

A submission for the [**2020 BASIC Ten Liner competition**](https://gkanold.wixsite.com/homeputerium/kopie-von-2020) (EXTREM-256 category)

By Randy Gill (randygill at yahoo dot com)

**Introduction**

Wizard of Wasd (rhymes with “paused”) is a fantasy adventure game with overhead graphics. It is written in the **[FastBasic](https://github.com/dmsc/fastbasic)** language, and runs on Atari 8-bit computers (or emulators) with 48K+ RAM.

The goal is to find the magical “Dagger of Venn” and use it to destroy the evil Blue Wizard. You must accomplish this before time runs out. If you do not, he will complete his spell and raise legions of undead to ravage the peaceful Land of Wasd!



**How to Run**

1. Download the ATR-format disk image file WIZOWASD.ATR using [**this link**](http://angrydill.com/wasd/WIZOWASD.ATR).
2. Write the image to a 130KB “enhanced density” diskette, and load it into your Atari computer's D1: drive.  
   – *or* –  
   Load file WIZOWASD.ATR as disk D1: in your favorite Atari emulator.
3. Turn on your Atari computer (or cold-reset [reboot] your emulator) and the program should load and run automatically.

**Gameplay Instructions**

* Move North / West / South / East using “W” / “A” / “S” / “D” (respectively) on the keyboard; those are the only controls you need!
* Item are picked up automatically when you walk onto them.
* Obstacles and enemies are defeated by running into them, provided you have the required inventory item.
* If you *don’t* have what you need, nothing bad will happen, so feel free to experiment!
* To quit the game before the time runs out, press the [Esc] key.

**Gameplay Tips**

* Grab everything you can! There is no limit on player inventory, and everything you can take is needed *somewhere*.
* There are *no* “hidden passages”, so you don't have to waste time ramming into every wall you see.
* Nothing in the game is random.
* If you run out of time, try playing again! The more you know about the Land of Wasd, the faster you’ll solve its challenges.
* If you do win, try playing again to see if you can get a better remaining time score.
* The author’s personal best remaining time is 560; see if you can match it, or even beat it!

**Backstory**

The evil wizard “Balkor the Blue” has returned — to again threaten the peaceful land of Wasd! He has sealed himself off at the top of the tower on Eastern Island, protected by one of his undead minions. There the sorcerer is preparing the ingredients of a powerful spell, which when completed will raise hordes of undead creatures, and surely spell doom for all life in the land!

There is *only one hope* for defeating Balkor the Blue. Centuries ago the magical Dagger of Venn was forged using the combined powers of the nine most powerful good wizards in the realm, in case the evil wizard should ever return. Alas, the dagger was hidden away in a long-since forgotten location.

Your quest is to find this magic dagger, and use it to destroy the Blue Wizard once and for all. But you must complete your quest before time runs out, and he deploys his deadly spell!



**About the Author**

My name is Randy Gill, and I'm from Arlington, Texas, USA. I'm often online under the anagrammatic nickname "AngryDill." My day job involves software development, but working with business systems, sadly not games. I got my first computer (an Atari 800) at age 11, complete with two cartridges — Basketball and Atari BASIC. I started programming because I eventually got tired of shooting digital hoops.

I've always loved playing games, and while I enjoy many genres (RPG, Action-Adventure, FPS, etc.), my favorite aspects of them are always exploration and puzzle-solving. I wanted to create a simple game that distilled those elements down to their essentials, so I created this little adventure — my first Atari program in three decades! I hope you enjoy playing it, and maybe find the source code interesting and informative.

