

Building a Game-Playing AI

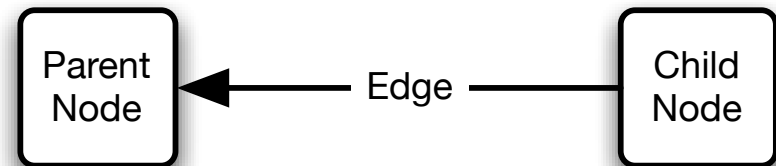
For Tic-Tac-Toe

Recursion, Trees, Adversarial Search (Min and Max)

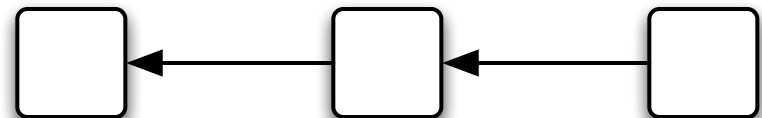
Trees

- A **data structure** is an organization of data in a special way that is useful to us. For example, an **array** is an collection of data in a sequence.

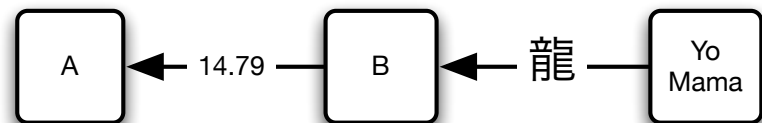
- In computer science, a **tree** is a data structure consisting of two kinds of data: **nodes** and **edges**. Each edge connects two nodes together. For each edge, one of those nodes is called the **parent** and the other node is called the **child**.



- A node can be both a parent and a child.

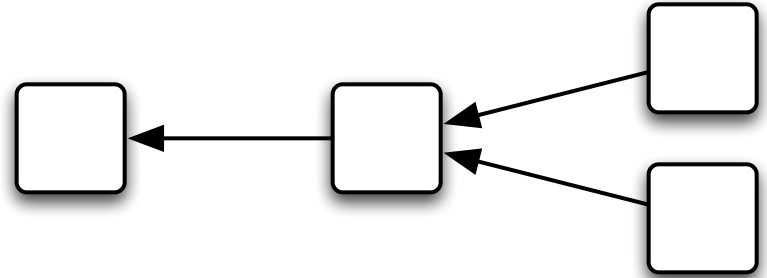


- Nodes and edges can be **labeled**. If the label is a number, we say the node or edge has been **weighted**.

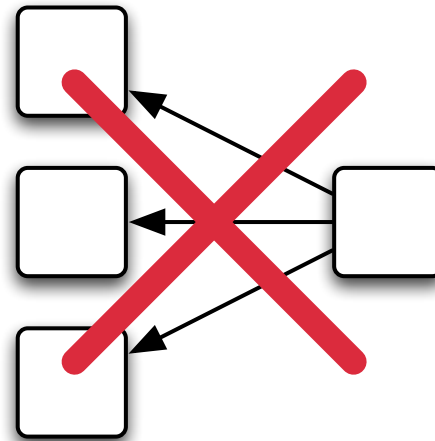


Trees

- Nodes may have multiple children.

