Flask Boggle

Download solution code <../../flask-boggle-solution.zip>.

boggle.py

```
"""Utilities related to Boggle game."""

from random import choice
import string


class Boggle():

    def __init__(self):

        self.words = self.read_dict("/usr/share/dict/words")

    def read_dict(self, dict_path):
        """Read and return all words in dictionary."""

        dict_file = open(dict_path)
        words = [w.strip() for w in dict_file]
        dict_file.close()
        return words

    def make_board(self):
        """Make and return a random boggle board."""

        board = []

        for y in range(5):
            row = [choice(string.ascii_uppercase) for i in range(5)]
            board.append(row)

        return board

    def check_valid_word(self, board, word):
        """Check if a word is a valid word in the dictionary and/or the boggle board"""

        word_exists = word in self.words
        valid_word = self.find(board, word.upper())

        if word_exists and valid_word:
            result = "ok"
        elif word_exists and not valid_word:
            result = "not-on-board"
        else:
            result = "not-word"

        return result

    def find_from(self, board, word, y, x, seen):
        """Can we find a word on board, starting at x, y?"""

        if x > 4 or y > 4:
            return
```

```
        # This is called recursively to find smaller and smaller words
        # until all tries are exhausted or until success.

        # Base case: this isn't the letter we're looking for.

        if board[y][x] != word[0]:
            return False

        # Base case: we've used this letter before in this current path

        if (y, x) in seen:
            return False

        # Base case: we are down to the last letter --- so we win!

        if len(word) == 1:
            return True

        # Otherwise, this letter is good, so note that we've seen it,
        # and try of all of its neighbors for the first letter of the
        # rest of the word
        # This next line is a bit tricky: we want to note that we've seen the
        # letter at this location. However, we only want the child calls of this
        # to get that, and if we used `seen.add(...)` to add it to our set,
        # *all* calls would get that, since the set is passed around. That would
        # mean that once we try a letter in one call, it could never be tried again,
        # even in a totally different path. Therefore, we want to create a *new*
        # seen set that is equal to this set plus the new letter. Being a new
        # object, rather than a mutated shared object, calls that don't descend
        # from us won't have this `y,x` point in their seen.
        #
        # To do this, we use the | (set-union) operator, read this line as
        # "rebind seen to the union of the current seen and the set of point(y,x))."
        #
        # (this could be written with an augmented operator as "seen |= {(y, x)}",
        # in the same way "x = x + 2" can be written as "x += 2", but that would seem
        # harder to understand).

        seen = seen | {(y, x)}

        # adding diagonals

        if y > 0:
            if self.find_from(board, word[1:], y - 1, x, seen):
                return True

        if y < 4:
            if self.find_from(board, word[1:], y + 1, x, seen):
                return True

        if x > 0:
            if self.find_from(board, word[1:], y, x - 1, seen):
                return True

        if x < 4:
            if self.find_from(board, word[1:], y, x + 1, seen):
                return True
```

```
        # diagonals
        if y > 0 and x > 0:
            if self.find_from(board, word[1:], y - 1, x - 1, seen):
                return True

        if y < 4 and x < 4:
            if self.find_from(board, word[1:], y + 1, x + 1, seen):
                return True

        if x > 0 and y < 4:
            if self.find_from(board, word[1:], y + 1, x - 1, seen):
                return True

        if x < 4 and y > 0:
            if self.find_from(board, word[1:], y - 1, x + 1, seen):
                return True
        # Couldn't find the next letter, so this path is dead

        return False

    def find(self, board, word):
        """Can word be found in board?"""

        # Find starting letter --- try every spot on board and,
        # win fast, should we find the word at that place.

        for y in range(0, 5):
            for x in range(0, 5):
                if self.find_from(board, word, y, x, seen=set()):
                    return True

        # We've tried every path from every starting square w/o luck.
        # Sad panda.

        return False
```