**Acknowledgements**

I would like to express my gratitude to each of the following wonderful people, without whom this game would have never been made:

* My good friend “Doctor Clu”, for rekindling my interest in vintage Atari, and for loaning me lots of gear to get back into it.
* The hosts of [**ANTIC, the Atari 8-Bit Podcast**](https://ataripodcast.libsyn.com/) (Brad, Kevin, and Randy) for informing me about the 10-liner competition, and for all they do to disseminate and preserve information about my favorite retro system.
* The makers of many tools I used during the development of this game, including [**Atari800**](https://atari800.github.io/) (my favorite 8-bit emulator), [**Vim editor**](https://www.vim.org/) (in which most of the code was written), [**Tiled**](https://www.mapeditor.org/) (used for designing the world map), **[Pixelesque](https://play.google.com/store/apps/details?id=com.rj.pixelesque&hl=en_US)** (used for drawing the custom character glyphs), [**GIMP**](https://www.gimp.org/) (used for editing the graphics), [**Python**](https://www.python.org/) (in which I wrote the utilities to extract, compress, and encode the game data), [**BW-DOS**](https://atariwiki.org/wiki/Wiki.jsp?page=BEWE%20DOS%201.30%20Manual) (command-line-based Atari DOS), and *especially* **[FastBasic](https://github.com/dmsc/fastbasic)**.
* Above all, my very loving (and very *lovely*) wife Debbie, who supports me in all the things I do!

**Minified Source Code**



**Formatted and Commented Source Code**

. ==============

. WIZARD OF WASD

. ==============

. A submission for the 2020 BASIC 10-liner competition

. By Randy Gill (randygill at yahoo dot com)

. Designed for (integer) FastBasic v4.0, for Atari 8-bit computers

. Dimension arrays to hold expanded game data, packed source data, inventory

DIM W(1520), D$(5), V(51)

. Initialize address pointer variables

Y = (ADR(W) / 512 + 1) \* 512 :. start address of custom glyphs

K = Y + 512 :. start address of entity catalog

M = K + 260 :. start address of map data (skipping 52 entities \* 5 bytes)

. Copy character set into RAM for modification

MOVE $E000, Y, 512

. =================================

. EXPAND COMPRESSED GAME DATA

. =================================

. Extract data for custom character glyphs, entity attributes, and game map to

. available memory. Game data is packed in strings with each character repre-

. senting 6 bits (a "sextet"). Data is decoded/expanded as a series of variable-

. length chunks, each prefixed by a two-sextet "header" defining encoding mode

. number (3 bits) and length (9 bits)

. Encoding mode numbers

. ---------------------

. 7 = Octet-Stream: Encodes 3 bytes in 4 printable chars; like Base64 but

. easier to decode in BASIC (no translation table or 32-bit ints required)

. 4 = Sextet-Stream: Packs a 6-bit value into each char; for values < 64, e.g

. table of entity relationships

. 3 = Triad-Stream: Packs two 3-bit values (< 8) into each char; useful for

. base map (having only common entities, 0-7)

. 2 = Sextet-Run: encodes an sextet that repeats a specified # of times (6-bit

. RLE)

. 1 = Octet-Run: encodes an octet that repeats a specified # of times (8-bit

. RLE)

. Modes that aren't true encodings

. --------------------------------

. 5 = Header: indicates chunk header is being processed

. 0 = Gap: advances destination pointer by specified count; used to skip char

. glyphs not being redefined

. The encoding mode numbers (variable E) were carefully chosen so that whether

. the encoding uses a holding sextet, the cycle of when holding sextets appear,

. and whether the encoding is a "run" (vs. a "stream") could be calculated from

. it. Respectively: ABS(E-3)>1, ABS(E-3+(E=3)), and E<3

. Initialize vars...

Q = Y :. data destination pointer (set to starting address of custom char glyphs)

E = 5 :. encoding mode (initialized to "header" encoding, which prefaces a chunk)

R = 1 :. remaining bytes in current chunk or header

I = 0 :. index of current element in source data string array $D()

P = 0 :. pointer to source data character (RAM addr within $D(I))

L = 0 :. limit (ending) RAM address of current source data element $D(I)

C = 0 :. count within decoding cycle (significant for header and octet encodings)

. Show indicator that data is being loaded

? "Load";

. D$(n) holds compressed data for custom glyphs, game entities, and the game map

D$(0)="08?hoooWo33;\_G[@1@`@0842@1h\ZZZZjZZnThlj\_ooj>Zn0In1N4P`P0P371>lHlX40oooWo333?33H`LHo1nlH5V@DhbkoOofVdRbj3hR

