3,14,2,7,10] |
| 14 | data6 **=** [6,10,5,10,4,10,3,10,2,10,2,11,2,12] |
| 15 | data7 **=** [2,13,2,14,6,11,6,12,6,13,6,14,7,12] |
| 16 | data8 **=** [14,12,8,12,8,14,9,12,9,13,9,14,10,12] |
| 17 | data9 **=** [11,9,11,10,11,11,11,12,12,9,13,9,13,10] |
| 18 | data10 **=** [13,11,13,12,13,13,13,14,14,14] |
| 19 | data **=** data1 **+** data2 **+** data3 **+** data4 **+** data5 **+** data6 **+** data7 **+** data8 **+** data9 **+** data10 |
| 20 |  |
| 21 | # FUNCTIONS |
| 22 | **def** new\_game(): # GOSUB 640 in original BASIC version |
| 23 | **global** a, b, z, y, x, s, m, a, e, d |
| 24 | **print** ("================================================================\n") |
| 25 | input ("STRONGHOLD OF THE DWARVEN LORDS\nNew Game - Press Enter...") |
| 26 | # Item zero and the zero at the beginning of each sub-list will be ignored... |
| 27 | # ... as the BASIC program uses indices 1-15 |
| 28 | a **=** [[0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0]] |
| 29 | b **=** random.randint(1, 3) |
| 30 | z, y **=** 14, 14 |
| 31 | **if** b **==** 2: |
| 32 | y **=** 2 |
| 33 | **if** b **==** 3: |
| 34 | z **=** 2 |
| 35 | x, s **=** 1, 2 |
| 36 | **for** b **in** (range(1, 16)): |
| 37 | **for** c **in** (range(1, 16)): |
| 38 | a[b].append(x) |
| 39 | **if** random.randint(1, 10) > 8: |
| 40 | a[b][c] **=** s |
| 41 | **if** c<2 **or** c>14 **or** b<2 **or** b>14: |
| 42 | a[b][c] **=** x |
| 43 | d, e **=** 2, 2 |
| 44 | **for** f **in** (range(0, 136, 2)): |
| 45 | b **=** data[f] |
| 46 | c **=** data[f**+**1] |
| 47 | a[b][c] **=** s |
| 48 | a[z][y] **=** s # Makes sure the gold isn't in a wall |
| 49 | m **=** **-**15 |
| 50 | **return** |
| 51 |  |
| 52 | **def** show\_map(): # GOSUB 480 'help' in original BASIC version |
| 53 | **global** b, m |
| 54 | **print** ("\n================================================================\n") |
| 55 | **print** ("North") |
| 56 | b **=** 15 |
| 57 | **while** b > 0: |
| 58 | strng **=** "" |
| 59 | **for** c **in** (range(1, 16)): |
| 60 | **if** a[b][c] **==** x: |
| 61 | strng **+=** ("#") |
| 62 | **elif** b **==** d **and** c **==** e: |
| 63 | strng **+=** "\*" |
| 64 | **elif** a[b][c] **==** s: |
| 65 | strng **+=** " " |
| 66 | **print** (strng) |
| 67 | b **-=** 1 |
| 68 | print ("South") |
| 69 | m **+=** 15 |
| 70 | a[d][e] **=** s |
| 71 | # Here I've omitted two lines from the BASIC version: |
| 72 | # 600 FOR J = 1 TO 2000:NEXT J - Makes the prgram pause. |
| 73 | # 610 CLS - Clear screen. Not possible in Python Shell? |
| 74 |  |
| 75 | **def** move(): # Lines 50 to 410 - Main game script from BASIC version |
| 76 | **global** m, d, e |
| 77 | m **+=** 1 |
| 78 | **print** ("\n================================================================\n") |
| 79 | **print** ("STEP NUMBER", m) |
| 80 | **if** a[d**+**1][e] **==** s: |
| 81 | print ("NORTH: OPEN") |
| 82 | **elif** a[d**+**1][e] **==** x: |
| 83 | **print**("NORTH: WALL") |
| 84 | **if** a[d**-**1][e] **==** s: |
| 85 | **print** ("SOUTH: OPEN") |
| 86 | **elif** a[d**-**1][e] **==** x: |
| 87 | print("SOUTH: WALL") |
| 88 | **if** a[d][e**+**1] **==** s: |
| 89 | print ("EAST: OPEN") |
| 90 | **elif** a[d][e**+**1] **==** x: |
| 91 | print("EAST: WALL") |
| 92 | **if** a[d][e**-**1] **==** s: |
| 93 | print ("WEST: OPEN") |
| 94 | **elif** a[d][e**-**1] **==** x: |
| 95 | print("WEST: WALL") |
| 96 | **print** ("THE DWARVEN SOURCE BEAM READS:", (100 **\*** abs(z**-**d) **+** 10 **\*** abs(y**-**e))) |
| 97 | print ("Which direction do you want to move...") |
| 98 | a\_string **=** input ("N - north, S - south, E - east, W - west, H - help ? ") |
| 99 | **if** a\_string **==** "H": |
| 100 | show\_map() |
| 101 | **elif** a\_string **==** "N": |
| 102 | d **+=** 1 |
| 103 | **elif** a\_string **==** "S": |
| 104 | d **-=** 1 |
| 105 | **elif** a\_string **==** "E": |
| 106 | e **+=** 1 |
| 107 | **elif** a\_string **==** "W": |
| 108 | e **-=** 1 |
| 109 | **if** z **==** d **and** y **==** e: |
| 110 | win() |
| 111 | **if** a[d][e] **==** x: # In the original you could walk through walls... Now you can't! |
| 112 | **print**("\nOuch! You just walked into a wall...") |
| 113 | **if** a\_string **==** "N": |
| 114 | d **-=** 1 |
| 115 | **elif** a\_string **==** "S": |
| 116 | d **+=** 1 |
| 117 | **elif** a\_string **==** "E": |
| 118 | e **-=** 1 |
| 119 | **elif** a\_string **==** "W": |
| 120 | e **+=** 1 |
| 121 | move() |
| 122 |  |
| 123 | **def** win(): |
| 124 | **print** ("\nYou found the Dwarven riches in just", m, "steps!\n") |
| 125 | show\_map() |
| 126 | # This feature has been added - The original version would just END. |
| 127 | new\_game() |
| 128 | show\_map() |
| 129 | move() |
| 130 |  |
| 131 | # MAIN PROGRAM |
| 132 | new\_game() |
| 133 | show\_map() |
| 134 | move() |

