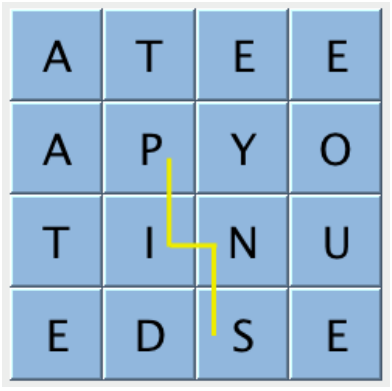
Boggle

Write a program to play the word game Boggle®.

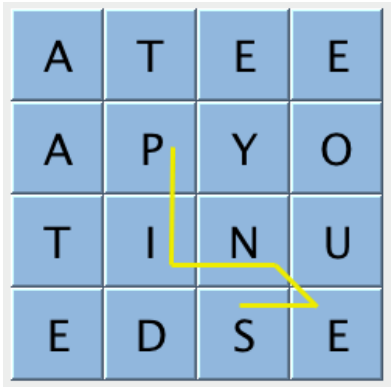
Boggle is a word game designed by Allan Turoff and distributed by Hasbro. It involves a board made up of 16 cubic dice, where each die has a letter printed on each of its sides. At the beginning of the game, the 16 dice are shaken and randomly distributed into a 4-by-4 tray, with only the top sides of the dice visible. The players compete to accumulate points by building valid words out of the dice according to the following rules:

- A valid word must be composed by following a sequence of *adjacent dice*—two dice are adjacent if they are horizontal, vertical, or diagonal neighbors.
- A valid word can use each die at most once.
- A valid word must contain at least 3 letters.
- A valid word must be in the dictionary (which typically does not contain proper nouns).

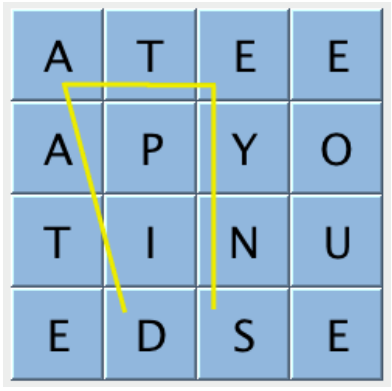
Here are some examples of valid and invalid words:



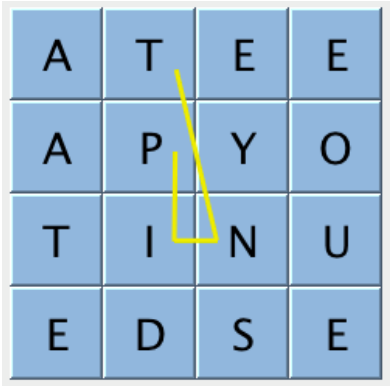
PINS
(valid)



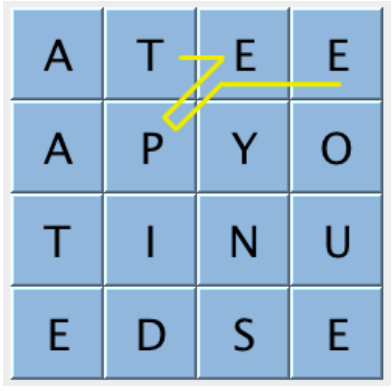
PINES
(valid)



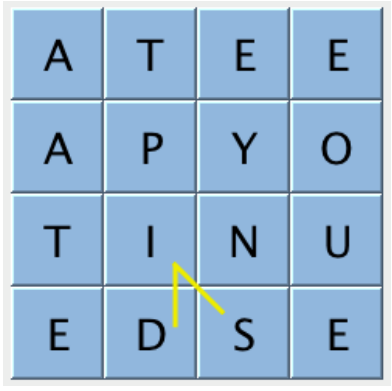
DATES
(invalid—dice not adjacent)



PINT
(invalid—path not sequential)



TEPEE
(invalid—die used more than once)

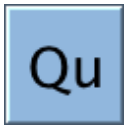


SID
(invalid—word not in dictionary)

Scoring. Words are scored according to their length, using this table:

word length	*points*
0–2	0
3–4	1
5	2
6	3
7	5
8+	11

The* Qu *special case. In the English language, the letter `Q` is almost always followed by the letter `U`. Consequently, the side of one die is printed with the two-letter sequence `Qu` instead of `Q` (and this two-letter sequence must be used together when forming words). When scoring, `Qu` counts as two letters; for example, the word `QuEUE` scores as a 5-letter word even though it is formed by following a sequence of 4 dice.



