+ # $ **Bago Game**

This is a Windows 3.0 implementation of one of my favorite games, Big Boggle from Parker Brothers. Pick a topic below if you need help:

More advanced topics:

+ top:000
# main_index
$ Bago Documentation

+ $ # **Rules of the Game**

The object is to make as many words as possible by following a continuous path through adjacent letters in the rack. (Diagonal counts as adjacent.) The path need not be straight, but each letter in the path can only be used once. Only words of 4 or more letters may be used. If one can form several words by the addition of suffixes, all the words will score. For instance, if you find "RATE," "RATES," and "RATED" in the rack, you may include them all in your list. Each round is usually 3 minutes long.

{bml rack.bmp}This rack contains BEER, BEERS, COIN, DARE, DARES, DATE, DATER, DATES, DEAR, DEER, DOER, DOES, DOSE, PIER, PIERS, RATE, RATED, RATES, RATS, READ, READS, REED, REEDS, REIN, RESET, RESETS, RISE, RISER, ROSE, SEAR, SEARED, SEAT, SEATS, SEED, SEEDS, SEER, SETS, SOON, STAR, STARS, STEER, STEERS, TARS, TEAR, TEARS, TEES, TERSE, and probably many more words.

At the end of the round, word duplications between players are crossed out and do not score. The remaining words are scored as follows:

4	5	6	7	8+
1	2	3	5	11

+ top:010
$ Rules of the Game
# rules

+ # $ **Installation**

This program runs under Microsoft Windows 3.0 or later.

There are 3 files used for the game:

the main program

a dictionary of words in ASCII format

the help file you are now reading.

Copy all 3 files to a directory, then run **BAGO.EXE**. You'll probably want to create a program item for BAGO, using the program manager.

+ top:015
# install
$ Installing the Program

+ # $ **How to Play**

To start a round, click on the hourglass window.

Next, just start typing any words you find (see the rules), following each by a return. The words will appear on your list.

If you are using a mouse, you may click on the letter cubes directly. If you accidentally click the wrong letter, just click on it again, and it will be unselected.
{bml enterup.bmp} The enter button can be used to end your words.

At the end of the round, BAGO disqualifies duplicate and illegal words between your list and its own with the following codes:

(indent)	Computer also found this word
"	You listed this word more than once.
<	Your word is less than 4 letters.
?	Your word does not appear in the rack. Check again.
-	This is not a word.

The remaining words are scored as described under Rules of the game.

When the game is over, you may click on any word in either list to see how it could be formed on the board. Use the TAB key to switch between lists.

If you win the game, BAGO will put up a congratulations window. Click on the window (or type any key) to make it go away.

Note: use the {bmc stopup.bmp} button if you wish to end the round immediately.

+ top:020
# running
$ How to Play

+ # $ **Commands**

Game
- New Game
- End Round
- Statistics
- Clear Statistics

Edit
- Load Rack
- Save Rack
- Set Rack
- Quit

Dictionary
- Load
- Save
- Show
- Optimize
- Cull Infrequent
- Reset Frequencies

Options
- Difficulty
- Sound
- Game Duration

About...
- Timed Game
- Learn Words
- Computer Plays
- Rotatable Cubes

Help
- Reference Card
- Index

+ top:030
# commands
$ Commands

+ # $ **New Game**

{bml eggtimer.bmp}Click on the hourglass to start a new game. Or select New Game under the Game menu popup. Typing ^N will do the same thing.

+ com:010
# game_newgame
$ New Game

+ # $ **End Round**

{bml stopup.bmp}Click on the stop sign to stop the current round and score it. Use this if you want to stop before the timer runs out. This is also the only way of ending a round if the Timed Game option is disabled. The End Round selection under the Game menu popup does the same thing. ^E will end the round, too.

+ com:020
# game_endround
$ End Round

[+] [#] [$] **Statistics**

Displays the number of games played and a running score. If you have played a game, a suggestion is given as to what level of difficulty would be challenging to you. This suggestion will be pretty wild unless the difficulty is already set near your playing level, and you have played several games.

+ com:030
# game_statistics
$ Statistics

[+] [#] [$] Clear Statistics

Clears the running score and number of games played.

