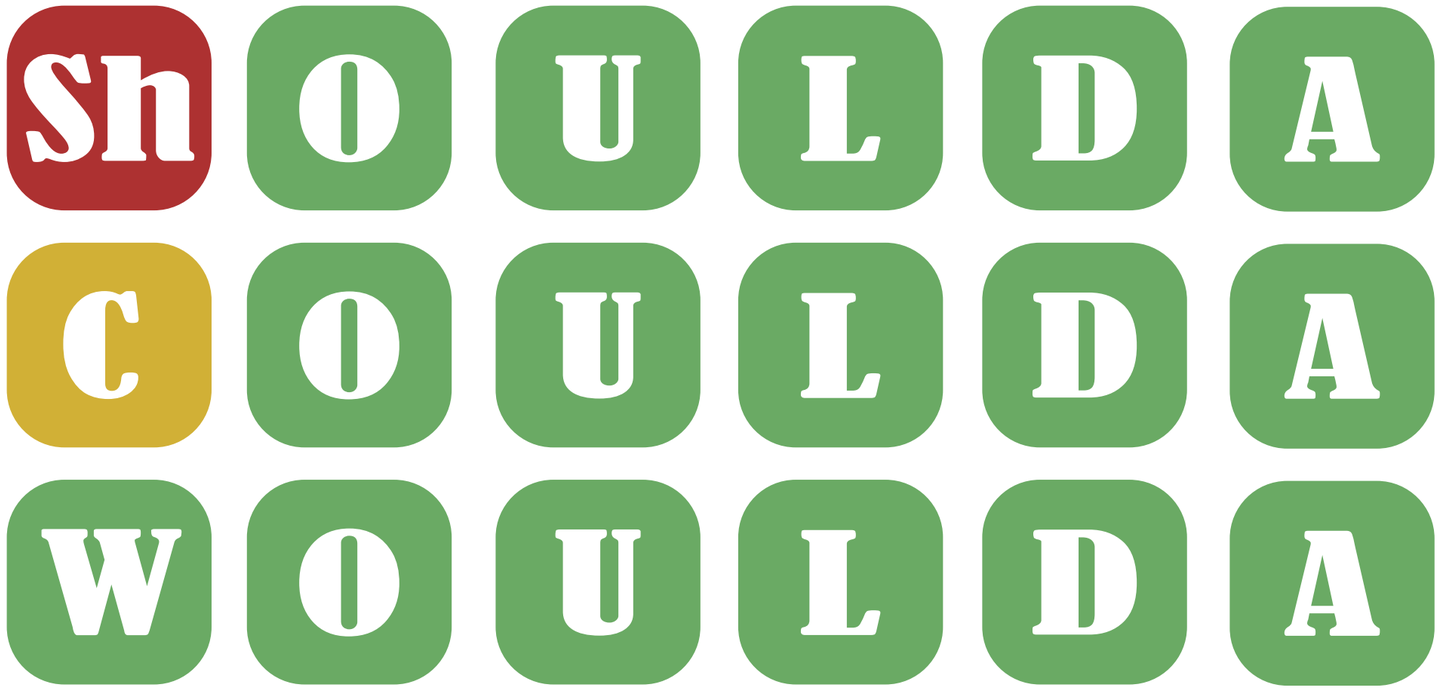

Getting the most out of iXML

Bethan Tovey-Walsh
linguacelta.com

Your scientists were so pre-occupied with whether or not they *could*, they didn't stop to think if they *should.*

Dr. Ian Malcolm
Jurassic Park, 1993

What could(n't) we do?

- some limitations of iXML
- finding ways around the limitations

What should(n't) we do?

- when to use workarounds
- when to hand off tasks

What could(n't) we do?

Could we reproduce the logic of a Wordle puzzle in iXML?

Requirements:

- set a five-letter target word
- assess a guessed word against the target
- determine correctness of the guess
- determine correctness of each guessed character

What could(n't) we do?

Grammar 1:

- determine correctness of the guess

What could(n’t) we do?

