A Computer Analysis of Boggle™

Craig S. Kaplan[*]

May 14, 2001

Abstract

Boggle is a fast-paced word search game played on a five-by-five grid of letters. To remove any trace of fun from the game, I have conducted an extensive analysis of Boggle, using a newly-constructed software tool. I present the tool and the Boggle insights it provides.

1 Introduction

In the game of Boggle[1] [4], players are given three minutes to find all the words they can in a five-by-five grid of letters. The letters are printed in random combinations on twenty-five dice, and a new board is generated by shaking the dice inside a specially-designed holder. Words are scored by length, though no points are received for a word written down by more than one player. A legal word is one of at least four letters that can be formed from a sequence of adjacent dice on the board, each die being used at most once. Note that the letter 'Q' does not appear alone on a die face. To increase its usefulness, it appears together with 'U', and the digram is in fact counted as two letters for scoring purposes.

Frequently, a round of Boggle will end with the frustrating feeling that the board was not mined to its full capacity. I am left wanting more time to stare at the board, knowing that strange, surprising and wonderful words are hidden in that jumble of letters; words that, if revealed, might propel me further along the path to true Boggle mastery. Inspired by this belief, I constructed a computer Boggle player. The player is based on a core algorithm that takes a Boggle board and a word list, and determines which words from the list appear on the board. Here, I present the algorithm, pay lip service to complexity theory, and discuss some useful Boggle insights I obtained through statistical analysis of randomly-generated boards.

1.1 Related work

Boggle has been studied extensively by researchers and hobbyists [3]. Problems related to board construction and analysis have also been featured in various computer science courses. Boyan [1] used machine learning to try to find a board with as high a score as possible, though his system is not constrained to the space of boards expressible with the standard dice. Some researchers have also studied the "inverse Boggle problem" [5], which, given a list of words, asks for a board on which all those words appear.

[*]`csk@cs.washington.edu`

[1]This work is actually based on Boggle Deluxe, a variant of the original four-by-four Boggle.

G	S	R	E	G
N	E	S	N	T
O	I	T	U	S
R	R	A	T	S
R	C	E	Qu	P

(a) 1181 points

R	E	T	T	R
S	Y	F	M	D
F	C	G	N	R
I	C	W	D	D
E	E	T	Qu	N

(b) 0 points

R	S	T	C	S
D	E	I	A	E
G	N	L	R	P
E	A	T	E	S
M	S	S	I	D

(c) 3271 points

Figure 1: Three extreme Boggle boards.

2 Algorithm and implementation

There are finitely many legal paths through a Boggle board. Some of these paths will correspond to legal words and others will not. By preprocessing an English dictionary to construct a constant-query-time data structure such as a hash table, we can take a given arrangement of the dice, form the letter sequence generated by each path, and check in constant time whether that sequence is a word. Because the number of possible paths is an *a priori* constant (certainly less than $25! \approx 1.55 \times 10^{25}$), the total time necessary for computing the score of a board is $\mathrm{O}(1)$.

My implementation is exactly unlike the algorithm described above. I iterate over the word list, testing each word in turn to see whether it appears on the board. I perform a depth-first search starting on every cell in the board until I find the word or establish that it cannot be found. As an optimization, I construct bit fields of letters and digrams that appear, using these bit fields to eliminate many impossible words without having to do a search. In theory, a faster algorithm would turn the word list into a trie data structure, halting the search for a word when a prefix of that word is not found. Yet even without intelligent algorithmic optimization, the implementation is quite fast, finding all possible words (from `/usr/dict/words`) on a Boggle board in about a hundredth of a second.

3 The endlessly fascinating world of Boggle

In this section, I report on the many scintillating Boggle facts I uncovered using my implementation. To collect information, I created a test program that generates random boards (using correct Boggle dice), evaluating total scores and tracking occurences of every word in the list. The test program has a hard-coded word list. A separate program reads words from standard input, allowing for dictionaries other than `/usr/dict/words`. Most of the statistics given below are derived from a run of the test program over a random sampling of one million Boggle boards.

3.1 Total board score

The total board score is the total value of all words that appear in a given board. On average, the total board score of an arbitrary Boggle board is about 136 with a standard deviation of 81. During a round of Boggle, this information could help in deciding how much effort to expend in hunting for more words.

Of course, extreme cases are also interesting. In my sample of one million boards, the highest total board score found was 1181. The search also yielded a board with a score of zero[2]. The boards that yield

[2]When run with the Scrabble™ Dictionary, the same board yields 18 points, using words such as “yett” and “ngwee”.

those scores are shown in Figure 1. Boyan [1] has found a board with an amazing total score of 3271, but again that board is not realizable using the standard Boggle dice.

3.2 Words to live by

When playing Boggle competitively, it would be useful to know which words occur most and least frequently in practice. For example, since there is no penalty for writing down a word that does not appear, one could memorize a list of the most frequently-occuring words and write them down mechanically at the start of every round, all the while searching the board for less common words. Perhaps one could even have them engraved onto a rubber stamp.

In the million-board trial, the twenty-five most frequently-occuring words, along with the percentage of boards in which they appear, are: toes (11%), eats (11%), sate (11%), earn (11.1%), seen (11.1%), nets (11.2%), teas (11.3%), seat (11.4%), seer (11.5%), rein (11.5%), tone (11.7%), near (11.7%), note (11.8%), tire (11.8%), tore (11.9%), rite (11.9%), tent (12%), tier (12.5%), ante (12.9%), rent (13.2%), rate (13.3%), neat (13.4%), tree (13.7%), tear (13.7%), and teen (14.5%).

Many words, even with as few as four letters, never appear. That is because they could never be realized on any Boggle board. “Baby”, for example, is impossible because only one of the the twenty-five dice contains a ‘B’. “Back”, too, cannot appear because the same die contains the only ‘K’. The general question of whether a word can be formed from the Boggle dice is in fact an instance of the bipartite matching problem [2, pages 600–604].

3.3 The attractiveness of sesquipedality

The discovery of a particularly long word in a round of Boggle can confer a special joy upon the player lucky enough to find it. Plenty of long words are waiting to be found in the average Boggle board, although occurences of four-letter words outnumber all longer words put together in the million boards tested. In my testing, 258 12-letter words came up, 45 13-letter words (from “acquaintances” to “mercenariness”), and one whopping 14-letter word, “preconditioned” (which really should be worth more than eleven points).

4 Conclusion

Although I have successfully probed some of the mysteries of this ancient and venerated game, one important question still remains: will a computer Boggle player ever defeat the human champion? This challenge I leave in the hands of the Artificial Intelligence community.

References

[1] J. A. Boyan. *Learning Evaluation Functions for Global Optimization*. PhD thesis, Carnegie Mellon University, 1998.

[2] Thomas H. Cormen, Charles E. Leieserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1992.

[3] The Wordgame Programmers Egroup. Boggle - source code. `http://www.gtoal.com/wordgames/boggle.html`.

[4] Hasbro Incorporated. Boggle. `http://www.boggle.com/`.

[5] Problem of the Month. The inverse Boggle problem. `http://members.tripod.com/˜POTM/BOGGLE/problem.long.html`.