IG Problem 3A: Fanfiction Ship Names

Written by Ben Yan

```
/* This function takes two one-word names, and smashes them together into
 * a fanfic-ready ship name, according to the vowel / consonant rules above.
 */
function shipThem(nameOne, nameTwo) {
      let oneFirstVowel = findFirstVowel(nameOne);
      let twoFirstVowel = findFirstVowel(nameTwo);

      if (oneFirstVowel > 0) { // starts with a consonant
            return nameOne.substr(, oneFirstVowel)
                        + nameTwo.substr(twoFirstVowel, );

      } else { // starts with a vowel
            oneSecondVowel = findFirstVowel(nameOne, 1);
            return nameOne.substr(, oneSecondVowel)
                        + nameTwo.substr(twoFirstVowel + 1, );
      }
}

/* Returns index of first vowel at or after the start position, and -1 if
 * no vowel could be found. If the start is omitted, it's assumed to be 0.
 */
function findFirstVowel(word, start) {
      if (start === undefined) start = 0;
      for (let index = start; index < word.length; index++) {
            if (isEnglishVowel(word[index])) {
                  return index;
            }
      }
      return -1; // vowel not found :(
}

/* returns true if the string is of length one, and a lowercase vowel */
function isEnglishVowel(ch) {
      return ch.length === 1 && "aeiou".indexOf(ch) >= 0;
}
```

JavaScript Console

```
> shipThem("bruce", "selina")
brelina
> shipThem("megumi", "yuji")
muji
> shipThem("wednesday", "enid")
wenid
> shipThem("armin", "eren")
armren
>
```



IG Problem 3B: Boggle – The Mind-Boggling Word Game

Written by Ben Yan

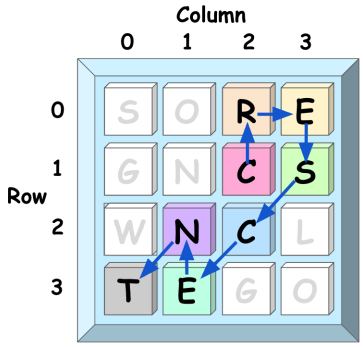
```
/* Returns true if the path of squares traces the given word on the Boggle
 * grid, without violating any of the game conditions, and false otherwise.
 */
function validateWord(grid, word, path) {
      if (word.length != path.length) return false;

      // checks the letters traced by path match the word
      for (let i = 0; i < path.length; i++) {
            let row = path[i][0];
            let col = path[i][1];
            if (grid[row][col] !== word[i]) return false;
      }

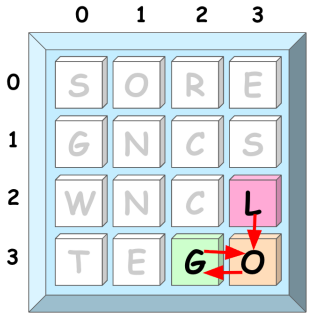
      // checks that each move to next square is valid
      for (let i = 1; i < path.length; i++) {
            if (!moveIsValid(path[i - 1], path[i])) return false;
      }

      // check that no pair of path locations are the same
      for (let i = 0; i < path.length; i++){
            for (let j = i + 1; j < path.length; j++) {
                  let sameRow = path[i][0] == path[j][0];
                  let sameCol = path[i][1] == path[j][1];
                  if (sameRow && sameCol) return false;
            }
      }

      return true; // all requirements satisfied, wahoo!
}
```



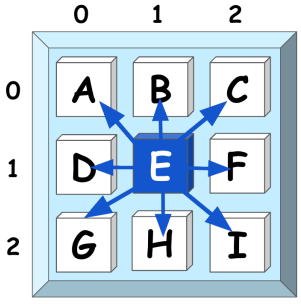
CRESCENT *on grid*



Failed attempt at ***LOGO****, whose path self-intersects*

```
/* Checks that two squares adjoin each other horizontally, vertically, or
 * diagonally. Assumes prev != next, where each is a [row, column] list.
 */
function moveIsValid(prev, next) {
      let neighboringRow = Math.abs(prev[0] - next[0]) <= 1
      let neighboringCol = Math.abs(prev[1] - next[1]) <= 1

      return neighboringRow && neighboringCol;
}
```



Valid next squares from ***[1,1]***