PARSE

What could(n't) we do?

```
-word = success; failure.

char1 = "P".
not_char1 = ~["P"].
char2 = "A".
not_char2 = ~["A"].
char3 = "R".
not_char3 = ~["R"].
char4 = "S".
not_char4 = ~["S"].
char5 = "E".
not_char5 = ~["E"].
```

What could(n't) we do?

```
anychar = ["A"-"Z"].

-failure1 = not_char1, anychar, anychar, anychar, anychar.
-failure2 = anychar, not_char2, anychar, anychar, anychar.
-failure3 = anychar, anychar, not_char3, anychar, anychar.
-failure4 = anychar, anychar, anychar, not_char4, anychar.
-failure5 = anychar, anychar, anychar, anychar, not_char5.

success = char1, char2, char3, char4, char5.
failure = failure1; failure2; failure3; failure4; failure5.
```

What could(n't) we do?

```
-word = success; failure.

char1 = "P".
not_char1 = ~["P"].
char2 = "A".
not_char2 = ~["A"].
char3 = "R".
not_char3 = ~["R"].
char4 = "S".
not_char4 = ~["S"].
char5 = "E".
not_char5 = ~["E"].

anychar = ["A"-"Z"].
```

```
-failure1 = not_char1, anychar,
anychar, anychar, anychar.
-failure2 = anychar, not_char2,
anychar, anychar, anychar.
-failure3 = anychar, anychar, not_
char3, anychar, anychar.
-failure4 = anychar, anychar,
anychar, not_char4, anychar.
-failure5 = anychar, anychar,
anychar, anychar, not_char5.

success = char1, char2, char3,
char4, char5.

failure = failure1; failure2;
failure3; failure4; failure5.
```

What could(n't) we do?


```
coffeepot -pp -g:wordlent1.ixml PARSE

<success>
    <char1>P</char1>
    <char2>A</char2>
    <char3>R</char3>
    <char4>S</char4>
    <char5>E</char5>
</success>
```

What could(n't) we do?

What could(n't) we do?

C O U L D

```
coffeepot -pp -g:wordlent1.ixml COULD

<failure xmlns:ixml='http://invisiblexml.org/NS'
ixml:state='ambiguous'>
   <not_char1>C</not_char1>
   <anychar>O</anychar>
   <anychar>U</anychar>
   <anychar>L</anychar>
   <anychar>D</anychar>
</failure>
```

What could(n't) we do?

C O U L D

```
coffeepot --parse-count 5 -pp -g:wordlent1.ixml COULD

<failure xmlns:ixml='http://invisiblexml.org/NS'
ixml:state='ambiguous'>
   <anychar>C</anychar>
   <not_char2>O</not_char2>
   <anychar>U</anychar>
   <anychar>L</anychar>
   <anychar>D</anychar>
</failure>
```

What could(n't) we do?

C O U L D

```
coffeepot --parse-count 5 -pp -g:wordlent1.ixml COULD

<failure xmlns:ixml='http://invisiblexml.org/NS'
ixml:state='ambiguous'>
   <anychar>C</anychar>
   <anychar>O</anychar>
   <not_char3>U</not_char3>
   <anychar>L</anychar>
   <anychar>D</anychar>
</failure>
```

What could(n't) we do?

Grammar 1:

- determine correctness of the guess

Grammar 2:

- determine correctness of the characters

What could(n't) we do?

```
-word = success; failure.

char1 = "P".
not_char1 = ~["P"].
char2 = "A".
not_char2 = ~["A"].
char3 = "R".
not_char3 = ~["R"].
char4 = "S".
not_char4 = ~["S"].
char5 = "E".
not_char5 = ~["E"].

success = char1, char2, char3, char4, char5.
```

What could(n't) we do?

```
failure = char1, failure2to5;
          not_char1, (success2to5; failure2to5).

-failure2to5 = char2, failure3to5;
          not_char2, (success3to5; failure3to5).

-failure3to5 = char3, failure4to5;
          not_char3, (success4to5; failure4to5).

-failure4to5 = not_char4, not_char5;
          char4, not_char5;
          not_char4, char5.

-success2to5 = char2, char3, char4, char5.
-success3to5 = char3, char4, char5.
-success4to5 = char4, char5.
```

What could(n't) we do?


```
coffeepot -pp -g:wordlent2.ixml PARSE

<success>
    <char1>P</char1>
    <char2>A</char2>
    <char3>R</char3>
    <char4>S</char4>
    <char5>E</char5>
</success>
```

What could(n't) we do?

C O U L D

```
coffeepot -pp -g:wordlent2.ixml COULD

<failure>
   <not_char1>C</not_char1>
   <not_char2>0</not_char2>
   <not_char3>U</not_char3>
   <not_char4>L</not_char4>
   <not_char5>D</not_char5>
</failure>
```

What could(n’t) we do?