### How to play

First, read the following introduction by Tim Hartnell , whose prose revealed his great enthusiasm for game programming…

When the game starts you are shown a map of the cavern system with your position indicated as an asterisk \*. Your aim is to find the hidden gold with the minimum number of steps. Each turn you are given a brief description of your surroundings and must choose the direction to take next. Once the map scrolls off the screen you are only allowed to look at it again by choosing H for help. However, each time you ask for help you are penalised 15 steps.

### How the program works

For those with a basic understanding of Python, you’ll see that the main parts of the program are structured as follows:

* Line 8 – A few lists of numbers that are all added together on line 19 to make one big list.
* Line 21 - Functions: *new\_game, show\_map, move* and *win*
* Line 132 - Main program – calls the functions in order to initiate the first game

Tim’s original variable names aren’t very descriptive. For me this added to the fun of working out how the program works. My code only uses basic Python constructs. The new\_game function is probably the most complicated part of the program; it creates a list of 15 x 15-item ‘sub-lists’ that represents the map. This is done by first, creating a grid that is designated as ‘all wall’, but with a few random spaces, then adding more spaces according to the *data* list. Hopefully this will become clearer as you type in the program in and play the game.

Have fun finding the gold!

Martin Hodgson, November 2013

[www.nerdsville.org](http://www.nerdsville.org)

[martinhodgson@hotmail.com](mailto:martinhodgson@hotmail.com)