Game Trees

Tic Tac Toe

- representing pebbles on a board
- the class Board
- the value of a board

A Game Tree for Tic Tac Toe

- an example of looking ahead
- the class BoardTree

Calculating Game Trees

- suggestions for next best move

MCS 275 Lecture 39
Programming Tools and File Management
Jan Verschelde, 21 April 2008

Tic Tac Toe

a board game for two players

Tic tac toe is a game

1. for two players
2. on a square 3-by-3 board
3. each player has 3 pebbles
4. goal is to have pebbles lined up

Playing strategy:

1. look ahead: enumerate future moves
2. evaluate: each board has a value
3. optimize: select move with largest value

Tic Tac Toe

a board game for two players

Tic tac toe is a game

1. for two players
2. on a square 3-by-3 board
3. each player has 3 pebbles
4. goal is to have pebbles lined up

Playing strategy:

1. look ahead: enumerate future moves
2. evaluate: each board has a value
3. optimize: select move with largest value

Tic Tac Toe

a board game for two players

Tic tac toe is a game

1. for two players
2. on a square 3-by-3 board
3. each player has 3 pebbles
4. goal is to have pebbles lined up

Playing strategy:

1. look ahead: enumerate future moves
2. evaluate: each board has a value
3. optimize: select move with largest value

Tic Tac Toe

a board game for two players

Tic tac toe is a game

1. for two players
2. on a square 3-by-3 board
3. each player has 3 pebbles
4. goal is to have pebbles lined up

Playing strategy:

1. look ahead: enumerate future moves
2. evaluate: each board has a value
3. optimize: select move with largest value

Tic Tac Toe

a board game for two players

Tic tac toe is a game

1. for two players
2. on a square 3-by-3 board
3. each player has 3 pebbles
4. goal is to have pebbles lined up

Playing strategy:

1. look ahead: enumerate future moves
2. evaluate: each board has a value
3. optimize: select move with largest value

Tic Tac Toe

a board game for two players

Tic tac toe is a game

1. for two players
2. on a square 3-by-3 board
3. each player has 3 pebbles
4. goal is to have pebbles lined up

Playing strategy:

1. look ahead: enumerate future moves
2. evaluate: each board has a value
3. optimize: select move with largest value

Tic Tac Toe

a board game for two players

Tic tac toe is a game

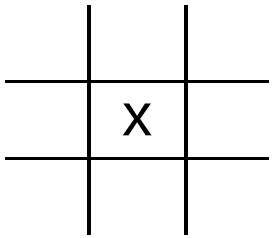
1. for two players
2. on a square 3-by-3 board
3. each player has 3 pebbles
4. goal is to have pebbles lined up

Playing strategy:

1. look ahead: enumerate future moves
2. evaluate: each board has a value
3. optimize: select move with largest value