What could(n't) we do?


```
coffeepot -pp -g:wordlent2.ixml PEARL

<failure>
   <char1>P</char1>
   <not_char2>E</not_char2>
   <not_char3>A</not_char3>
   <not_char4>R</not_char4>
   <not_char5>L</not_char5>
</failure>
```

What could(n't) we do?

Grammar 1:

- determine correctness of the guess

Grammar 2:

- determine correctness of the characters

Grammar 3:

- label misplaced characters

What could(n't) we do?

```
-word = success; failure.

not_char = ~["PARSE"].

char1 = "P".
target_char1 = ["ARSE"].
char2 = "A".
target_char2 = ["PRSE"].
char3 = "R".
target_char3 = ["PASE"].
char4 = "S".
target_char4 = ["PARE"].
char5 = "E".
target_char5 = ["PARS"].
```

What could(n't) we do?

```
success = char1, char2, char3, char4, char5.

failure = char1, failure2to5;
      (target_char1; not_char), (success2to5; failure2to5).
-failure2to5 = char2, failure3to5;
      (target_char2; not_char), (success3to5; failure3to5).
-failure3to5 = char3, failure4to5;
      (target_char3; not_char), (success4to5; failure4to5).
-failure4to5 = char4, failure5;
      (target_char4; not_char), (success5; failure5).
-failure5 = target_char5; not_char.

-success2to5 = char2, char3, char4, char5.
-success3to5 = char3, char4, char5.
-success4to5 = char4, char5.
-success5 = char5.
```

What could(n't) we do?


```
coffeepot -pp -g:wordlent3.ixml PARSE

<success>
    <char1>P</char1>
    <char2>A</char2>
    <char3>R</char3>
    <char4>S</char4>
    <char5>E</char5>
</success>
```

What could(n't) we do?

C O U L D

```
coffeepot -pp -g:wordlent3.ixml COULD

<failure>
   <not_char>C</not_char>
   <not_char>O</not_char>
   <not_char>U</not_char>
   <not_char>L</not_char>
   <not_char>D</not_char>
</failure>
```

What could(n't) we do?


```
coffeepot -pp -g:wordlent3.ixml PEARL

<failure>
   <char1>P</char1>
   <target_char2>E</target_char2>
   <target_char3>A</target_char3>
   <target_char4>R</target_char4>
   <not_char>L</not_char>
</failure>
```

What could(n’t) we do?

What could(n't) we do?


```
coffeepot -pp -g:wordlent3.ixml PAPER

<failure>
   <char1>P</char1>
   <char2>A</char2>
   <target_char3>P</target_char3>
   <target_char4>E</target_char4>
   <target_char5>R</target_char5>
</failure>
```

What could(n't) we do?

Grammar 1:

- determine correctness of the guess

Grammar 2:

- determine correctness of the characters

Grammar 3:

- label misplaced characters

Grammar 4:

- label repeated characters correctly in guesses

What could(n't) we do?

```
-word = success; failure.
not_char = ~["PARSE"].

char1 = "P".
char2 = "A".
char3 = "R".
char4 = "S".
char5 = "E".
```

What could(n't) we do?

```
target_chars1234 = ["PARS"].
target_chars1234 = ["PARS"].
target_chars1235 = ["PARE"].
target_chars1245 = ["PASE"].
target_chars1345 = ["PRSE"].
target_chars2345 = ["ARSE"].
target_chars123 = ["PAR"].
target_chars124 = ["PAS"].
target_chars125 = ["PAE"].
[...]
target_chars45 = ["SE"].
target_chars1 = ["P"].
target_chars2 = ["A"].
target_chars3 = ["R"].
target_chars4 = ["S"].
target_chars5 = ["E"].
```

What could(n't) we do?

```
success = char1, char2, char3, char4, char5.

failure = fail_state00000; fail_state00005; fail_state00040;
          fail_state00045; fail_state00300; fail_state00305;
          fail_state00340; fail_state00345; fail_state02000;
          fail_state02005; fail_state02040; fail_state02045;
          fail_state02300; fail_state02305; fail_state02340;
          fail_state02345; fail_state10000; fail_state10005;
          fail_state10040; fail_state10045; fail_state10300;
          fail_state10305; fail_state10340; fail_state10345;
          fail_state12000; fail_state12005; fail_state12040;
          fail_state12045; fail_state12300; fail_state12305;
          fail_state12340.
```