+ com:040
# game_clearstatistics
$ Clear Statistics

[+] [#] [$] **Load Rack**

Loads a Rack from a file. The first 25 alphabetic characters in the file will be loaded into the rack, starting with the top row, left to right. Case is ignored. For example, the following two files contain equivalent data:

File 1:
ABCDE
FGHIJ
KLMNO
PQRST
UVWXY

File 2:
Abcde-FGHI, jklmn. OPQRSTUVWXYZILOVENURIA

This command is a good way to input Boggle racks that appeared in real life games. You can also input a file saved by the Save Rack command.

Note that this loads a rack for use on the *next* New Game command; you will not see the new rack until you start the next game.

+ com:050
# game_loadrack
$ Load Rack

+ # $ **Save Rack**

Saves the rack being displayed into a file. Your list of words, and the computer's list of words, if any, is also saved. This file is suitable for input to Load Rack.

+ com:060
# game_saverack
$ Save Rack

[+] [#] [$] **Set Rack**

Will prompt for a rack number. The *next* time New Game is selected, that rack number will be used. This is useful if you want to compare your results with a friend's, or replay a certain rack in the future. Note: the rack number of a game, if any, is displayed on the title bar.

+ com:070
# game_tournament
$ Tournament

+ # $ **Quit**

Quits the program. Your options are saved in **BAGO.INI**, and will be in effect the next time you play.,

---

+ com:080
# game_quit
$ Quit

[+] [#] [$] **Difficulty**

Sets the level of difficulty when competing against the computer. The racks remain the same, but the computer can be set to play less aggressively. Here are some sample settings for a dictionary size of 500, and game length of 3 minutes:

Smartness = 5	Beginners
Smartness = 30	Challenging for established players
Smartness = 100	Computer virtually invincible

+ com:090
# opt_difficulty
$ Difficulty

+ # $ **Sound**

If you find the end-of-round bells annoying, you can use this command to turn them off.

+ com:100
# opt_sound
$ Sound

[+] [#] [$] Game Duration

The length of a round an be set from 30 to 600 seconds. Note that you can also use the Timed Game option to turn off the timer completely.

+ com:110
# opt_gameduration
$ Game Duration

+ # $ **Timed Game**

The default is to have a timed game, but you can turn off the timer, allowing yourself as much time as you want for each round.

+ com:120
# opt_timedgame
$ Timed Game

+ # $ **Learn Words**

If this mode is selected, the computer will learn new words from you as you play.

When you put a word on your list that is not in the dictionary, the computer will ask you whether the new word should be added to the dictionary. BAGO will try to strip any suffixes that it thinks are on the word, and sometimes fails. For instance, if your word was 'WATER,' BAGO might try to add a root word of 'WATE'. Answer NO to when prompted if a bogus word is created.

Note that BAGO does not automatically know what suffixes are proper for the word, so you will have to use the dictionary edit command with the VIRGIN button to manually pick appropriate suffixes. Also, remember that only the dictionary in memory is affected; you need to do a dictionary save to add the new words learned to disk.

+ com:130
# opt_learnwords
$ Learn Words

[+] [#] [$] **Computer Plays**

If this is disabled, you will play a solitaire game. Otherwise, BAGO plays against you.

+ com:140
# opt_computerplays
$ Computer Plays

[+] [#] [$] **Rotatable Cubes**

If this option is enabled, the cubes will appear just like they do in a real game: upright, upside-down, and turned 90 degrees either way. This is a good if you are training for the real game, as it strengthens your pattern recognition.

+ com:150
# opt_rotatablecubes
$ Rotatable cubes

+ # $ **Load**

Reloads the dictionary from the file **BAGO.DIC**. The current dictionary in memory, if any, is replaced.

+ com:160
# dict_load
$ Load Dictionary

[+] [#] [$] **Save**

Saves the dictionary in memory to the file **BAGO.DIC**.

+ com:170
# dict_save
$ Save Dictionary

[+] [#] [$] **Show**

A leftover from debugging which will probably disappear in the future. This command writes the entire dictionary into the computer play window, in alphabetical order. To clear out the window, disable Computer Play, then enable it again.

+ com:180
# dict_show
$ Show Dictionary

[+] [#] [$] Edit

This advanced command allows you to edit the dictionary. This is fun, but most players will not need to do this. The fields that appear in the dialog box are as follows:

Root word:
Enter the word you wish to work with. If the word is not already in the dictionary, BAGO asks if you wish to add it.