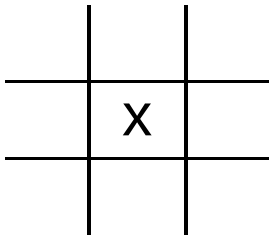
Playing the Game

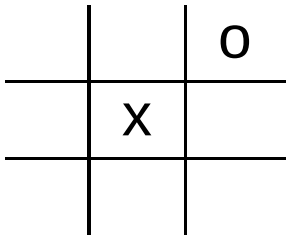
placing pebbles x and o



Playing the Game

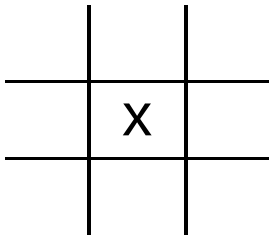
placing pebbles x and o

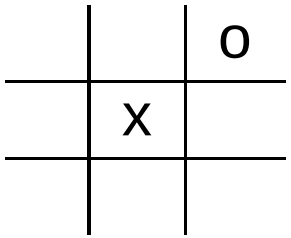


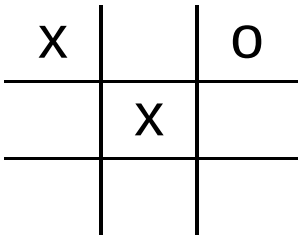


Playing the Game

placing pebbles x and o

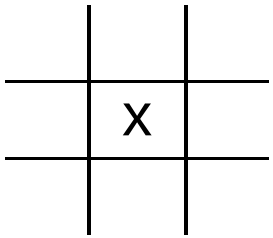


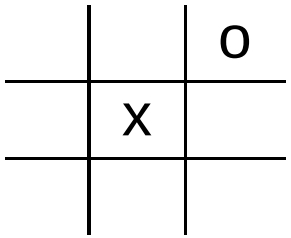


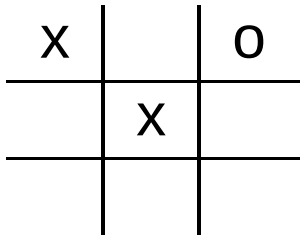


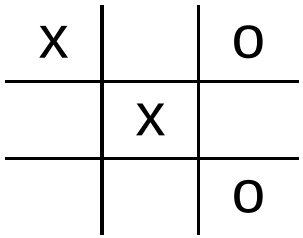
Playing the Game

placing pebbles x and o



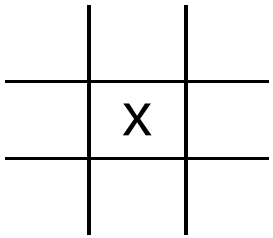


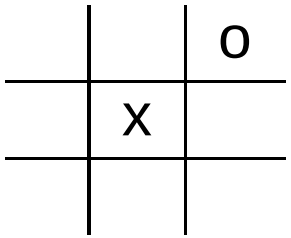


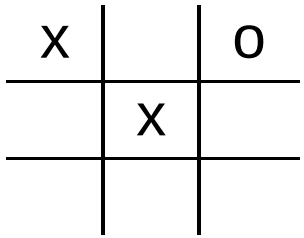


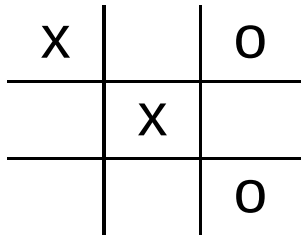
Playing the Game

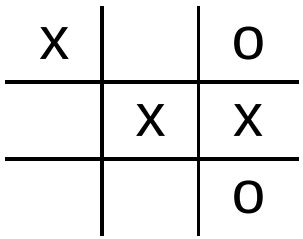
placing pebbles x and o





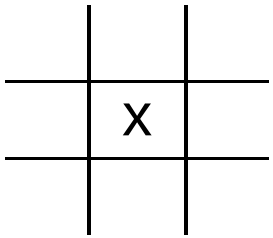


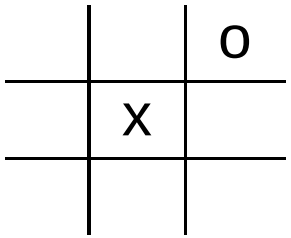


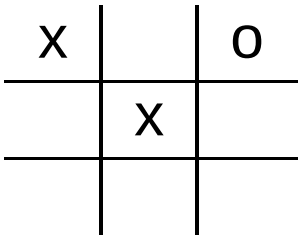


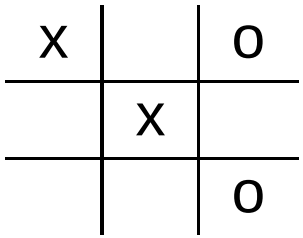
Playing the Game

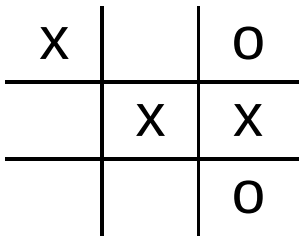
placing pebbles x and o

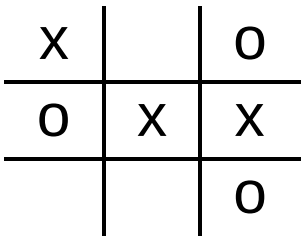






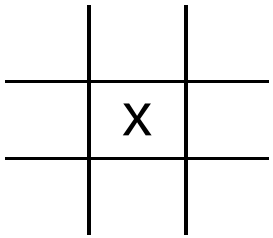


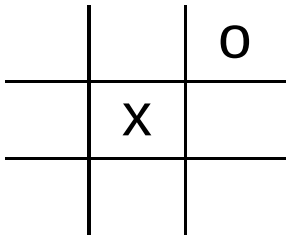


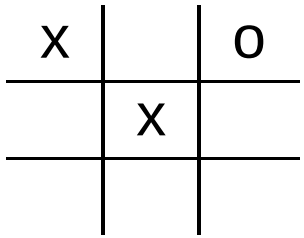


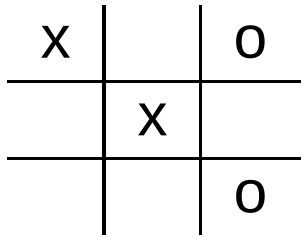
Playing the Game

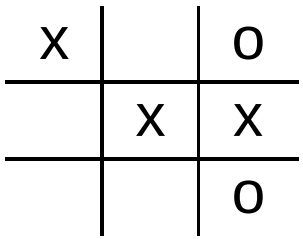
placing pebbles x and o

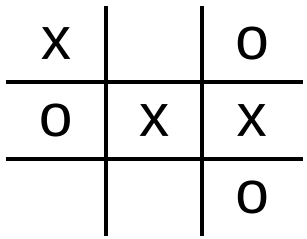


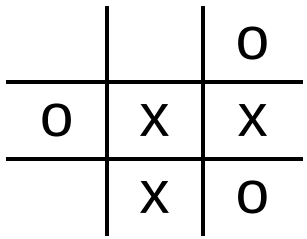






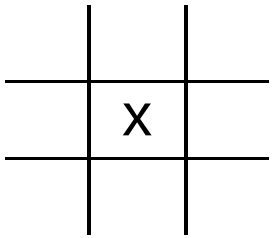


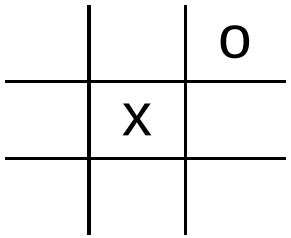


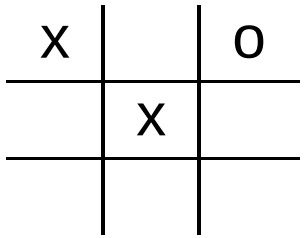


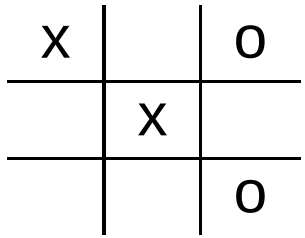
Playing the Game

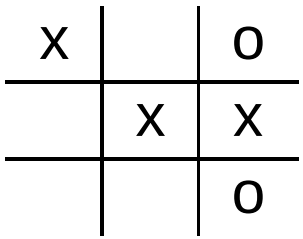
placing pebbles x and o

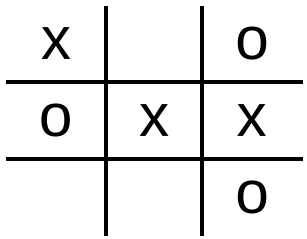


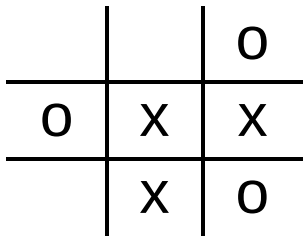


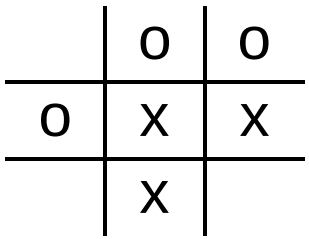






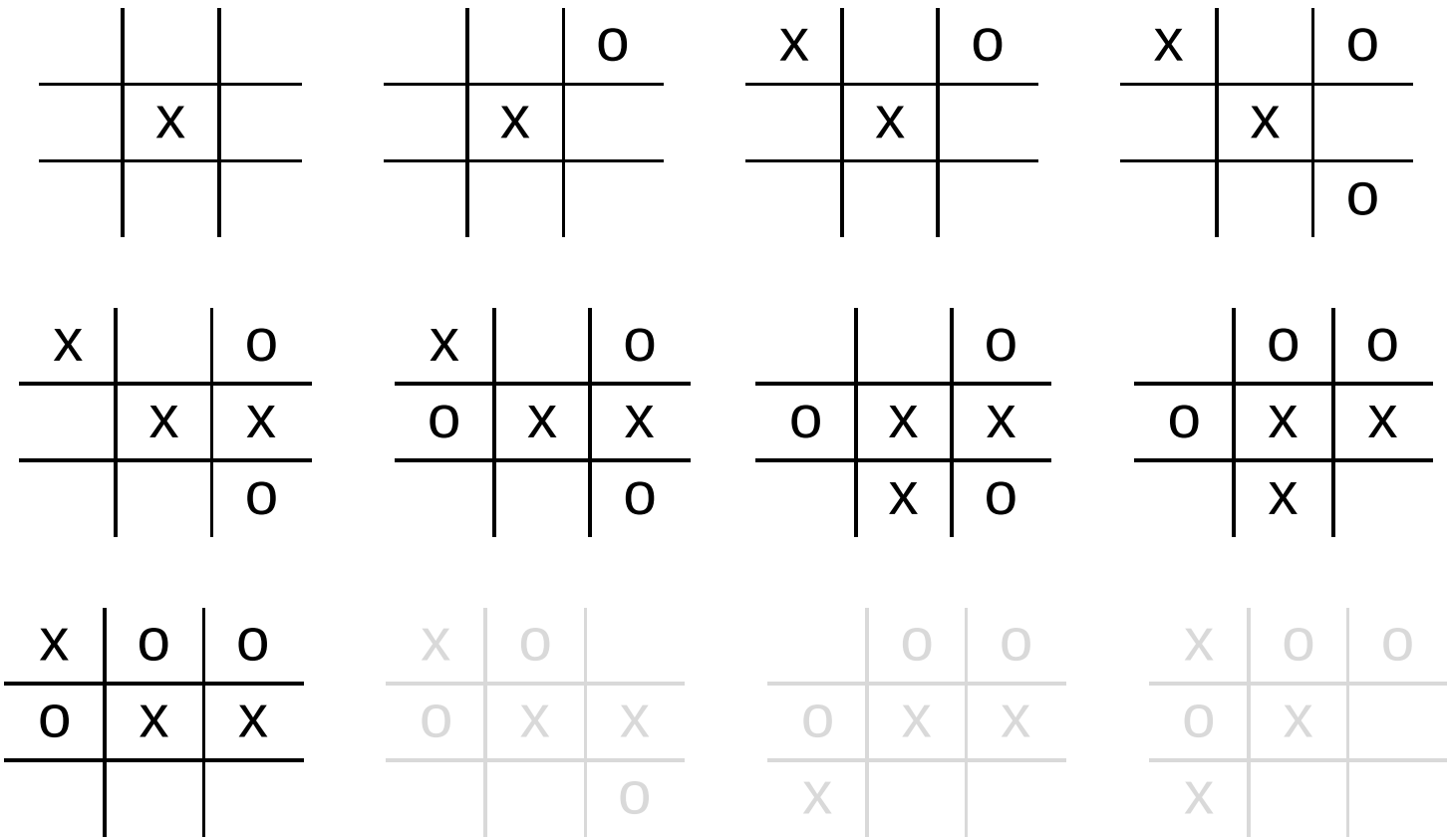






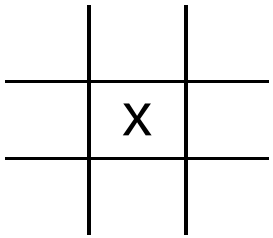
Playing the Game

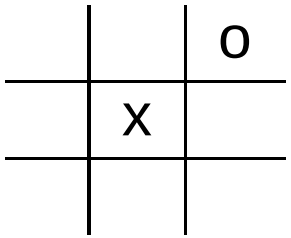
placing pebbles x and o

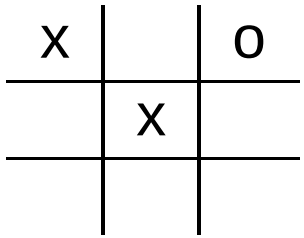


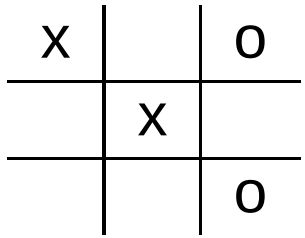
Playing the Game

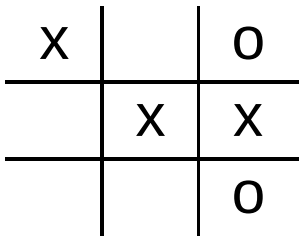
placing pebbles x and o

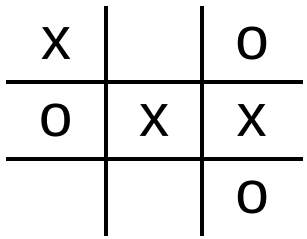


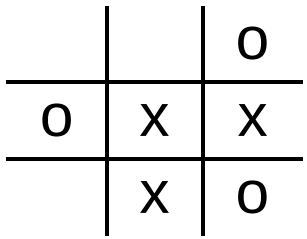