What could(n't) we do?

```
-fail_state12340 = char1, char2, char3, char4,
  (char1>not_char; char2>not_char; char3>not_char;
  char4>not_char; not_char).

[...]

-fail_state02300 = (target_chars45; char2>not_char;
  char3>not_char; not_char),
  char2, char3,
  (target_chars15; char2>not_char; char3>not_char; not_char),
  (target_chars14; char2>not_char; char3>not_char; not_char).

[...]

-fail_state00000 = (target_chars2345; not_char),
  (target_chars1345; not_char), (target_chars1245; not_char),
  (target_chars1235; not_char), (target_chars1234; not_char).
```

What could(n’t) we do?

```
-fail_state02300 =
  (target_chars45; char2>not_char; char3>not_char; not_char),
  char2,
  char3,
  (target_chars15; char2>not_char; char3>not_char; not_char),
  (target_chars14; char2>not_char; char3>not_char; not_char).
```

What could(n't) we do?


```
-fail_state02300 =
    (target_chars45; char2>not_char;
                                char3>not_char; not_char),
    char2,
    char3,
    (target_chars15; char2>not_char;
                                char3>not_char; not_char),
    (target_chars14; char2>not_char;
                                char3>not_char; not_char).
```

What could(n't) we do?


```
-fail_state12000 =

        char1,

        char2,

        (target_chars45; char1>not_char;
                         char2>not_char; not_char),
        (target_chars35; char1>not_char;
                         char2>not_char; not_char),
        (target_chars34; char1>not_char;
                         char2>not_char; not_char).
```

What could(n't) we do?


```
-fail_state12000 =
P     char1,
A     char2,
P     (target_chars45; char1>not_char;
                       char2>not_char; not_char),
E     (target_chars35; char1>not_char;
                       char2>not_char; not_char),
R     (target_chars34; char1>not_char;
                       char2>not_char; not_char).
```

What could(n't) we do?


```
coffeepot -pp -g:wordlent4.ixml PAPER

<failure>
   <char1>P</char1>
   <char2>A</char2>
   <not_char>P</not_char>
   <target_chars35>E</target_chars35>
   <target_chars34>R</target_chars34>
</failure>
```

What could(n't) we do?

```
target_chars1234 = -char1; -char2; -char3; -char4.

[...]

target_chars4 = -char4.
target_chars5 = -char5.

[...]

-fail_state02300 = (target_chars45>misplaced;
char2>incorrect; char3>incorrect; not_char>incorrect),
char2>correct, char3>correct, (target_chars15>misplaced;
char2>incorrect; char3>incorrect; not_char>incorrect),
(target_chars14>misplaced; char2>incorrect; char3>incorrect;
not_char>incorrect).
```

What could(n't) we do?


```
coffeepot -pp -g:wordlent4a.ixml PAPER

<failure>
   <correct>P</correct>
   <correct>A</correct>
   <incorrect>P</incorrect>
   <misplaced>E</misplaced>
   <misplaced>R</misplaced>
</failure>
```

What could(n’t) we do?

PAPER

What could(n't) we do?


```
coffeepot -pp -g:wordlent4b.ixml PAPER

Found 4 possible parses.
<success xmlns:ixml='http://invisiblexml.org/NS'
ixml:state='ambiguous'>
   <correct>P</correct>
   <correct>A</correct>
   <correct>P</correct>
   <correct>E</correct>
   <correct>R</correct>
</success>
```

What could(n't) we do?


```
<success xmlns:ixml='http://
invisiblexml.org/NS'
ixml:state='ambiguous'>
   <correct>P</correct>
   <correct>A</correct>
   <correct>P</correct>
   <correct>E</correct>
   <correct>R</correct>
</success>
<failure xmlns:ixml='http://
invisiblexml.org/NS'
ixml:state='ambiguous'>
   <incorrect>P</incorrect>
   <correct>A</correct>
   <correct>P</correct>
   <correct>E</correct>
   <correct>R</correct>
</failure>
```

```
<failure xmlns:ixml='http://
invisiblexml.org/NS'
ixml:state='ambiguous'>
   <correct>P</correct>
   <correct>A</correct>
   <incorrect>P</incorrect>
   <correct>E</correct>
   <correct>R</correct>
</failure>
<failure xmlns:ixml='http://
invisiblexml.org/NS'
ixml:state='ambiguous'>
   <misplaced>P</misplaced>
   <correct>A</correct>
   <misplaced>P</misplaced>
   <correct>E</correct>
   <correct>R</correct>
</failure>
```

What could(n’t) we do?