OK
Use this to confirm your choice of root word.

Exit
This exits dictionary editing mode.

DELETE
Deletes the root word (and all of its suffixes) from the dictionary. Good for getting rid of accidentally entered bogus words.

Prev
Moves to the word directly before the currently displayed word, in alphabetical order.

Next
Moves to the next word, in alphabetical order.

Virgin
Moves to a word in the dictionary that has never been examined. This is a quick way to get to all the words that need your help with suffix assignment.

Frequency:
Displays the count of how often the word (and its forms) came up in actual play. If a new value is typed, this is entered into the dictionary.

Which of the following are words?
BAGO knows some primitive syntactical rules for adding suffixes to a root word, but often fails, as English is a language of exceptions. Check off only the boxes which are legitimate words. If a word does not appear correctly in one of the boxes, the suffixed word can be added explicitly.

Remember that any changes you make only affect the dictionary in memory; you must save the dictionary to disk to make the changes permanent.

+ com:190
# dict_edit
$ Edit Dictionary

[+] [#] [$] Optimize

Bago maintains a count of how many times a word is used. Obviously, a word such as "TIRE" will be found much more frequently than "WALLAROO". Optimizing helps the program to look at commonly used words first.

The words most commonly found are placed near the top of the dictionary's binary tree. This improves the Bago's speed at finding words when competing against a human player.

Words which have the same usage frequency are added to the tree in such a fashion that the tree will be approximately balanced. This will minimize the search path for any given word, improving processing of a human player's words, and dictionary editing. The balancing of the tree also makes it unlikely that the recursive processing routines will overflow the stack from deep levels of recursion.

In cases of a huge dictionary that has a lot of words with the same frequency, the optimize function might recurse very deeply and overflow the system stack. This will manifest itself as the program hanging or going nuts. An inelegant but effective workaround is to break the **BAGO.DIC** file into several pieces using any text editor, then optimize each of the pieces individually. Finally the optimized pieces can be concatenated into one file, and that file optimized.

+ com:200
# dict_optimize
$ Optimize Dictionary

[+] [#] [$] Cull Infrequent

Bago keeps a count of how often a dictionary word was found in actual play. When the dictionary becomes too large, you can use the Cull command to eliminate infrequently occurring words. Specify the minimum frequency required for words to keep. For example, to eliminate all words that did not appear in play 3 or more times, specify 3 as the minimum frequency.

Note that this command takes **BAGO.DIC** (on disk) as input, *not* the dictionary in memory. Because of this, you can Cull a dictionary (or several concatenated dictionaries) which is too large to fit in memory. Results are written to **BAGO.NEW**, which you can copy back to **BAGO.DIC** when you are satisfied.

+ com:210
# dict_cullinfrequent
$ Cull Infrequent

[+] [#] [$] **Reset Freqencies**

Resets all the word count frequencies to zero. Use this if you want to start over and have BAGO collect word frequencies from ground zero. See the technical discussion for a description of word frequencies. Remember to do a save dictionary to make the new frequencies permanent.

+ com:220
# dict_resetfrequencies
$ Reset Dictionary Frequencies

[+] [#] [$] **Reference Card**

Displays a quick reference card showing the scoring values, and the word disqualification codes used by BAGO. For most players, this is all that will ever be needed.

+ com:225
# help_refcard
$ Reference Card

+ # $ **About...**

Shows copyright notice and version number.

+ com:230
# help_about
$ About Bago

+ # $ **Index...**

This menu pick brings up the help system which you are now using.

+ com:240
# help_doc
$ Index

+ # $ **Technical Discussion**

The Dictionary

The dictionary file, **BAGO.DIC**, is in ASCII format for your convenience. Each line of the dictionary consists of a root word, its frequency, and some flags indicating valid suffixes. BAGO appends suffixes on the fly, so that the entry for "MATE" will encompass "MATES," "MATING," "MATED," and so forth.

You can manually create a dictionary with any text editor if you wish. The format is fairly liberal. Just make sure that there is only one word per line, and that words are no longer than 10 characters. You do not need to include frequency and suffix information - BAGO will assume default values. In general, however, the dictionary edit command should be used from within BAGO.