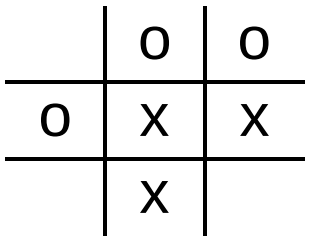


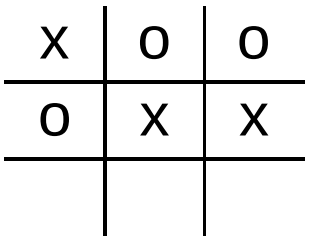


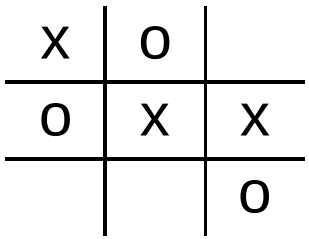






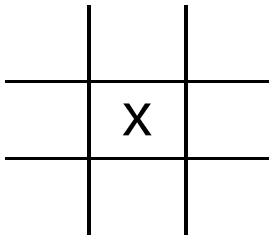


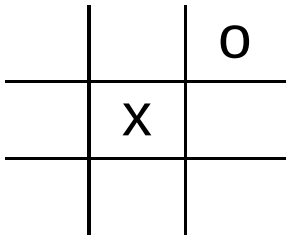


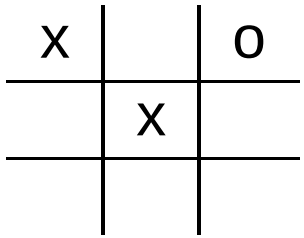


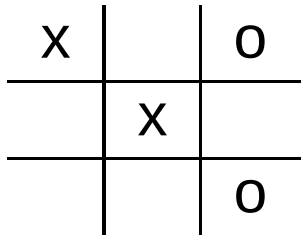
Playing the Game

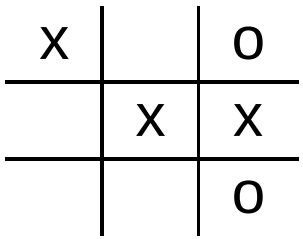
placing pebbles x and o

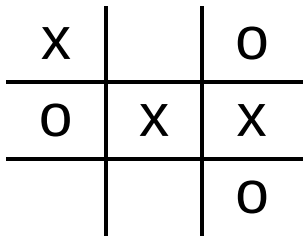


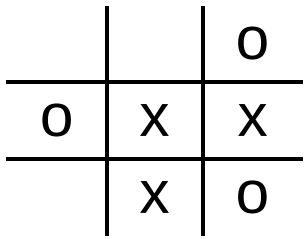


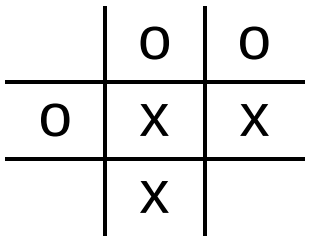


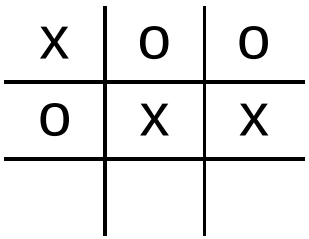


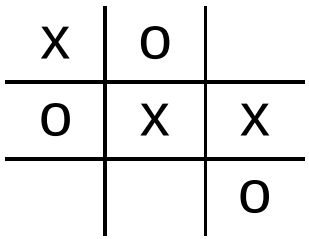


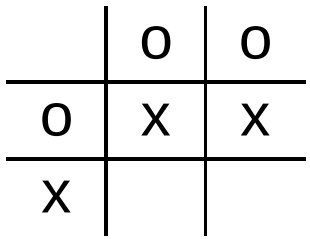






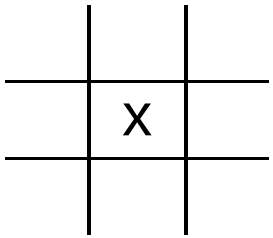


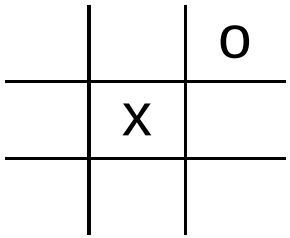


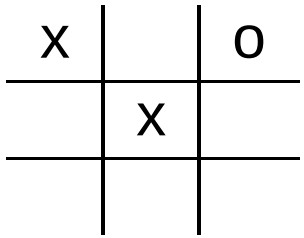


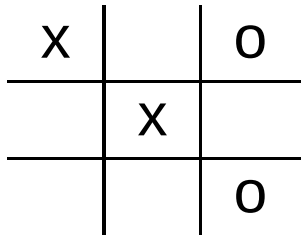
Playing the Game

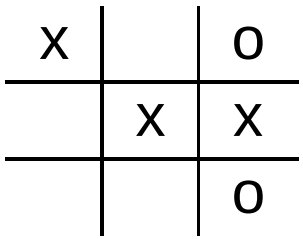
placing pebbles x and o

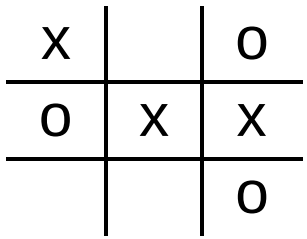


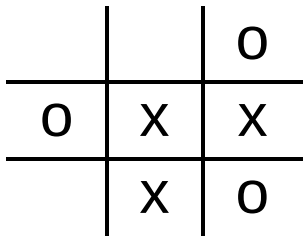


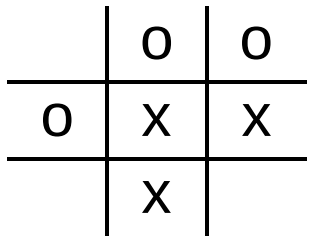


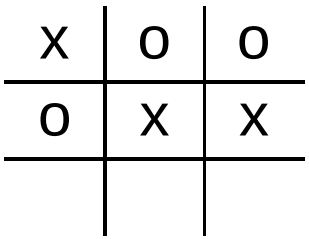


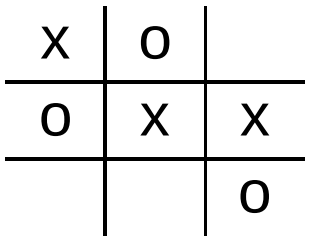


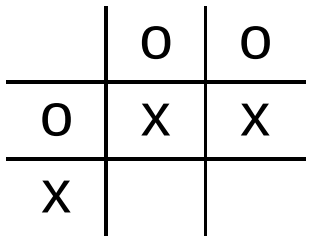


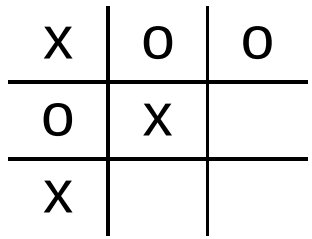












Game Trees

Pebbles on a Board

using numpy arrays

The board is a 3-by-3 matrix

→ `numpy` double arrays are convenient

A board is an integer matrix:

- 0: an empty space
- 1: first pebble x
- 2: second pebble o

Defining a class `Board`:

1. constructor: initialize board
2. string representation
3. parse string into a board

Pebbles on a Board

using numpy arrays

The board is a 3-by-3 matrix
$\rightarrow$ `numpy` double arrays are convenient

A board is an integer matrix:

- 0: an empty space
- 1: first pebble x
- 2: second pebble o

Defining a class `Board`:

1. constructor: initialize board
2. string representation
3. parse string into a board

Pebbles on a Board

using numpy arrays

The board is a 3-by-3 matrix
$\rightarrow$ `numpy` double arrays are convenient

A board is an integer matrix:

- 0: an empty space
- 1: first pebble x
- 2: second pebble o

Defining a class `Board`:

1. constructor: initialize board
2. string representation
3. parse string into a board

Pebbles on a Board

using numpy arrays

The board is a 3-by-3 matrix