What could(n't) we do?

A R R A Y

```
coffeepot -pp -g:wordlent4c.ixml ARRAY

Found 16 possible parses.
```

```
<success xmlns:ixml='http://invisiblexml.org/NS'
ixml:state='ambiguous'>
   <correct>A</correct>
   <correct>R</correct>
   <correct>R</correct>
   <correct>A</correct>
   <correct>Y</correct>
</success>
```

What could(n’t) we do?

What could(n't) we do?


```
coffeepot -pp -g:wordlent4c.ixml NANNA

Found 108 possible parses.

<success xmlns:ixml='http://invisiblexml.org/NS'
ixml:state='ambiguous'>
   <correct>N</correct>
   <correct>A</correct>
   <correct>N</correct>
   <correct>N</correct>
   <correct>A</correct>
</success>
```

What could(n't) we do?

Grammar 1:

- determine correctness of the guess

Grammar 2:

- determine correctness of the characters

Grammar 3:

- label misplaced characters

Grammar 4:

- label repeated characters in guesses

Grammar 5:

- permit target words with repeated characters

What could(n't) we do?

Permit target words with repeated characters

- limitations of context-free grammars

What could(n't) we do?

Permit target words with repeated characters

- limitations of context-free grammars

- what we want to say:
    - if we guess "APPLE", character 2 is misplaced, because the P in position 1 has not yet been correctly matched, although the other P in position 3 has been matched

What could(n't) we do?

Permit target words with repeated characters

- limitations of context-free grammars

- what we want to say:
    - if we guess "PAPPA", character 4 is incorrect, because the Ps in positions 1 and 3 have both been correctly matched

What could(n't) we do?

Permit target words with repeated characters

- limitations of context-free grammars:
  - any time we want to say something like "X is misplaced if and only if there is another X elsewhere in the input", we're talking about context-sensitivity

  - context-free grammars can only directly say "X = Y", and not "X (when it precedes another X) = Y, otherwise X = Z"

What could(n't) we do?

Permit target words with repeated characters

- finding ways around the limitations
  - sometimes, we can find ways of expressing contextual constraints using only a context-free grammar

  - in addition, iXML has features that can help us to construct the XML output we want, even though we've had to construct the grammar with extra complexity

What could(n't) we do?

X (when followed by an alphabetic character) = Y
X (when followed by a digit 0-9) = Z.

```
S    =    Y_X ; Z_X.
-Y_X =    X>Y, ["a"-"z"].
-Z_X =    X>Z, ["0"-"9"].
-X   =    "x".
```

If we parse "xa", we get <S><Y>x</Y>a</S>.
If we parse "x1", we get <S><Z>x</Z>1</S>.

What could(n't) we do?

Permit target words with repeated characters

- finding ways around the limitations
  - in order to do this, we often need to add in extra rules and nonterminals

- if we're trying to reconstruct quite a bit of context, we may end up with a very complex grammar

What could(n’t) we do?

PAPER

What could(n't) we do?

```
-word = success; failure.

not_char = ~["PAER"].

char13 = 'P'.
char2 = 'A'.
char4 = 'E'.
char5 = 'R'.

success = char13>correct, char2>correct, char13>correct,
char4>correct, char5>correct.
```

What could(n't) we do?

```
-fail_state02300 =
      (not_char>incorrect; char2>incorrect; char4>misplaced;
          char5>misplaced),
      char2>correct,
      char13>correct,
      (not_char>incorrect; char13>misplaced; char2>incorrect;
          char5>misplaced),
      (not_char>incorrect; char13>misplaced; char2>incorrect;
          char4>misplaced).
```

What could(n't) we do?


```
-fail_state02300 =

    (not_char>incorrect; char2>incorrect;
                         char4>misplaced; char5>misplaced),
    char2>correct,

    char13>correct,

    (not_char>incorrect; char2>incorrect;
                         char13>misplaced; char5>misplaced),
    (not_char>incorrect; char2>incorrect;
                         char13>misplaced; char4>misplaced).
```