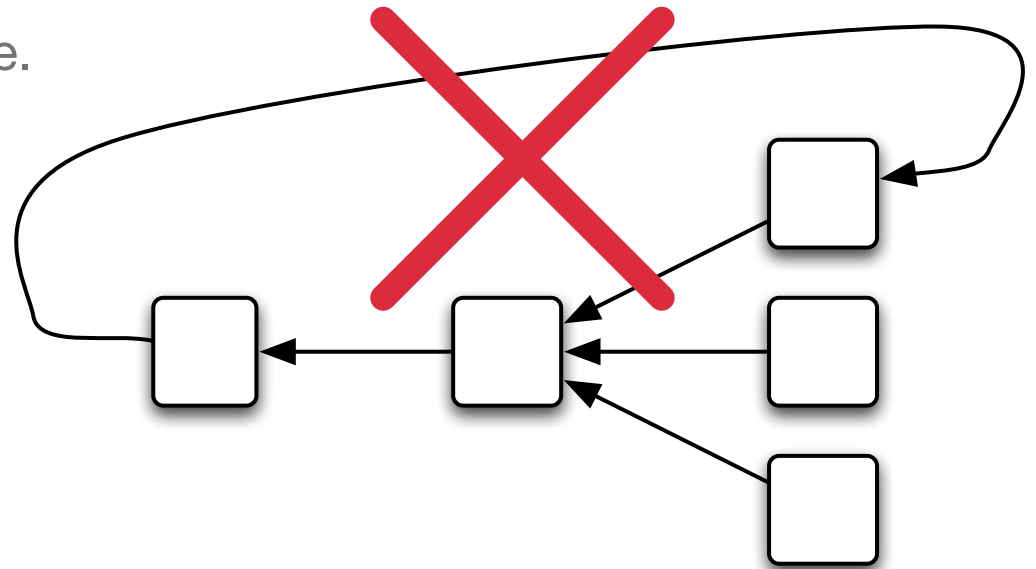


- Nodes **may not have** multiple parents.



Trees

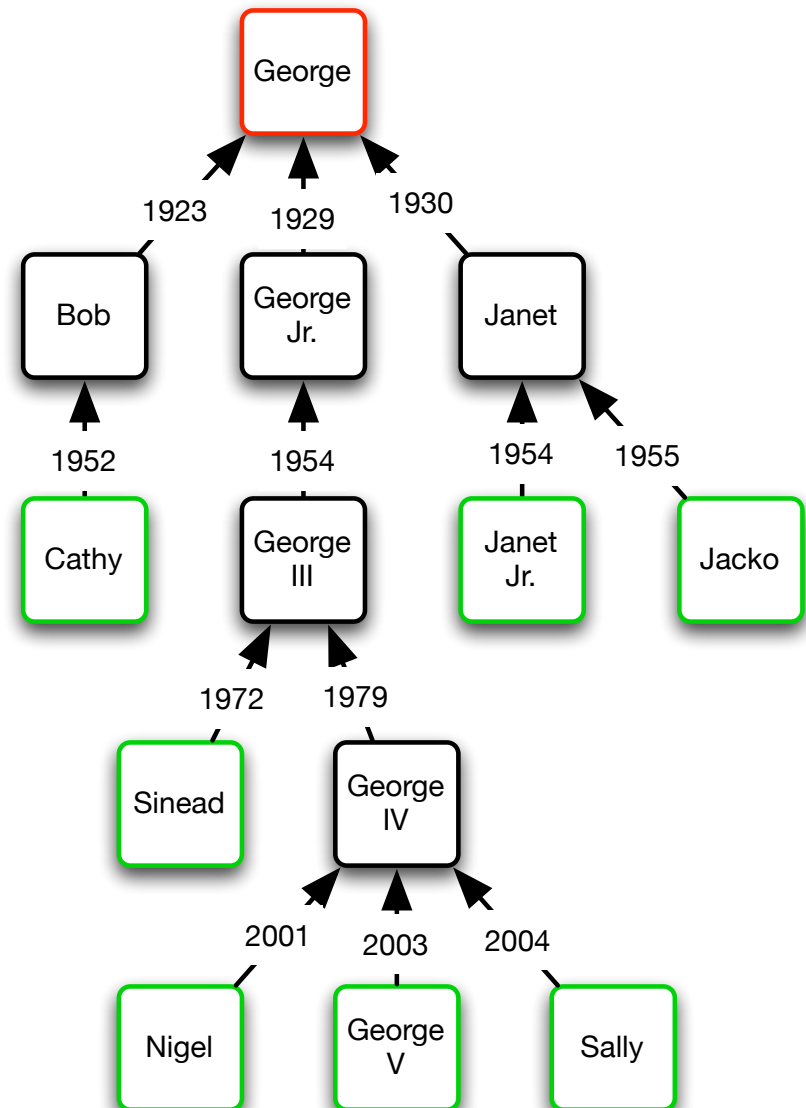
- You may not have **cycles** in a tree. That is, if you follow a path from a child to a parent, and then to its parent, etc. you can **never get back to the child**.