Your task. Your challenge is to write a Boggle solver that finds all valid words in a given Boggle board, using a given dictionary. Implement an immutable data type `BoggleSolver` with the following public API:

```
public class BoggleSolver
{
    // Initializes the data structure using the given dictionary file name String.
    //   (You can assume each word in the dictionary contains only the uppercase letters A - Z.)
    public BoggleSolver(String dictionaryFileName)

    // Returns the set of all valid words in the given Boggle board, as an Iterable object
    //   All collections classes implement the Iterable interface, e.g. List/ArrayList, Set/HashSet
    //   Basically, any standard Java collection is-a Iterable (used for for-each loops)
    public Iterable<String> getAllValidWords(BoggleBoard board)

    // Returns the score of the given word if it is in the dictionary, zero otherwise.
    //   (You can assume the word contains only the uppercase letters A - Z.)
    public int scoreOf(String word)
}
```

When searching for words contained on the board (using a dictionary to check if they're valid), use a Hash Set. A Hash Set (implemented in `java.util.HashSet`) is a data structure that doesn't allow duplicates (useful for filtering out duplicate words found on the board) and has O(1) *contains* (useful for the dictionary lookups).

The board data type. To save you some time, you are provided an immutable data type BoggleBoard.java for representing Boggle boards. It includes constructors for creating Boggle boards using the actual Hasbro dice and the distribution of letters in the English language, a file, or a character array. It contains methods for accessing the individual letters, and a method to print out the board for debugging. Here is the full API:

```
public class BoggleBoard
{
    // Initializes a random 4-by-4 Boggle board.
    //   (by rolling the Hasbro dice)
    public BoggleBoard()

    // Initializes a random m-by-n Boggle board.
    //   (using the frequency of letters in the English language)
    public BoggleBoard(int m, int n)

    // Initializes a Boggle board from the specified filename.
    public BoggleBoard(String filename)

    // Initializes a Boggle board from the 2d char array.
    //   (with 'Q' representing the two-letter sequence "Qu")
    public BoggleBoard(char[][] a)

    // Returns the number of rows.
    public int rows()

    // Returns the number of columns.
    public int cols()

    // Returns the letter in row r and column c.
    //   (with 'Q' representing the two-letter sequence "Qu")
    public char getLetter(int r, int c)

    // Returns a string representation of the board.
    public String toString()
}
```

***Testing*.** You are provided several dictionary and board files for testing. When importing the "data" folder, don't drag it into the "src" folder – drag it into the root directory (the main project folder).

- *Dictionaries.* A dictionary consists of a sequence of words, separated by whitespace, in alphabetical order. You can assume that each word contains only the uppercase letters `A` through `Z`. For example, here are the two files "dictionary-algs4.txt" and "dictionary-yawl.txt":

```
% dictionary-algs4.txt
ABACUS
ABANDON
ABANDONED
ABBREVIATE
...
QUEUE
...
ZOOLOGY
```

```
% dictionary-yawl.txt
AA
AAH
AAHED
AAHING
...
PNEUMONOULTRAMICROSCOPICSILICOVOLCANOCONIOSIS
...
ZYZZYVAS
```

  The former is a list of 6,013 words that appear in Algorithms 4th edition; the latter is a comprehensive list of 264,061 English words (known as "Yet Another Word List") that is widely used in word-game competitions.