app.py

```
from flask import Flask, request, render_template, jsonify, session
from boggle import Boggle

app = Flask(__name__)
app.config["SECRET_KEY"] = "fdfgkjtjkkg45yfdb"

boggle_game = Boggle()


@app.route("/")
def homepage():
    """Show board."""

    board = boggle_game.make_board()
    session['board'] = board
    highscore = session.get("highscore", 0)
    nplays = session.get("nplays", 0)

    return render_template("index.html", board=board,
```

```
                           highscore=highscore,
                           nplays=nplays)


@app.route("/check-word")
def check_word():
    """Check if word is in dictionary."""

    word = request.args["word"]
    board = session["board"]
    response = boggle_game.check_valid_word(board, word)

    return jsonify({'result': response})


@app.route("/post-score", methods=["POST"])
def post_score():
    """Receive score, update nplays, update high score if appropriate."""

    score = request.json["score"]
    highscore = session.get("highscore", 0)
    nplays = session.get("nplays", 0)

    session['nplays'] = nplays + 1
    session['highscore'] = max(score, highscore)

    return jsonify(brokeRecord=score > highscore)
```

templates/index.html

```
<!doctype html>
<html>
  <head>
    <title>Boggle</title>
    <link rel="stylesheet" href="/static/boggle.css">
  </head>
  <body>
    <section id="boggle">

      <table class="board">
        <tbody>
          {% for row in board %}
          <tr>
            {% for cell in row %}
            <td>{{ cell }}</td>
            {% endfor %}
          </tr>
          {% endfor %}
        </tbody>
      </table>

      <p>High Score:
        <b>{{ highscore }}</b>
        in {{ nplays }} plays
      </p>

      <p>
        Score: <b class="score">0</b>
```

```
        Seconds Left: <b class="timer"></b>
      </p>

      <form method="POST" class="add-word">
        <input name="word" class="word" autofocus>
        <button>Enter</button>
      </form>

      <p class="msg">
        <!-- our JS will put messages here dynamically -->
      </p>

      <ul class="words">
        <!-- our JS will words here as they score -->
      </ul>

    </section>
    <script src="http://unpkg.com/jquery"></script>
    <script src="https://unpkg.com/axios@0.19.0/dist/axios.js"></script>
    <script src="/static/boggle.js"></script>
    <script>
      let game = new BoggleGame("boggle", 60);
    </script>
  </body>
</html>
```

templates/index.html

```
<!doctype html>
<html>
  <head>
    <title>Boggle</title>
    <link rel="stylesheet" href="/static/boggle.css">
  </head>
  <body>
    <section id="boggle">

      <table class="board">
        <tbody>
          {% for row in board %}
          <tr>
            {% for cell in row %}
            <td>{{ cell }}</td>
            {% endfor %}
          </tr>
          {% endfor %}
        </tbody>
      </table>

      <p>High Score:
        <b>{{ highscore }}</b>
        in {{ nplays }} plays
      </p>

      <p>
        Score: <b class="score">0</b>
        Seconds Left: <b class="timer"></b>
      </p>
```

```
      <form method="POST" class="add-word">
        <input name="word" class="word" autofocus>
        <button>Enter</button>
      </form>

      <p class="msg">
        <!-- our JS will put messages here dynamically -->
      </p>

      <ul class="words">
        <!-- our JS will words here as they score -->
      </ul>

    </section>
    <script src="http://unpkg.com/jquery"></script>
    <script src="https://unpkg.com/axios@0.19.0/dist/axios.js"></script>
    <script src="/static/boggle.js"></script>
    <script>
      let game = new BoggleGame("boggle", 60);
    </script>
  </body>
</html>
```