What could(n't) we do?


```
-fail_state02300 =

    (not_char>incorrect; char2>incorrect;
                         char4>misplaced; char5>misplaced),
    char2>correct,

    char13>correct,

    (not_char>incorrect; char2>incorrect;
                         char13>misplaced; char5>misplaced),
    (not_char>incorrect; char2>incorrect;
                         char13>misplaced; char4>misplaced).
```

What could(n't) we do?


```
coffeepot -pp -g:wordlent9.ixml PAPER

<success>
   <correct>P</correct>
   <correct>A</correct>
   <correct>P</correct>
   <correct>E</correct>
   <correct>R</correct>
</success>
```

What could(n't) we do?


```
coffeepot -pp -g:wordlent9.ixml PAPPA

<failure>
   <correct>P</correct>
   <correct>A</correct>
   <correct>P</correct>
   <incorrect>P</incorrect>
   <incorrect>A</incorrect>
</failure>
```

What could(n't) we do?


```
coffeepot -pp -g:wordlent9.ixml APPLE

<failure>
   <misplaced>A</misplaced>
   <misplaced>P</misplaced>
   <correct>P</correct>
   <incorrect>L</incorrect>
   <misplaced>E</misplaced>
</failure>
```

What could(n't) we do?

Could we reproduce the logic of a Wordle puzzle in iXML?

Requirements:

- ✅ set a five-letter target word
- ✅ assess a guessed word against the target
- ✅ determine correctness of the guess
- ✅ determine correctness of each guessed character

What could(n't) we do?

- some limitations of iXML
- finding ways around the limitations

What should(n't) we do?

- when to use workarounds
- when to hand off tasks

What should(n't) we do?

- grammar 1 has 19 nonterminals

- grammar 9 has 39 nonterminals

- a new grammar is required for each new word, since we have to calculate the correct pattern of repeats etc.

- writing these grammars by hand is probably not efficient

What should(n’t) we do?

- once we've worked out how to reproduce the logical structure we want, generating further grammars with e.g. a Python program is a valid option

What should(n't) we do?

Should we generate a web app to play Wordle games, using our iXML grammar?

- a combination of iXML insertions, renaming, and suppressions could permit us to generate a static HTML page
- how would this look for our basic first grammar?

What should(n’t) we do?

```
html = head, body.
head = title.
title = +"Wordlen't".
body = img, (success>div; failure>div).
img = @src, @width.
src = +"wordlent_banner.png".
width = +"300".

success = success_table>table.
success_table = success_tr>tr.
success_tr = char1, char2, char3, char4, char5.

failure = failure_table>table.
failure_table = failure_tr>tr.
failure_tr = failure1; failure2; failure3; failure4; failure5.
```

What should(n't) we do?

```
char1 = @success_class, "P".
-not_char1 = @not_class, ~["P"].
char2 = @success_class, "A".
-not_char2 = @not_class, ~["A"].
char3 = @success_class, "R".
-not_char3 = @not_class, ~["R"].
char4 = @success_class, "S".
-not_char4 = @not_class, ~["S"].
char5 = @success_class, "E".
-not_char5 = @not_class, ~["E"].

-anychar = @any_class, ["A"-"Z"].

not_class>class = +"not_colour".
any_class>class = +"any_colour".
@success_class>class = +"success_colour".
```

What should(n’t) we do?

```
<html>
   <head>
      <title>Wordlen't</title>
   </head>
   <body>
      <img src='wordlent_banner.png' width='300'/>
      <div>
         <table>
            <tr>
               <char1 class='success_colour'>P</char1>
               <char2 class='success_colour'>A</char2>
               <char3 class='success_colour'>R</char3>
               <char4 class='success_colour'>S</char4>
               <char5 class='success_colour'>E</char5>
            </tr>
         </table>
      </div>
   </body>
</html>
```

What should(n't) we do?

Should we generate a web app to play Wordle games, using our iXML grammar?

- a combination of iXML insertions, renaming, and suppressions could permit us to generate a static HTML page

- but the machinery needed is a lot more complicated than this basic page

What should(n't) we do?

Should we generate a web app to play Wordle games, using our iXML grammar?

- trying to do this in pure iXML would be complex and frustrating, and would undoubtedly require us to make compromises in functionality

- it's okay to decide that you've hit a limit

What could we do?

- spend a lot of time writing complex grammars
- get very frustrated
- end up writing sub-optimal code

What should we do?

- experiment
- be flexible
- combine the best of iXML with the best of other technologies
- learn what works