- Every tree has **one node** which has no parent. It is called the **root**.
- Every tree has **multiple nodes** which have no children. They are called **leaves**.

Tree

- It is traditional to draw a tree with the **root on the top** and the **leaf nodes on the bottom** (yes, trees “grow down” in Computer Science).
- Here is a more elaborate tree. We’ve the root and the leaf nodes so you can see them.



Print this tree

- You have the following functions:

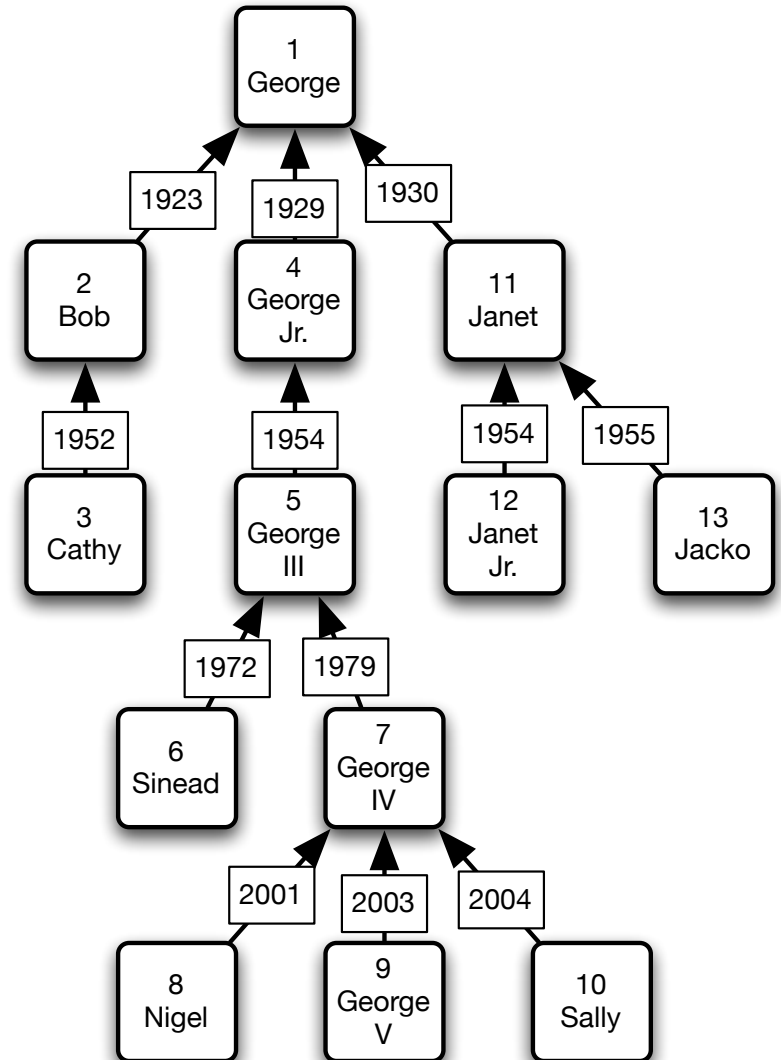
root() provides you with the root node.

numChildren(*node*) gives you the number of children to a node.

children(*node*) gives all the children of the node *node*.

print(*node*) prints a node to the screen.

- Write some code which prints the nodes out in the order shown at right (1–13). It must work for any size/shape tree.