- *Boggle boards.* A boggle board consists of two integers M and N, followed by the $M \times N$ characters in the board, with the integers and characters separated by whitespace. You can assume the integers are nonnegative and that the characters are uppercase letters `A` through `Z` (with the two-letter sequence `Qu` represented as either `Q` or `Qu`). Here are the files "board4x4.txt" and "board-q.txt":

```
% board4x4.txt
4 4
A  T  E  E
A  P  Y  O
T  I  N  U
E  D  S  E
```

```
% board-q.txt
4 4
S  N  R  T
O  I  E  L
E  Qu T  T
R  S  A  T
```

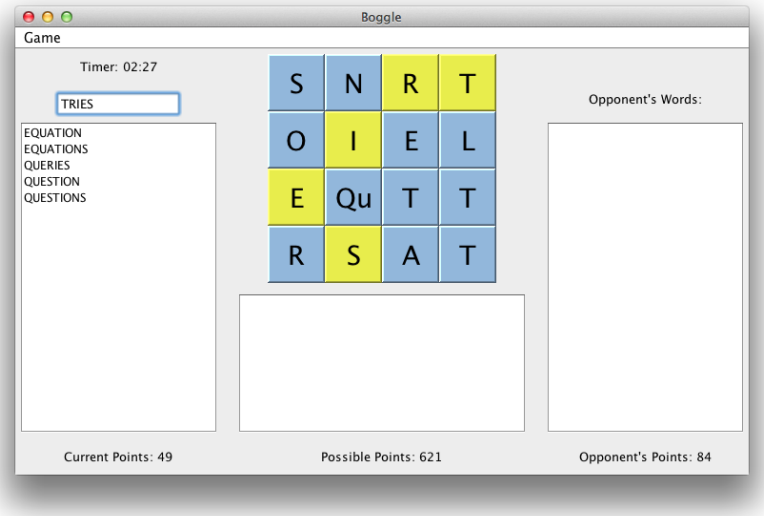
A sample test client has been provided in BoggleSolver's `main` method. Add other tests as you see fit.

Here is the (partial) output, using the dictionary / starter board specified:

```
...
QUESTIONS, points = 11
TIE, points = 1
REST, points = 1
SITS, points = 1
REQUEST, points = 5
TIN, points = 1
SIN, points = 1
LET, points = 1
NET, points = 1
TEN, points = 1
SIT, points = 1
TRIES, points = 2

Score = 84
```

Interactive game. Once you have a working version of `BoggleSolver.java`, compile and run BoggleGame.java to play Boggle against a computer opponent. To enter a word, either type it in the text box or click the corresponding sequence of dice on the board. The computer opponent has various levels of difficulty, ranging from finding only words from popular nursery rhymes (easy) to words that appear in Algorithms 4/e (medium) to finding every valid word (impossible).



(Advanced) Faster solution

Running your solver on a 4x4 board with quite a few words takes a few seconds - what would happen with a 10x10 board?

If you print the current state of `word` as the solver runs, you'll notice that the solver attempts to find words with "prefix" Strings that clearly aren't possible words (e.g. "NSRX"). You can implement a major performance improvement / optimization with a concept called ***pruning***. When the current path corresponds to a string that is not a prefix of any word in the dictionary, there is no need to expand the path further.

To do this, you will need to devise a data structure for the `dictionary` that supports a fast "prefix query" operation: given a prefix, is there any word in the dictionary that starts with that prefix?

(Advanced) Challenges for the bored

Here are some challenges:

- Find a maximum scoring 4-by-4 Hasbro board. Here is the best known board (4540 points), which was discovered by Robert McAnany in connection with this Coursera course.
- Find a maximum scoring 4-by-4 Hasbro board using the Zinga list of 584,983 Italian words.
- Find a minimum scoring 4-by-4 Hasbro board.
- Find a maximum scoring 5-by-5 Deluxe Boggle board.
- Find a maximum scoring *N*-by-*N* board (not necessarily using the Hasbro dice) for different values of *N*.
- Find a board with the most words (or the most words that are 8 letters or longer).
- Find a 4-by-4 Hasbro board that scores *exactly* 2,500, 3,000, 3,500, or 4,000 points.
- Design an algorithm to determine whether a given 4-by-4 board can be generated by rolling the 16 Hasbro dice.
- How many words in the dictionary appear in no 4-by-4 Hasbro boards?
- Add new features to BoggleGame.java.
- Extend your program to handle arbitrary Unicode letters and dictionaries. You may need to consider alternate algorithms and data structures.
- Extend your program to handle arbitrary strings on the faces of the dice, generalizing your hack for dealing with the two-letter sequence `Qu`.

Unless otherwise stated, use the dictionary-yawl.txt dictionary.