The one that minimizes the maximum remaining candidates

Max (codemaker response): assume we get the response with the largest remaining candidate set

Min (our guess): pick the one that, worst case, results in the smallest candidate set

How do we calculate this?

For all codes not yet guessed:
Consider all possible responses:
Calculate the size of the remaining candidates if we guessed that code and got that response
Select response with largest remaining for that code
Select code with smallest max
We can precompute the entire tree of possibilities

Expensive upfront to compute

Playing becomes fast
**Game tree**

Parent: [Red, Red, Green]

<table>
<thead>
<tr>
<th>codemaker response</th>
<th>(0,0)</th>
<th>(0,1)</th>
<th>(0,2)</th>
<th>(0,3)</th>
<th>(1,0)</th>
<th>(1,1)</th>
<th>(1,2)</th>
<th>(2,0)</th>
<th>(3,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>candidates remaining</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**What now?**

**Game tree**

Parent: [Red, Red, Green]

<table>
<thead>
<tr>
<th>codemaker response</th>
<th>(0,0)</th>
<th>(0,1)</th>
<th>(0,2)</th>
<th>(0,3)</th>
<th>(1,0)</th>
<th>(1,1)</th>
<th>(1,2)</th>
<th>(2,0)</th>
<th>(3,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>candidates remaining</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Use lose to indicate we don’t have any options left (this shouldn’t happen if we use a reasonable strategy).

**Game tree**

Parent: [Red, Red, Green]

<table>
<thead>
<tr>
<th>codemaker response</th>
<th>(0,0)</th>
<th>(0,1)</th>
<th>(0,2)</th>
<th>(0,3)</th>
<th>(1,0)</th>
<th>(1,1)</th>
<th>(1,2)</th>
<th>(2,0)</th>
<th>(3,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>candidates remaining</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

**Game tree**

Parent: [Red, Red, Green]

<table>
<thead>
<tr>
<th>codemaker response</th>
<th>(0,0)</th>
<th>(0,1)</th>
<th>(0,2)</th>
<th>(0,3)</th>
<th>(1,0)</th>
<th>(1,1)</th>
<th>(1,2)</th>
<th>(2,0)</th>
<th>(3,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>candidates remaining</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

**Game tree**

Parent: [Red, Red, Green]

<table>
<thead>
<tr>
<th>codemaker response</th>
<th>(0,0)</th>
<th>(0,1)</th>
<th>(0,2)</th>
<th>(0,3)</th>
<th>(1,0)</th>
<th>(1,1)</th>
<th>(1,2)</th>
<th>(2,0)</th>
<th>(3,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>candidates remaining</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Use lose to indicate we don’t have any options left (this shouldn’t happen if we use a reasonable strategy).
What now?

Building the game tree

- If 0 options then Lose
- If 1 option and the response was (num_pegs, 0) then Win
- Otherwise, build another Tree:
  - Guess = one that minimizes the maximum remaining candidates over all responses
  - Break ties by 1) those that are still valid codes and 2) found first in candidate (valid) list
  - Recurse on responses
Representing the game tree

[Diagram of a game tree with nodes labeled (0,0), (0,1), (0,2), (0,3), (1,0), (1,1), (1,2), (2,0), (3,0) and a codemaker response of [Red, Red, Green].]

How do we store this tree?

The responses aren't explicitly stored in the tree. There is an implicit ordering to the subtrees that correspond to these responses.

```plaintext
datatype knuth_tree = Lose | Step of code * knuth_tree list | Win;
```
Representing the game tree

A simple example

Write some SML to create this tree.

What is the type signature of this function?

What does it do?
A simple example

```haskell
local
  fun first [] = raise InternalInconsistency
  | first (x::xs) = x;

fun badNextMove (Step (code, tree)) = (code, first tree)
  | badNextMove _ = raise InternalInconsistency;

knuth_tree -> (code @ knuth_tree)

Returns the next code and then always chooses the first element in the knuth tree (i.e. associated with response (0,0))
```

Midterm

- SML
  - datatypes (with non-zero constructors, recursive datatypes)
  - mutual recursion
  - handling exceptions

- Binary numbers
  - signed representation
  - adding
  - shifting

- Parsing: EBNF grammars

- Circuits
  - general ideas (building circuits, truth tables, etc.)
  - minterm expansion
  - specific circuits (decoders, multiplexers)

Midterm

- Encryption
  - encryption/decryption
  - modular arithmetic

- Resources:
  - We will provide you with the graphical pictures for the gates.
  - Like the previous midterms, you may bring one single-sided, 8.5” x 11” piece of paper with notes.

Course registration