Recursion

- Print the parent
 - If there is more than one child of the parent
 - For each child of the parent
 - Print the child
- **To print more than one child, we need to have a “do that again” concept**
 - Print the parent
 - If there is more than one child of the parent
 - For each child of the parent
 - Do the same thing as all of this, but for the child
- **Get rid of pronoun-like phrases (“all of this”) and instead name ourselves**
 - This is Function Foo, and it operates on a *node*:
 - Print the node
 - If there is more than one child of the node
 - For each child of the node
 - do Function Foo using the *child* instead of the node

Recursion

- **Convert to our pseudocode form:**

Function foo(node):

 print(node)

 if (numChildren(node) > 0)

 for each child in children(node)

 foo(child)

- **Now we need to start up foo:**

 foo(root)



- **Recursion needs three things:**

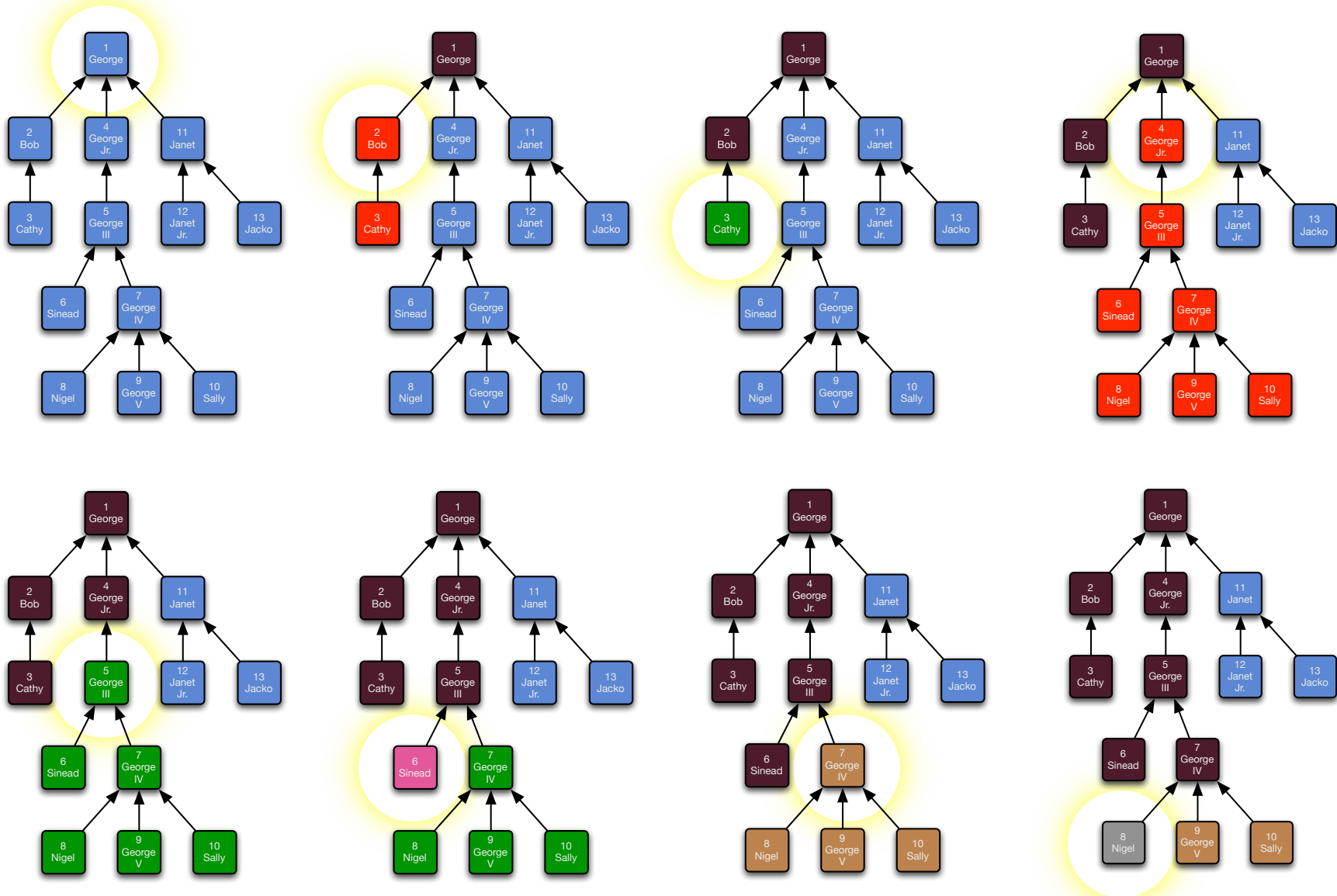
 A way for a function to **include itself**

 A **base case**: the rule which states when the self-inclusion should stop

 A **starting point**

Recursion in Action

- The current node is:  and old: 
- Each piece of a tree gets labelled a new color when we begin work on it



Tic-Tac-Toe is...