Word frequencies keep track of which words are most common. Each time you or BAGO finds a particular word in a rack, that word's frequency is incremented. It soon becomes clear that some words such as "NOTE" will be much more common than words such as "WALLAROO". The uncommon words can be culled out of the dictionary to free up space, without much penalty to BAGO's playing ability.

When BAGO starts up, it loads the entire dictionary (or as much as will fit) into a data structure in memory. Subsequent operations to the dictionary, such as adding new words, only affect the data structure in memory. If you quit without saving the dictionary, any changes made during the game will be lost.

The dictionary, as shipped, contains several hundred words with acceptable suffixes already defined. It was derived heuristically from the playing pattern of essentially only one person (H. Geo. Wrekshun). Certainly there must be a better set of words. If you find one, I'd be grateful if you sent it to me at my address.

Word Search Algorithm

BAGO uses a recursive algorithm to look at all possible words in the rack. There are many possible words - on the order of 0.25 * 25! (factorial), so the search is aborted on paths that are clearly dead ends. The algorithm is overkill; a 386 computer has time to exhaustively search a large (100,000) list of words during a typical game. BAGO was written to use a very small dictionary (a few hundred words) and still play well.

BAGO is subject to the same time limitations as you are; if the timer runs out, BAGO will stop searching for words. I think you will find the computer to be a formidable opponent, especially if the game timer is set to a very short period of time. I have never seen a normally played game where BAGO did not have enough time to exhaustively search its entire dictionary. With a dictionary size of 800 words, the computer usually finds about 50 words before I even find one. However, a human has a huge vocabulary, and can still win by finding long, obscure words that BAGO does not have in its dictionary.

+ top:040
# technical
$ Technical discussion

[+] [#] [$] **Things which do not work yet**

The rack was designed for VGA appearance, and looks ugly on any monitor without square pixels. Turning off the rotatable cubes option may help.

Pictorals are only displayed in 16 colors for generality. They do not look very good on anything except color VGA.

A combination of mouse pick on cubes and keyboard input confuses BAGO. Please use only one mode for any given word. Experienced players tend to find that the keyboard is faster.

Optimizing a dictionary that is over about 500 words, and alphabetized, causes recursion 500 levels deep, killing the program. Split your large dictionaries into pieces which are small, optimize the small pieces separately, then concatenate the optimized pieces and run a final optimize. If the dictionary is not alphabetized, this problem will generally not occur.

+ top:050
# bugs
$ Things which do not work yet

+ # $ **Author's remarks and address**

I most heartily release BAGO (including the source code) as freeware; that is, you may copy it freely or even modify it, as long as you don't charge anyone for the game or its derivatives. If you should make your own version, I request, but do not require, that you credit "H. G. Wrekshun" for the original program. Please post BAGO on bulletin boards and give it to your friends; I'll get better feedback as more people play it.

To get the source, send me a disk and self-addressed mailer with enough postage to get back to you. My development environment is the SDK with QuickC 2.5 / C 5.1. If any of you out there port BAGO to MacIntosh, X, or something else, I'd be interested in your results.

While I think most of the bugs have been ironed out of this game, I make no guarantees that this program will function as expected, or even reasonably. Run at your own risk!

If you can generate a dictionary which plays the game better than the one I've included, I'd love to have a copy of it.

I'm also looking for suggestions on improving the game, or its documentation.

If you find a rack with a huge number of words, please save it into a file and send it to me. (A printed copy of the file will be cheaper to send than a floppy disk.) The record at this time is about 125 words. Send me mail; I love mail. I'll be **very** grateful for your responses.

Best Regards: Hugh.

H.G. Wrekshun
430 Morse Ave
Sunnyvale, CA 94086-4331

also ryoung@pollux.hp.com

p.s. As much as I like this program, it's still basically playing with yourself. The real Boggle game, played against human competitors, is much more fun. Try it sometime!

+ top:060
# author
$ Author's address

[#] H. G. Wrekshun
430 Morse Ave
Sunnyvale CA 94086-4331
USA

also ryoung@pollux.hp.com

address

[#] {bml face.bmp}An undocumented option you can find if you are clever.

---

pictorals

[#] Virgin words in the dictionary are those which have never been manually examined, and thus have no suffix information.

virgin