$\rightarrow$ `numpy` double arrays are convenient

A board is an integer matrix:

- 0: an empty space
- 1: first pebble x
- 2: second pebble o

Defining a class `Board`:

1. constructor: initialize board
2. string representation
3. parse string into a board

Pebbles on a Board

using numpy arrays

The board is a 3-by-3 matrix
$\rightarrow$ `numpy` double arrays are convenient

A board is an integer matrix:

- 0: an empty space
- 1: first pebble x
- 2: second pebble o

Defining a class `Board`:

1. constructor: initialize board
2. string representation
3. parse string into a board

Pebbles on a Board

using numpy arrays

The board is a 3-by-3 matrix
$\rightarrow$ `numpy` double arrays are convenient

A board is an integer matrix:

- 0: an empty space
- 1: first pebble x
- 2: second pebble o

Defining a class `Board`:

1. constructor: initialize board
2. string representation
3. parse string into a board

Pebbles on a Board

using numpy arrays

The board is a 3-by-3 matrix
$\rightarrow$ `numpy` double arrays are convenient

A board is an integer matrix:

- 0: an empty space
- 1: first pebble x
- 2: second pebble o

Defining a class `Board`:

1. constructor: initialize board
2. string representation
3. parse string into a board

Working with Boards

```
>>> from classboard import *
>>> b = Board()
>>> b.m[1,2] = 1
>>> b.m[0,0] = 2
>>> b.m[1,1] = 1
>>> b.m[2,2] = 2
>>> b
O
 XX
  O
```

Working with Boards

```
>>> from classboard import *
>>> b = Board()
>>> b.m[1,2] = 1
>>> b.m[0,0] = 2
>>> b.m[1,1] = 1
>>> b.m[2,2] = 2
>>> b
2
  XX
    2
```

Working with Boards

```
>>> from classboard import *
>>> b = Board()
>>> b.m[1,2] = 1
>>> b.m[0,0] = 2
>>> b.m[1,1] = 1
>>> b.m[2,2] = 2
>>> b
O
 XX
  O
```

Representing Boards

```
>>> b
0
 XX
  0
>>> s = str(b)
>>> s
'0  \n XX\n  0'
>>> nb = Board()
>>> nb.parse(s)
>>> nb
0
 XX
  0
```

Representing Boards

```
>>> b
O
 XX
  O
>>> s = str(b)
>>> s
'O  \n XX\n  O'
>>> nb = Board()
>>> nb.parse(s)
>>> nb
O
 XX
  O
```

Representing Boards

```
>>> b
O
 XX
  O
>>> s = str(b)
>>> s
'O  \n XX\n  O'
>>> nb = Board()
>>> nb.parse(s)
>>> nb
O
 XX
  O
```

Representing Boards

```
>>> b
O
 XX
  O
>>> s = str(b)
>>> s
'O  \n XX\n  O'
>>> nb = Board()
>>> nb.parse(s)
>>> nb
O
 XX
  O
```