- **A total information game**

Everyone can see the entire state of the game at any time

- **A non-stochastic game**

There's no chance involved

- **A turn-based game**

Each player takes a turn

- **A zero-sum game**

If I win, you lose

- **A two-player game**

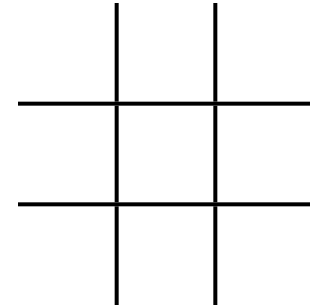
Aha! Two can play at that game!

Min and Max

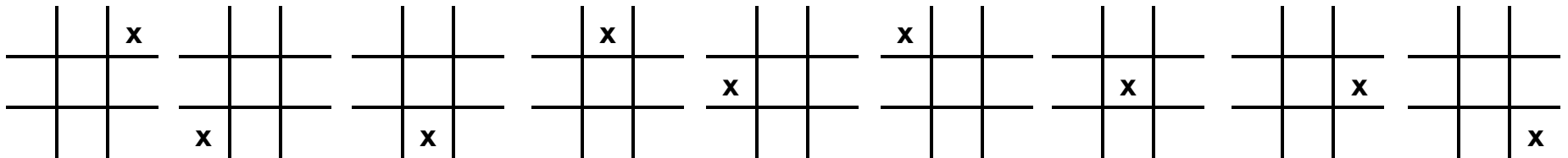
- Our two players are named Min and Max. Both Min and Max are trying to win.
- Because Tic-Tac-Toe is a **zero-sum two-player game**, Min wins when Max loses. So we may say that Max is **trying to make Max win**, and Min is **trying to make Max lose**.
- That way we can think of the game in terms of a single variable: how well Max is doing.
- We will denote “Max’s Turn” with ▲ and “Min’s Turn” with ▼.

All Possible Games

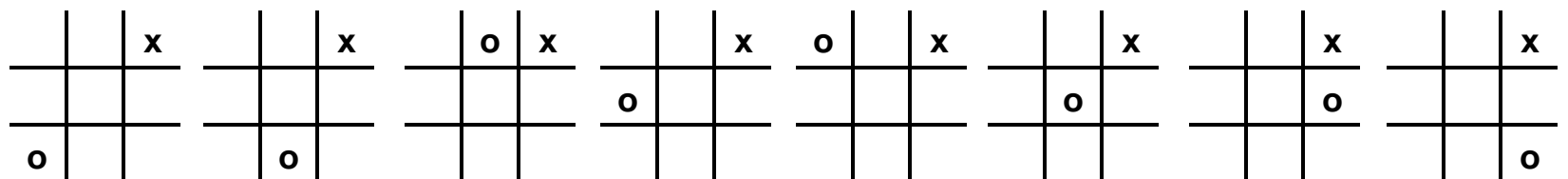
- The game starts out in the same way:



- Let's let Max be X (sure, why not?). Here are all of his **9** possible moves.



- Min will be O. For the X move at top left, there are only **8** moves for Min.