WJ\_?P5lZ8088888IS=fH14RdN5hUU"

D$(1)="6Ioi=mm00HHl1JHTEVVgeg4dOoFl8@7@=00P0D>OP?70=Pl6d<io1n183o7P00H0@l0nQn0X5hX`0842T1nmNIWnDZDn00lHDlnJ1nlH@78

FAjFMOZJ9^90PGL;0035LF05:L00Lc62L00;<E0?L21<9d56\_h097W0?7<03751?7M1?7`1<?P18?226\_O2;7d2;7f2?M[53M05>\_l46\_b50\_254\_T

5=\_j45\_Z53\_457\_\57Mn3;M"

D$(2)="549L46?Mm5;MB53M;57Ml63600;5;504006JN442C10N0043V58;<34PG62=@63OV51KS6;1d249`5;k=388>1770o0o01o20o39S\en7<H

m@mCmMmNmOmDm?m>m8mBmAm9m=mFmEmHmGmJmImLmKm:m;m<mQmPmSmRn0n0n2n0n2n0n0n0[TT0U0V0WAXYY0ZV:F43999TB42A43;AB9BB42@43=

9B99B]42>43>9A:=AZU2=43"

D$(3)="B99Z]AZ]9Q2;43AAZ=9B]E942943B9Z]99Z]:Q2943@AZ=Q9]E:2;43?AZ=Q]EB12<43>AZ:T]BR2<43C9AB9TERT:42843C9B:9QDTAB42

843CABY=9T<BB42843A9:]]99<>22:43B99]]=9<9Q2943C99]]=Q<9942843B9<YY]9T9Q2943>9T99A:Q2=43?9T<9BB942<4369Q<2627401110

42;4369QA2627501150542;"

D$(4)="43B99BBB9]=Q2943699A26136]=Q2:43@999T99=Q:F47106:E671002807306062703Cfff``ff`f0276710026671002603Y6666006ff

f`ff`````f`02663L00`0````0``ff62963;````f02663^``00`0``00`ff666f66f6f62863:00```2707306062763C`ff`f``f`028671002:0

3;600606:F63@0`0`0`0`2;"

D$(5)="63`0`0`0`0`f0`ff60606060f612663E0`0`0`0`f802663F0`0`0`0`f0`:c63;0`fhh02>63a```f00`f99aff606fn0hfVLT62963B`f

07`fTSd2663E00f600f>9962G6710O:F600"

REPEAT

IF P = L :. source data element ran dry (or 1st time through)

P = ADR(D$(I)) + 1 :. calculate source data pointer

L = P + LEN(D$(I)) :. ending address of source data array element

INC I :. next time it runs dry, it will use next element of D$

ENDIF

S = PEEK(P) - 48 :. set current sextet, subtracting the character offset

IF C = 0 AND ABS(E - 3) > 1 :. first byte of a multi-byte encoding becomes the "holding" sextet

H = S :. set the holding sextet

ELIF E = 5 :. "header" encoding; two-character sequence that defines following chunk

? "."; :. print periods to show it is busy working

E = H & 7 :. get the next encoding id number from 3 low bits of holding sextet

R = H & 56 \* 8 + S :. high 3 bits from the holding sextet plus 6 from current form (9-bit) chunk length

H = 0 :. clear holding sextet for next use

C = -1 :. set cycle count so it will be 0 next iteration (after incrementing)

IF E = 0 :. "gap" encoding

Q = Q + R :. advance the destination pointer Q

E = 5 :. back to header encoding for next chunk

ENDIF

ELSE

IF E < 3 :. sextet-run or octet-run encoding

MSET Q, R, H \* 64 + S :. fill R bytes starting at Q. Note that for sextet, H will be 0

Q = Q + R :. skip over range it just filled

R = 1 :. set to 1 because < 1 would signal prog to stop decoding altogether

ELSE

IF E = 3 :. triad-stream encoding

POKE Q, S & 7 :. put low-order triad at destination address

IF R > 1 :. if there are remaining chars in the chunk

INC Q

POKE Q, S & 56 / 8 :. put high-order triad at (next) destination address

DEC R :. decrement for high-order triad (decrement for low-order happens 8 lines down)