Game Trees

the Class Board

```
# L-39 MCS 275 Mon 21 Apr 2008 : classboard.py

# A board of pebbles in a tic tac toe game is
# represented by a 3-by-3 matrix.
# Its string representation uses X and O for
# the pebbles 1 and 2 respectively.
# The number of adjacent pebbles on a board
# determines the value of a board.

from numpy import *

class Board():
   """
   Exports a tic tac toe board.
   """
   def __init__(self):
      """
      Returns an empty board.
      """
      self.m = zeros((3,3),int)
```

the Class Board

```
# L-39 MCS 275 Mon 21 Apr 2008 : classboard.py

# A board of pebbles in a tic tac toe game is
# represented by a 3-by-3 matrix.
# Its string representation uses X and O for
# the pebbles 1 and 2 respectively.
# The number of adjacent pebbles on a board
# determines the value of a board.

from numpy import *

class Board():
   """
   Exports a tic tac toe board.
   """
   def __init__(self):
      """
      Returns an empty board.
      """
      self.m = zeros((3,3),int)
```

the Class Board

```
# L-39 MCS 275 Mon 21 Apr 2008 : classboard.py

# A board of pebbles in a tic tac toe game is
# represented by a 3-by-3 matrix.
# Its string representation uses X and O for
# the pebbles 1 and 2 respectively.
# The number of adjacent pebbles on a board
# determines the value of a board.

from numpy import *

class Board():
   """
   Exports a tic tac toe board.
   """
   def __init__(self):
      """
      Returns an empty board.
      """
      self.m = zeros((3,3),int)
```

Defining the String Representation

```
def __str__(self):
    """
    X is 1, O is 2,
    the rest is blank
    """
    s = ''
    for i in range(0,3):
        if i > 0: s = s + '\n'
        for j in range(0,3):
            if self.m[i,j] == 1:
                s = s + 'X'
            else:
                if self.m[i,j] == 2:
                    s = s + 'O'
                else:
                    s = s + ' '
    return s
```

Defining the String Representation

```
def __str__(self):
    """
    X is 1, O is 2,
    the rest is blank
    """
    s = ''
    for i in range(0,3):
        if i > 0: s = s + '\n'
        for j in range(0,3):
            if self.m[i,j] == 1:
                s = s + 'X'
            else:
                if self.m[i,j] == 2:
                    s = s + 'O'
                else:
                    s = s + ' '
    return s
```

Defining the String Representation

```
def __str__(self):
    """
    X is 1, O is 2,
    the rest is blank
    """
    s = ''
    for i in range(0,3):
        if i > 0: s = s + '\n'
        for j in range(0,3):
            if self.m[i,j] == 1:
                s = s + 'X'
            else:
                if self.m[i,j] == 2:
                    s = s + 'O'
                else:
                    s = s + ' '
    return s
```

Parsing Strings into Boards

```
def __repr__(self):
    """
    defines the representation of a board
    """
    return self.__str__()

def parse(self,s):
    """
    Converts a string into a board.
    """
    ind = 0
    for i in range(0,3):
        for j in range(0,3):
            if s[ind] == '\n': ind = ind + 1
            self.m[i,j] = 0
            if s[ind] == 'X': self.m[i,j] = 1
            if s[ind] == 'O': self.m[i,j] = 2
            ind = ind + 1
```

Parsing Strings into Boards

```
def __repr__(self):
    """
    defines the representation of a board
    """
    return self.__str__()

def parse(self,s):
    """
    Converts a string into a board.
    """
    ind = 0
    for i in range(0,3):
        for j in range(0,3):
            if s[ind] == '\n': ind = ind + 1
            self.m[i,j] = 0
            if s[ind] == 'X': self.m[i,j] = 1
            if s[ind] == 'O': self.m[i,j] = 2
            ind = ind + 1
```

Parsing Strings into Boards

```
def __repr__(self):
    """
    defines the representation of a board
    """
    return self.__str__()

def parse(self,s):
    """
    Converts a string into a board.
    """
    ind = 0
    for i in range(0,3):
        for j in range(0,3):
            if s[ind] == '\n': ind = ind + 1
            self.m[i,j] = 0
            if s[ind] == 'X': self.m[i,j] = 1
            if s[ind] == 'O': self.m[i,j] = 2
            ind = ind + 1
```

Game Trees

The Value of a Board

To compare which move is better,
we need to assign a value to a board.

Value of a board for a pebble
$\rightarrow$ count the number of pebbles lined up

Formula to evaluate a position:
number of own pebbles lined up
minus number of opponent pebbles lined up

The Value of a Board

To compare which move is better,
we need to assign a value to a board.

Value of a board for a pebble
$\rightarrow$ count the number of pebbles lined up

Formula to evaluate a position:
number of own pebbles lined up
minus number of opponent pebbles lined up

The Value of a Board

To compare which move is better,
we need to assign a value to a board.

Value of a board for a pebble
$\rightarrow$ count the number of pebbles lined up

Formula to evaluate a position:
number of own pebbles lined up
minus number of opponent pebbles lined up

Functions to Evaluate a Board

```
def Value2(self,T):
    """
    Returns 1 if the two tuples in T are
    not adjacent, returns 2 otherwise.
    T[0] and [1] are in lexicographical order.
    """
def Value3(self,T):
    """
    Returns 3 if three adjacent tuples,
    otherwise returns what the maximum
    of Value2 applied to all pairs in T.
    """
def Value(self,k):
    """
    The value of a board equals how
    many k's are next to each other.
    """
```

Functions to Evaluate a Board

```
def Value2(self,T):
    """
    Returns 1 if the two tuples in T are
    not adjacent, returns 2 otherwise.
    T[0] and [1] are in lexicographical order.
    """
def Value3(self,T):
    """
    Returns 3 if three adjacent tuples,
    otherwise returns what the maximum
    of Value2 applied to all pairs in T.
    """
def Value(self,k):
    """
    The value of a board equals how
    many k's are next to each other.
    """
```

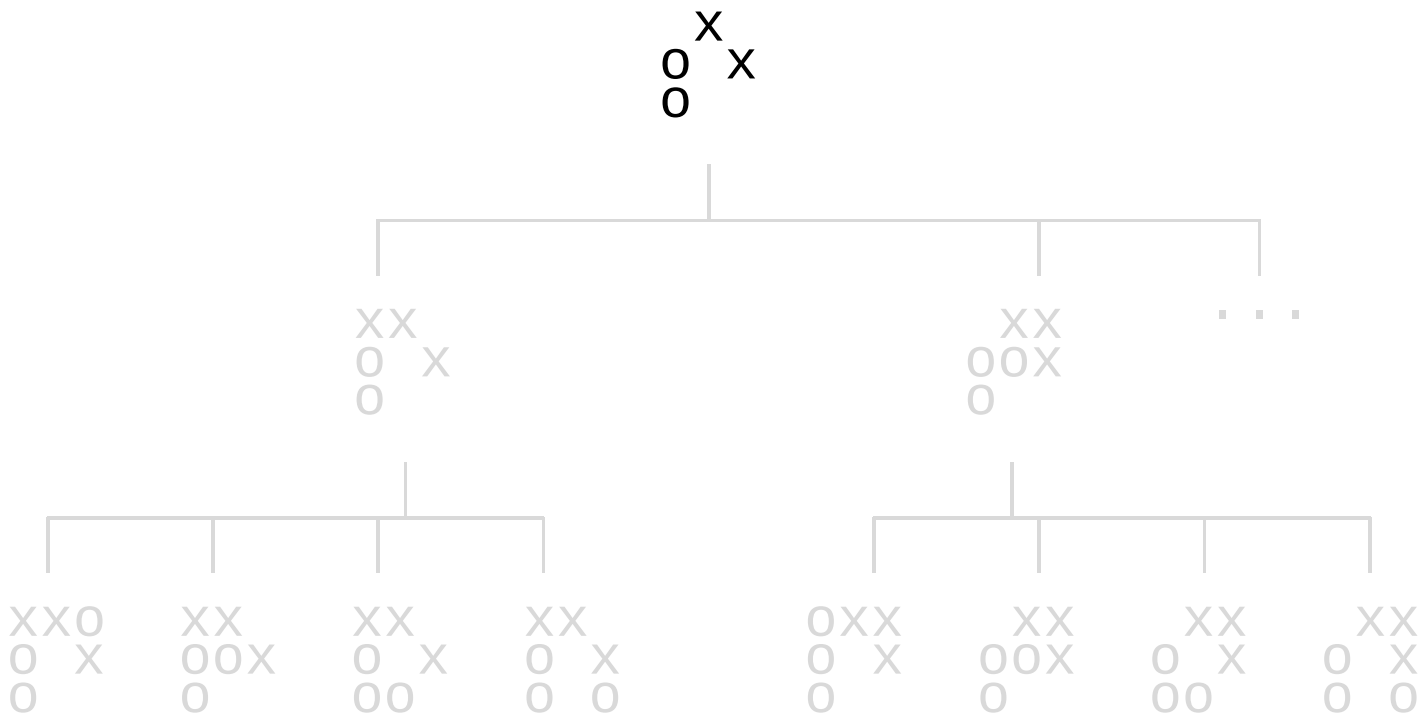
Functions to Evaluate a Board

```
def Value2(self,T):
    """
    Returns 1 if the two tuples in T are
    not adjacent, returns 2 otherwise.
    T[0] and [1] are in lexicographical order.
    """
def Value3(self,T):
    """
    Returns 3 if three adjacent tuples,
    otherwise returns what the maximum
    of Value2 applied to all pairs in T.
    """
def Value(self,k):
    """
    The value of a board equals how
    many k's are next to each other.
    """
```