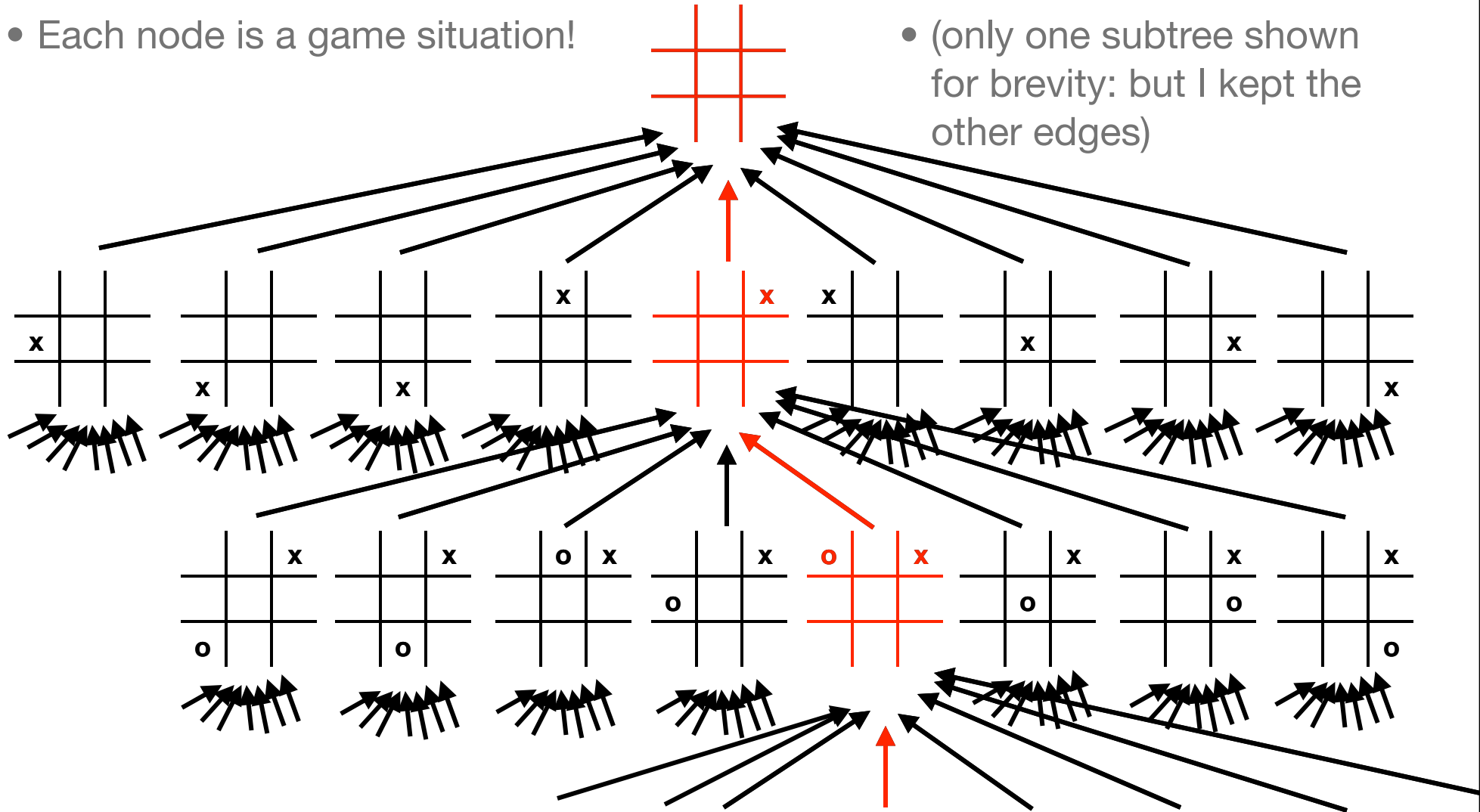
Questions

- Initially, Max gets to make one of 9 moves.
For each such move, Min gets to make one of 8 countermoves.
For each such move, Max gets to make one of 7 countermoves.
...
- How long is a game, in terms of number of moves?
- How many total games are there?

This is a Tree!

- Each node is a game situation!