ENDIF

ELSE :. sextet-stream or octet-stream encoding

POKE Q, H & 3 \* 64 + S :. for octet, bottom 2 bits of holding sextet become top bits of destination byte

H = H / 4 :. shift bits of the holding sextet, in preparation to decode next byte

ENDIF

INC Q :. increment to next destination address

ENDIF

DEC R :. decrement remaining chars (considering it just processed one)

IF R < 1 :. done with this chunk, get ready for next

E = 5 :. set encoding to header encoding

C = -1 :. so cycle count will become 0 once incremented

R = 1 :. set to 1 because < 1 would signal prog to stop decoding altogether

ENDIF

ENDIF

C = (C + 1) MOD ABS(E - 3 + (E = 3)) :. increment cycle count

INC P :. point to next character in data source string D$(I)

UNTIL R < 1 :. chunk with R = 0 tells decoder it is done

. =====================

. GAME REPLAY LOOP

. =====================

. Now that one-time setup is complete, perform initialization for each game

. Main concepts of game code

. --------------------------

. "Map" contains entire world (surface and underground); conceptually 27 x 65,

. but implemented as one-dimensional array of 1755 cells

. Each map cell contains id of an "entity"

. Entity is anything that can appear on the map (except player) and/or in

. inventory

. Entities are all one of five types, some common (can appear in multiple map

. cells), some unique (can only be in one map cell):

. - Passible (common): open areas, grass, forest, etc.

. - Impassible (common): walls, mountains, water

. - Item (unique): taken into inventory when stepped on

. - Obstacle (unique): needs appropriate item to defeat

. - Link (unique): warps to another location on map

. "Entity catalog" has 4 attributes of each of the 52 entities:

. - "Requirement" entity id: for obstacles, this is entity id of item that

. defeats it; impassibles have their own entity id (which can never be

. in inventory); other types identified by pseudo-entities 61 (link),

. 62 (item), or 63 (passable)

. - "Transition" entity id: entity that replaces this one when obstacle is

. defeated or item is taken; for links this is entity id it links to

. - Appearance (ATASCII char code): in graphics 18 implies glyph and color

. - Position (cell# on map): where initially placed, for unique entities only

REPEAT

. Loop through entity ids, reset the inventory of them to empty, and copy the

. entity ids to their default location on the map; start at 8 because 0-7 are

. the common entities that are not inventoried, and are already on the map

FOR I = 8 TO 51

V(I) = 0

POKE M + DPEEK(K + 3 \* I + 1), I

NEXT

. Configure graphics

GRAPHICS 18 :. Text mode having 20 cols x 12 rows, with 4 colors + black background

POKE 756, Y / 256 :. Tells Atari OS to use program's custom character set

Z = DPEEK(88) :. Start address of screen RAM

SETCOLOR 3,0,15 :. Make color 3 white, defaults ok for others (blue, green, burnt orange)

. Display game title and draw frame around map view

POSITION 7,0

? #6,"$D7$C9$DA$C1$D2$C4" :. "WIZARD"

POSITION 8,1

? #6,"$CF$C6 $F7" :. "OF W"

POSITION 10,2

? #6,"$E1$F3$E4" :. " ASD"

COLOR 63 :. Skull glyph, in orange color

PLOT 0,3

DRAWTO 8,3

DRAWTO 8,11

DRAWTO 0,11

DRAWTO 0,3

P = 464 :. Player position on map

T = 999 :. Time (moves) remaining

. ========================

. MAIN GAMEPLAY LOOP

. ========================

REPEAT

EXEC M :. Display map

POSITION 10,4

? #6,"$D4";T;" " :. Display time remaining

GET O :. Get movement key

. Calculate new "trial" position N, using algebraic expression instead of

. chained IFs for brevity. In (AT)ASCII: W=87 A=65 S=83 D=68

N = P + (O = 68) - ( O = 65) + 27 \* ((O = 83) - (O = 87))