Game Trees

An Example

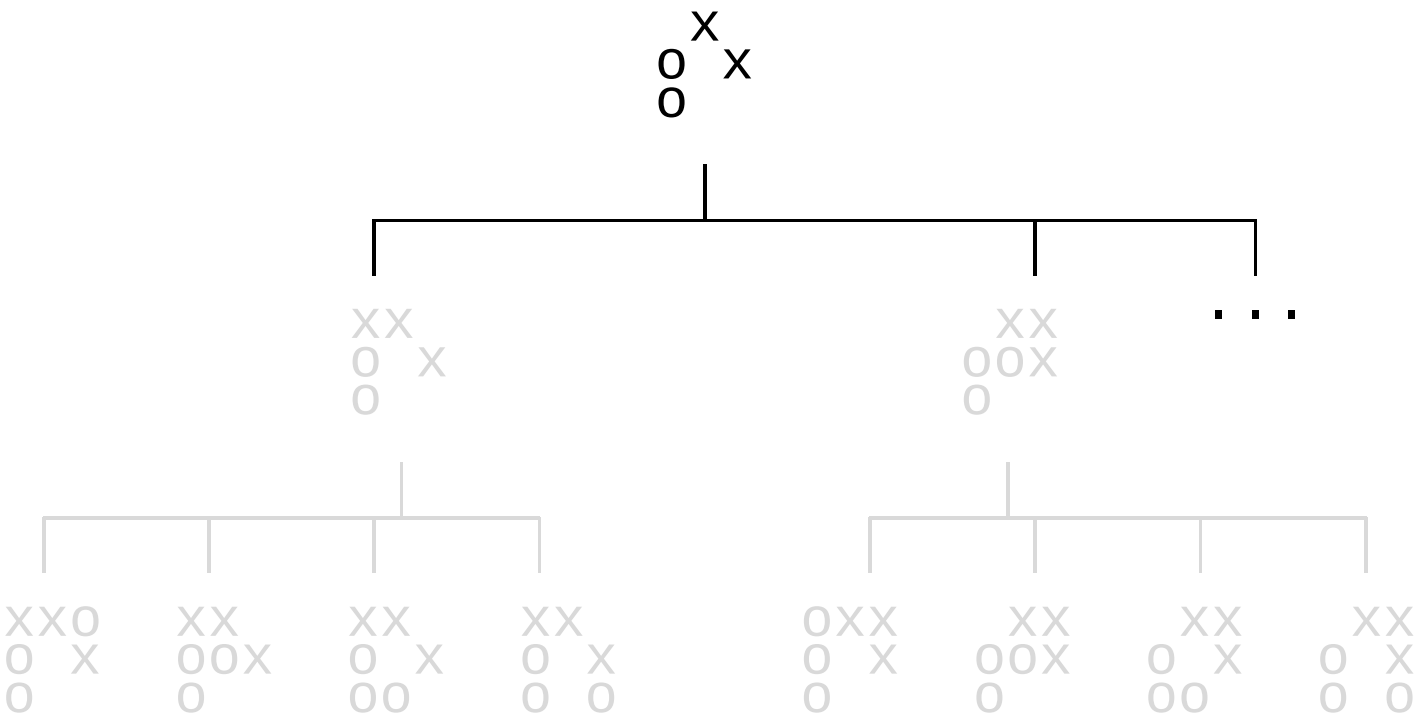
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

An Example

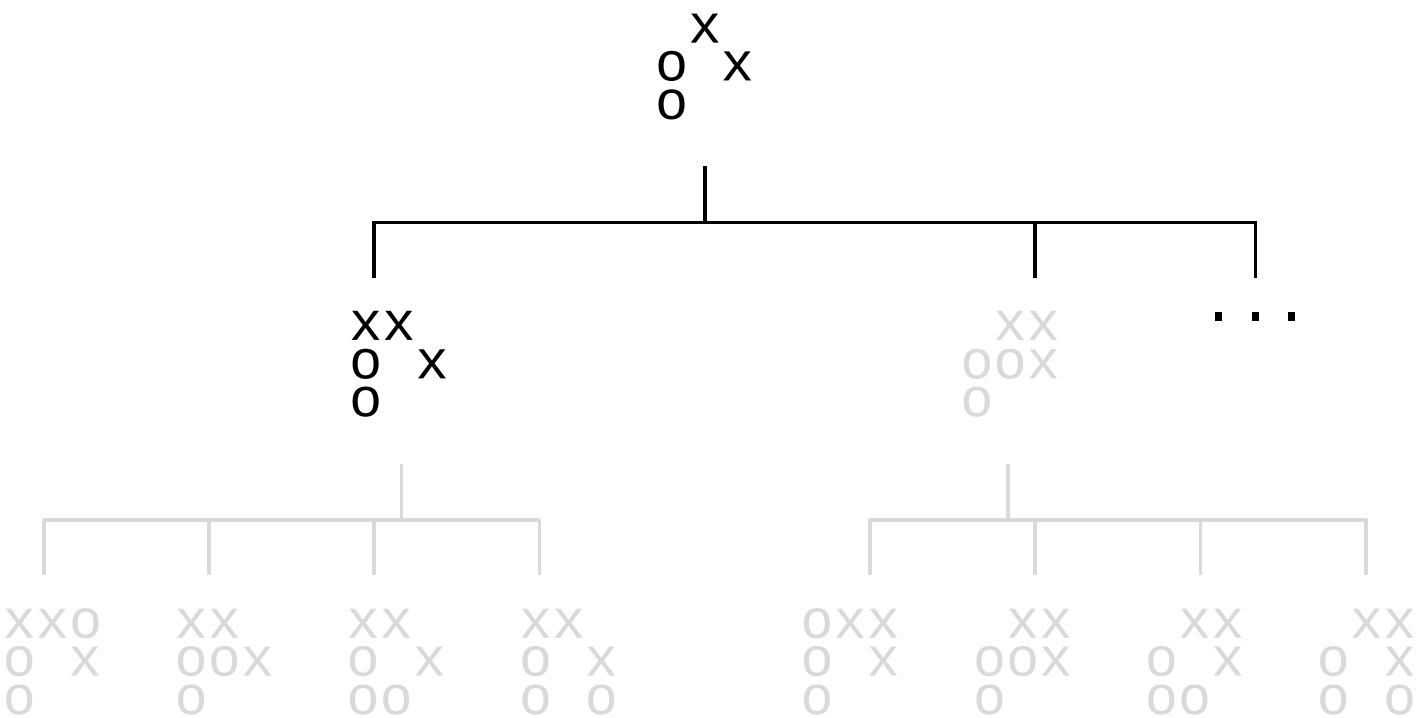
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