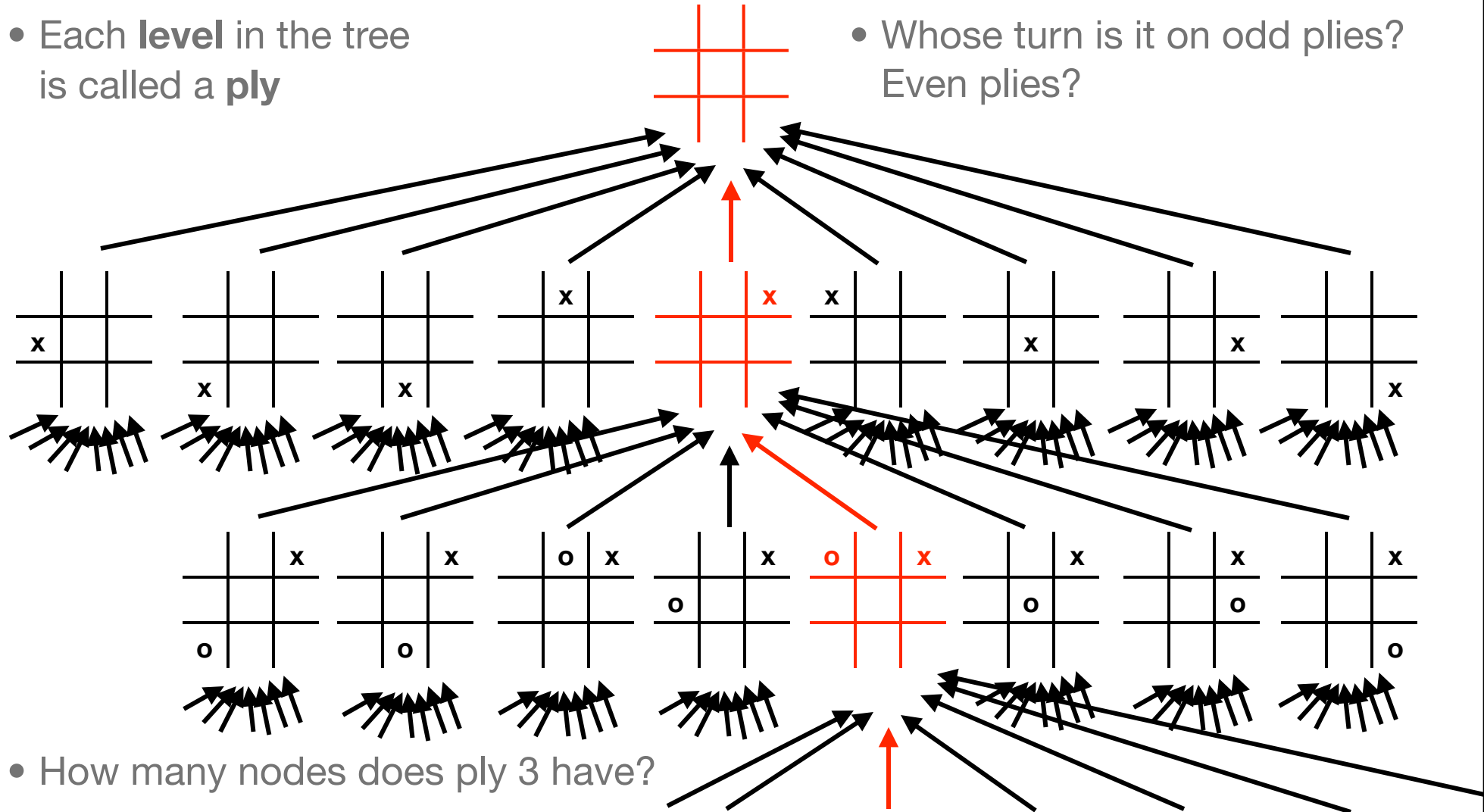
- (only one subtree shown for brevity: but I kept the other edges)



Plies

- Each **level** in the tree is called a **ply**

- Whose turn is it on odd plies?
Even plies?