static/boggle.js

```
class BoggleGame {
  /* make a new game at this DOM id */

  constructor(boardId, secs = 60) {
    this.secs = secs; // game length
    this.showTimer();

    this.score = 0;
    this.words = new Set();
    this.board = $("#" + boardId);

    // every 1000 msec, "tick"
    this.timer = setInterval(this.tick.bind(this), 1000);

    $(".add-word", this.board).on("submit", this.handleSubmit.bind(this));
  }

  /* show word in list of words */

  showWord(word) {
    $(".words", this.board).append($("<li>", { text: word }));
  }

  /* show score in html */

  showScore() {
    $(".score", this.board).text(this.score);
  }

  /* show a status message */

  showMessage(msg, cls) {
```

```
    $(".msg", this.board)
      .text(msg)
      .removeClass()
      .addClass(`msg ${cls}`);
  }

  /* handle submission of word: if unique and valid, score & show */

  async handleSubmit(evt) {
    evt.preventDefault();
    const $word = $(".word", this.board);

    let word = $word.val();
    if (!word) return;

    if (this.words.has(word)) {
      this.showMessage(`Already found ${word}`, "err");
      return;
    }

    // check server for validity
    const resp = await axios.get("/check-word", { params: { word: word }});
    if (resp.data.result === "not-word") {
      this.showMessage(`${word} is not a valid English word`, "err");
    } else if (resp.data.result === "not-on-board") {
      this.showMessage(`${word} is not a valid word on this board`, "err");
    } else {
      this.showWord(word);
      this.score += word.length;
      this.showScore();
      this.words.add(word);
      this.showMessage(`Added: ${word}`, "ok");
    }

    $word.val("").focus();
  }

  /* Update timer in DOM */

  showTimer() {
    $(".timer", this.board).text(this.secs);
  }

  /* Tick: handle a second passing in game */

  async tick() {
    this.secs -= 1;
    this.showTimer();

    if (this.secs === 0) {
      clearInterval(this.timer);
      await this.scoreGame();
    }
  }

  /* end of game: score and update message. */

  async scoreGame() {
```

```
    $(".add-word", this.board).hide();
    const resp = await axios.post("/post-score", { score: this.score });
    if (resp.data.brokeRecord) {
      this.showMessage(`New record: ${this.score}`, "ok");
    } else {
      this.showMessage(`Final score: ${this.score}`, "ok");
    }
  }
}
```

Tests

test.py

```
from unittest import TestCase
from app import app
from flask import session
from boggle import Boggle


class FlaskTests(TestCase):

    def setUp(self):
        """Stuff to do before every test."""

        self.client = app.test_client()
        app.config['TESTING'] = True

    def test_homepage(self):
        """Make sure information is in the session and HTML is displayed"""

        with self.client:
            response = self.client.get('/')
            self.assertIn('board', session)
            self.assertIsNone(session.get('highscore'))
            self.assertIsNone(session.get('nplays'))
            self.assertIn(b'<p>High Score:', response.data)
            self.assertIn(b'Score:', response.data)
            self.assertIn(b'Seconds Left:', response.data)

    def test_valid_word(self):
        """Test if word is valid by modifying the board in the session"""

        with self.client as client:
            with client.session_transaction() as sess:
                sess['board'] = [["C", "A", "T", "T", "T"],
                                 ["C", "A", "T", "T", "T"],
                                 ["C", "A", "T", "T", "T"],
                                 ["C", "A", "T", "T", "T"],
                                 ["C", "A", "T", "T", "T"]]
            response = self.client.get('/check-word?word=cat')
            self.assertEqual(response.json['result'], 'ok')

    def test_invalid_word(self):
        """Test if word is in the dictionary"""
```

```
        self.client.get('/')
        response = self.client.get('/check-word?word=impossible')
        self.assertEqual(response.json['result'], 'not-on-board')

    def non_english_word(self):
        """Test if word is on the board"""

        self.client.get('/')
        response = self.client.get(
            '/check-word?word=fsjdakfkldsfjdslkfjdlksf')
        self.assertEqual(response.json['result'], 'not-word')
```