An Example

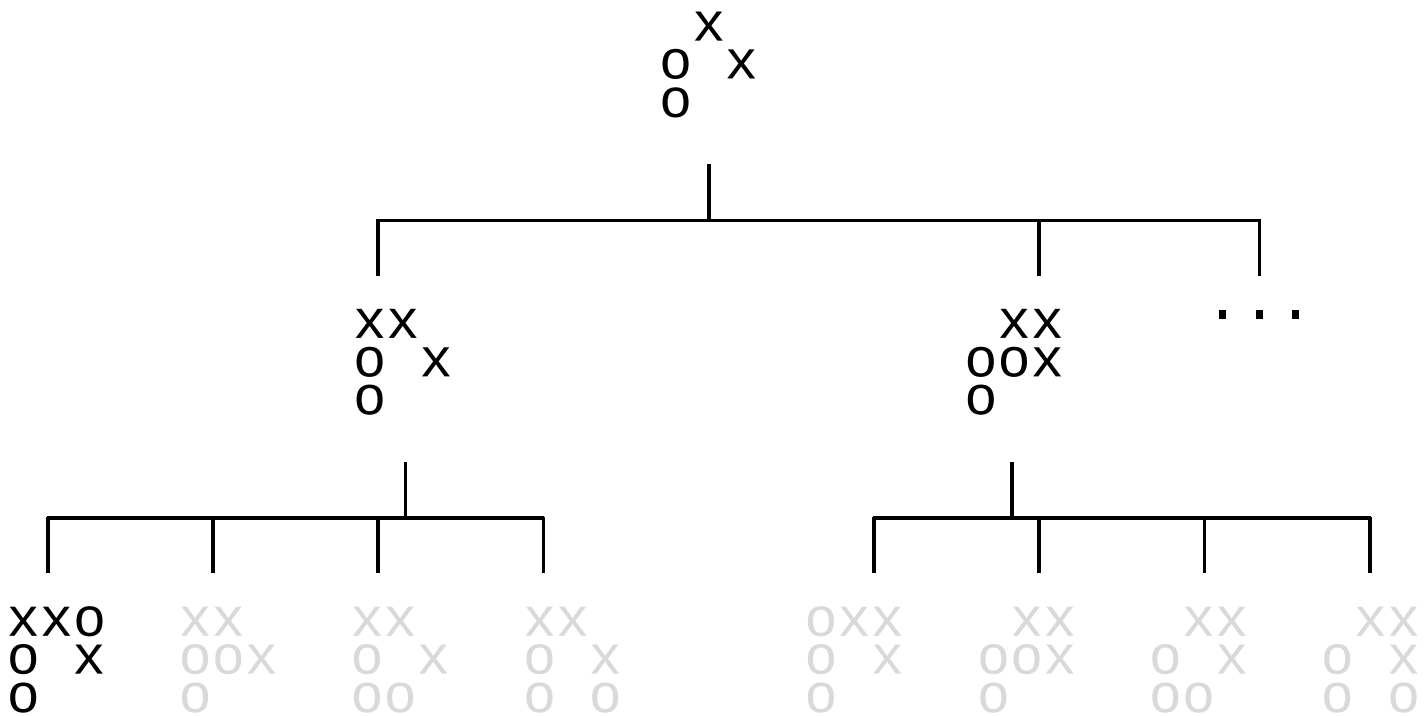
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

An Example

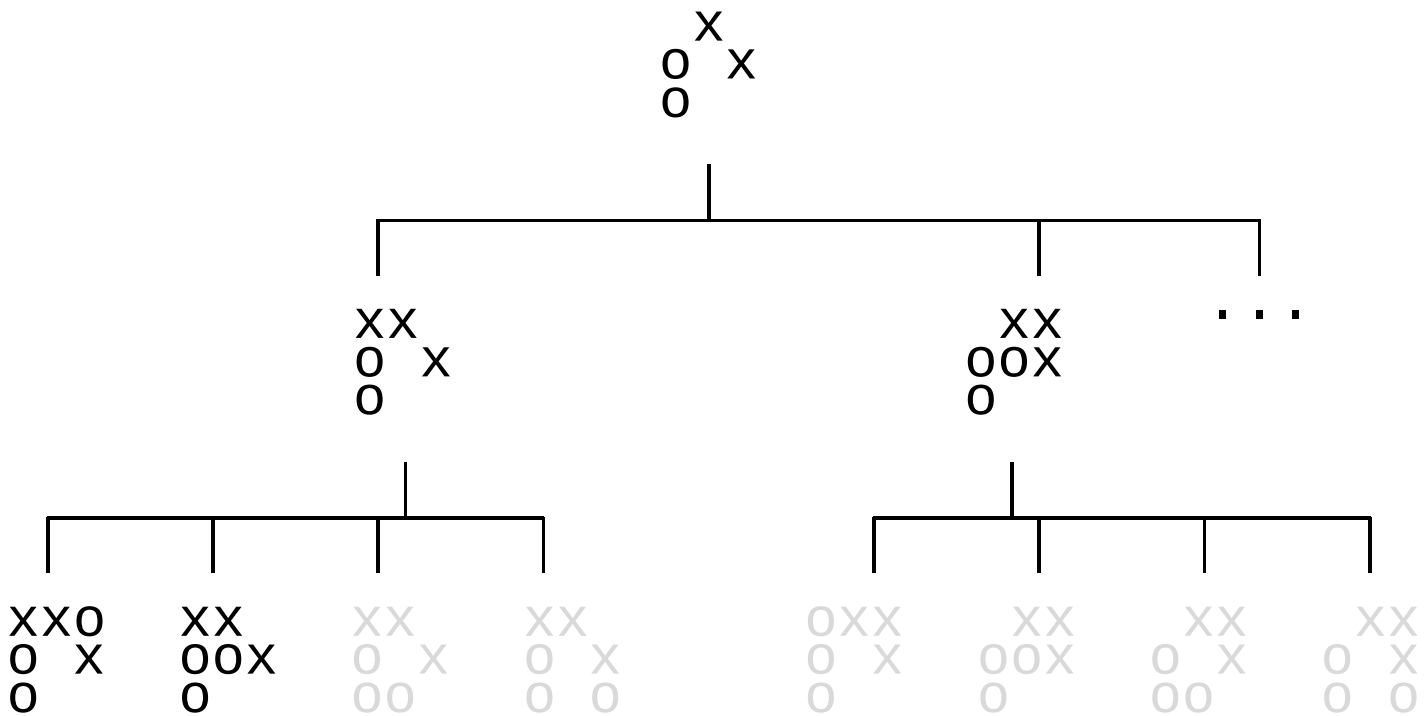
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