E = PEEK(M + N) :. Look up entity at new map position

A = K + 156 + 2 \* E :. Calculate address of entity's "requirement entity id" in entity catalog

R = PEEK(A) :. Get requirement entity id

IF N - P AND E - R :. If new pos different from existing (moving) and destination not impassable

. (Note above uses "-" as shorter alternative to "<>")

X = PEEK(A + 1) :. Get entity's "transition entity id", which follows requirement id

IF R < 61 :. Entity has an inventory requirement, hence an obstacle (or impassable)

IF V(R) :. If required item is in inventory

POKE M + N, R :. Show the item (briefly), indicating it is working to defeat obstacle

V(R) = 0 :. Remove item from inventory

POKE Z + 154 + R, 0 :. Clear item from inventory display

EXEC M :. Refresh map display

SI = 6 :. Sound effect step increment

SD = 0 :. Sound effect step duration

EXEC S :. Play sound effect

POKE M + N, X :. Replace map cell with the transition-to entity

O = 27 \* (E = 45) :. Set exit indicator if winning condition met (i.e. defeated obstacle is wizard)

ENDIF

ELSE :. This is an unobstructed entity (passable, item, or link)

P = N :. Move player to destination cell

IF R = 62 :. This is an "item" -- something that can be taken

V(E) = 1 :. Set slot in inventory array corresponding to this item to true

POKE Z + 154 + E, PEEK(K + E \* 3) :. Look up glyph for item and show on screen

POKE M + P, X :. Replace item on map with its transition-to entity

ELIF R = 61 :. This is a "link" (portal, entryway, stairs, ship, etc.)

EXEC M :. refresh map to show user on top of link cell, before they teleport away

PAUSE 9

P = DPEEK(K + 3 \* X + 1) :. Look up map address of the destination link, and set position to it

ENDIF

DEC T :. Decrement timer

O = 27 \* (T < 0) :. Sets O to 27 (exit indicator) if out of time

ENDIF

ENDIF

UNTIL O = 27 :. User pressed Escape, or game was won or lost

. Play the final sound. If E is 45 (Blue Wizard) then player won, and the

. sound increment will be set to (positive) 6, causing the fanfare sound to

. ascend in pitch. Otherwise (user lost by running out of time or pressing

. Esc) sound increment will be -6, causing descending fanfare. Either way

. the duration of each note will be 12 "jiffies"

SI = (E = 45) \* 12 - 6

SD = 12

EXEC S :. Play sound effect

. Show user they can start new game by typing "N"

POSITION 10,6

? #6,"$EE$C5$D7"; :. "New"

GET O

UNTIL O - 78 :. 78 is (AT)ASCII for "N"

END

. ===========================

. MAP DISPLAY PROCEDURE

. ===========================

. Refreshes the display window of the world map

. Parameters:

. K - Starting address of entity catalog in RAM

. M - Starting address of world map in RAM

. P - Current player position (cell #) on map

. Z - Starting address of screen (display) RAM

PROC M

FOR R = 0 TO 6 :. map display is 7 rows tall

FOR C = 0 TO 6 :. map display is 7 columns wide

G = 206 :. character code for player gylph in color white

IF C - 3 OR R - 3 :. this is not spot where the player glyph always appears, so get map glyph instead

G = PEEK(K + PEEK(M + P - 84 + 27 \* R + C) \* 3) :. look up entity in map cell, then glyph for entity

ENDIF

POKE Z + 81 + 20 \* R + C, G :. put glyph on screen

NEXT

NEXT

ENDPROC

. ============================

. SOUND EFFECT PROCEDURE

. ============================

. Plays an ascending or descending sound.

. Parameters:

. SI - Sound increment; amount pitch rises (or falls if negative) at each step

. SD - Sound duration; time each step is played, in "jiffies"

PROC S

J = 80

FOR I = 0 TO 8

J = J - SI

SOUND 2,J,10,6

PAUSE SD

NEXT I

SOUND

ENDPROC