The Score of a Game Situation

- If Max has **won** in a given game situation, the score is **1**

X	O	O
	X	
		X

- If Max has **lost** in a given game situation, the score is **-1**

O	O	O
X	X	
		X


- If Max has **tied** in a given game situation, the score is **0**

X	O	O
O	X	X
X	X	O

The Score of a Game Situation

- What about if the game's not completed? What's the score now?

x	o	o
	x	



- Let us assume that Max is an intelligent person. If there is a best move, he will make it. Thus the score of this position is the best score he can make out of it.

x	o	o
x	x	

x	o	o
	x	x

x	o	o
	x	
x		

x	o	o
	x	
	x	

x	o	o
	x	
		x

The Score of a Game Situation

- Thus if Max is playing, we can define the score of a game node as the **maximum score** over all of its children nodes!

x	o	o
	x	

▲ score=1

- Likewise, we assume Min is intelligent, and so if Min is playing, he will try to **minimize** Max's score.

- So if Min is playing, can define the score of a game as the **minimum score** over all its children nodes.

	o	o
x	x	
		x

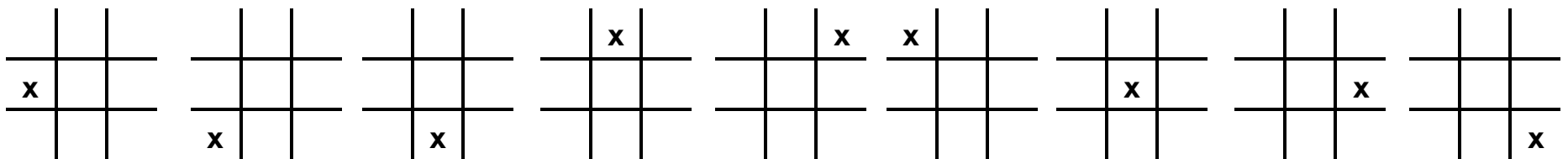
▼ score=-1

The Score of a Game Situation

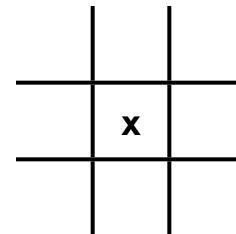
- If the game has ended and Max has won: **1**
- If the game has ended and Min has won: **-1**
- If the game has ended in a draw: **0**
- If the game has not ended and it is Max's turn: **$\max(\text{children}(game))$**
- If the game has not ended and it is Min's turn: **$\min(\text{children}(game))$**

What We Want To Do

- The Computer is Max.
- His enemy (you) is Min.
- It's Max's turn. What should he do?
- Max calculates the score of the **result** of each of his moves:



- Max makes the move with the highest score:



Calculating the Score

- We need a few functions:

max(x, y)

min(x, y)

Return the max (min) of x and

children(game)

Returns the children of the node *game*

maxWon(game)

minWon(game)

draw(game)

Return whether max won, min won, or
there was a draw

maxsTurn(game)

minsTurn(game)

Return whether or not it's max's or
min's turn in the game at the moment

Calculating the Score

- **It's recursive!**

Function score(game):

will return to you a score for game

If maxWon(game) return a 1

If minWon(game) return a -1

If draw(game) return a 0

If maxsTurn(game)

 highest = -1

 for each child in children(game)

 highest = max(highest, score(child))

 return highest

If minsTurn(game)

 lowest = 1

 for each child in children(game)

 lowest = min(lowest, score(child))

 return lowest

The Elements of Recursion

- **Including Ourselves**

Inside the `score(...)` function, we call `score(child)`

- **Base Case**

We don't call `score(child)` if the game has ended (there'd be no children!)

- **Starting Position**

Function `makeMove(game)`:

`bestGame = nothing`

`bestScore = -1`

For each `child` in `children(game)`

`s = score(child)`

 if `s > bestScore`

`bestScore = s, bestGame = child`

Make the move that resulted in `bestGame`

Things to Think About

- Tic-Tac-Toe is one thing. How about Chess? Go? Why are they harder?
- If we can't search all the way to the end in Chess, how can we compute the score of a node?
- Each time a function calls itself recursively, the programming language must record this event so it can return to it later. Once we have returned to it we can get rid of it. But while we're holding onto these records (known collectively as the **call stack**), if there are too many of them, eventually will exhaust memory. Is this going to be a problem in our case?
- Some games (like Backgammon) include a degree of **chance**. How can we modify this game?