An Example

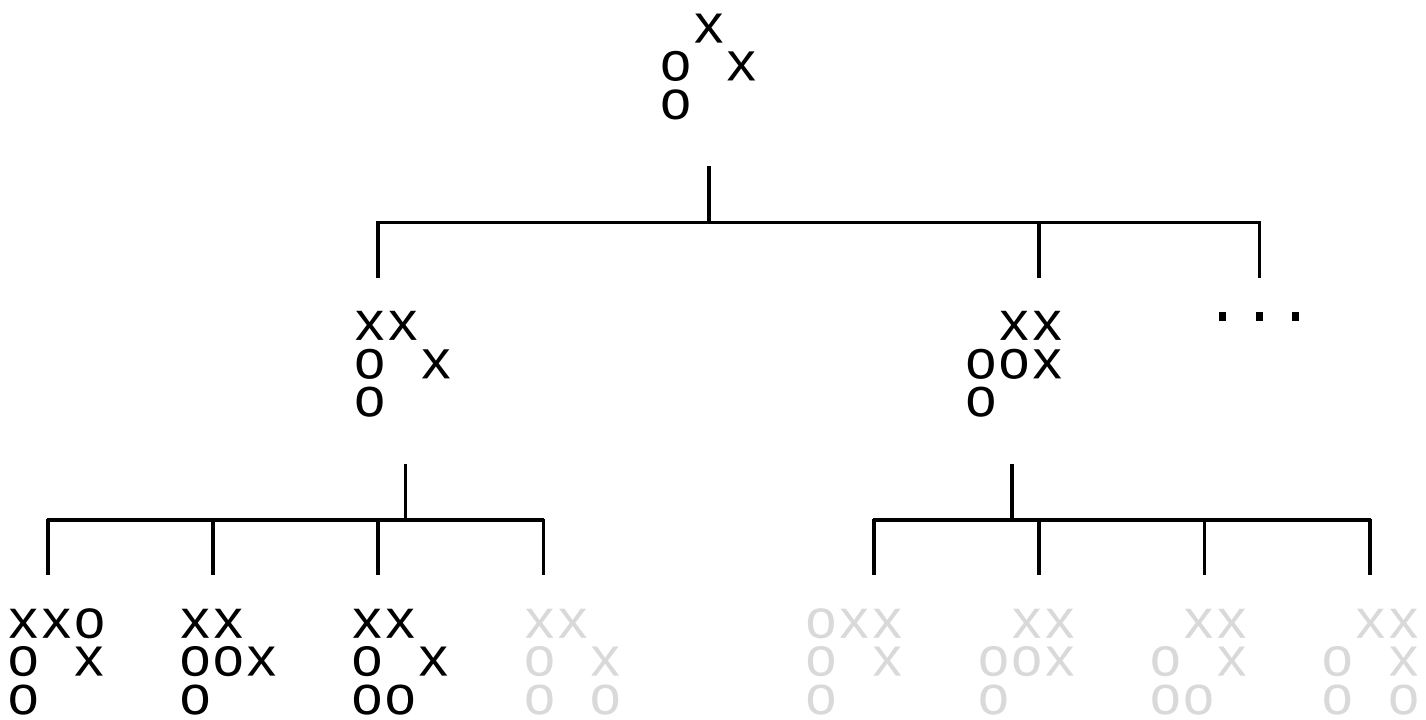
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

An Example

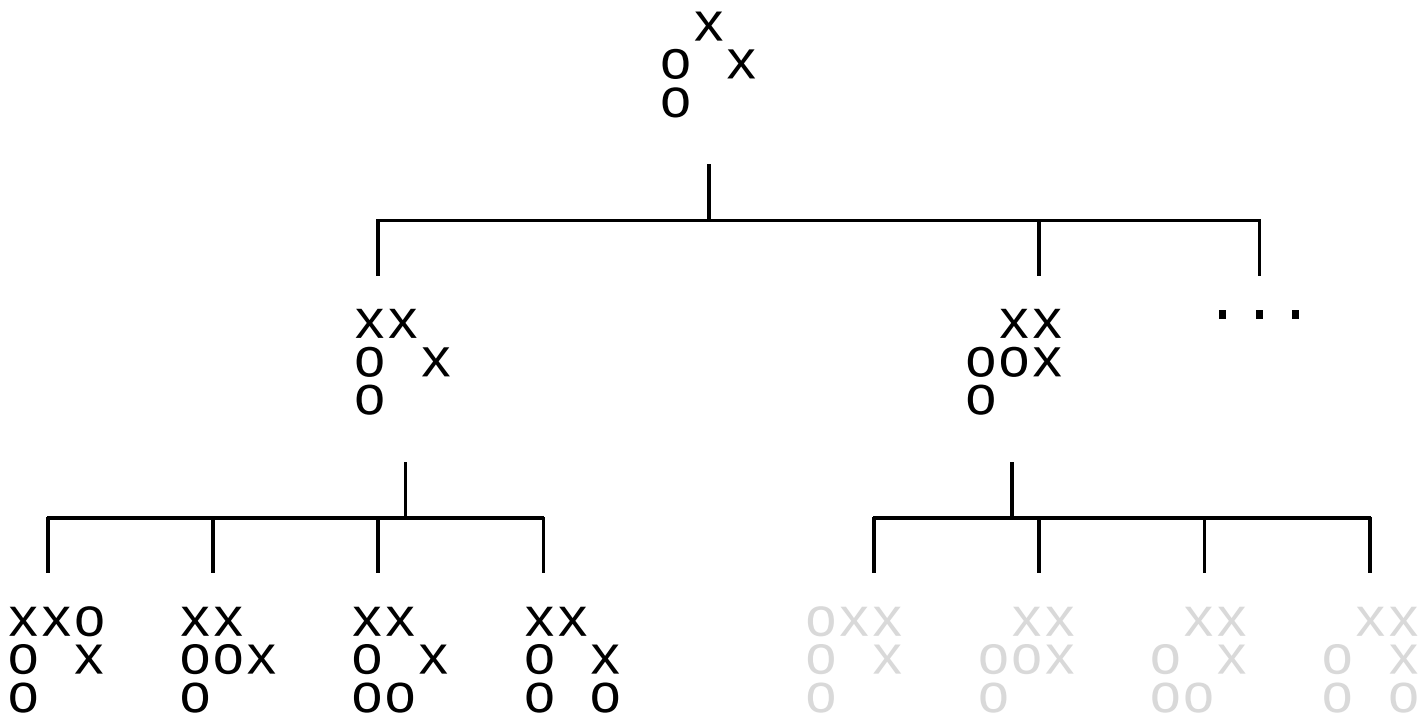
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

An Example

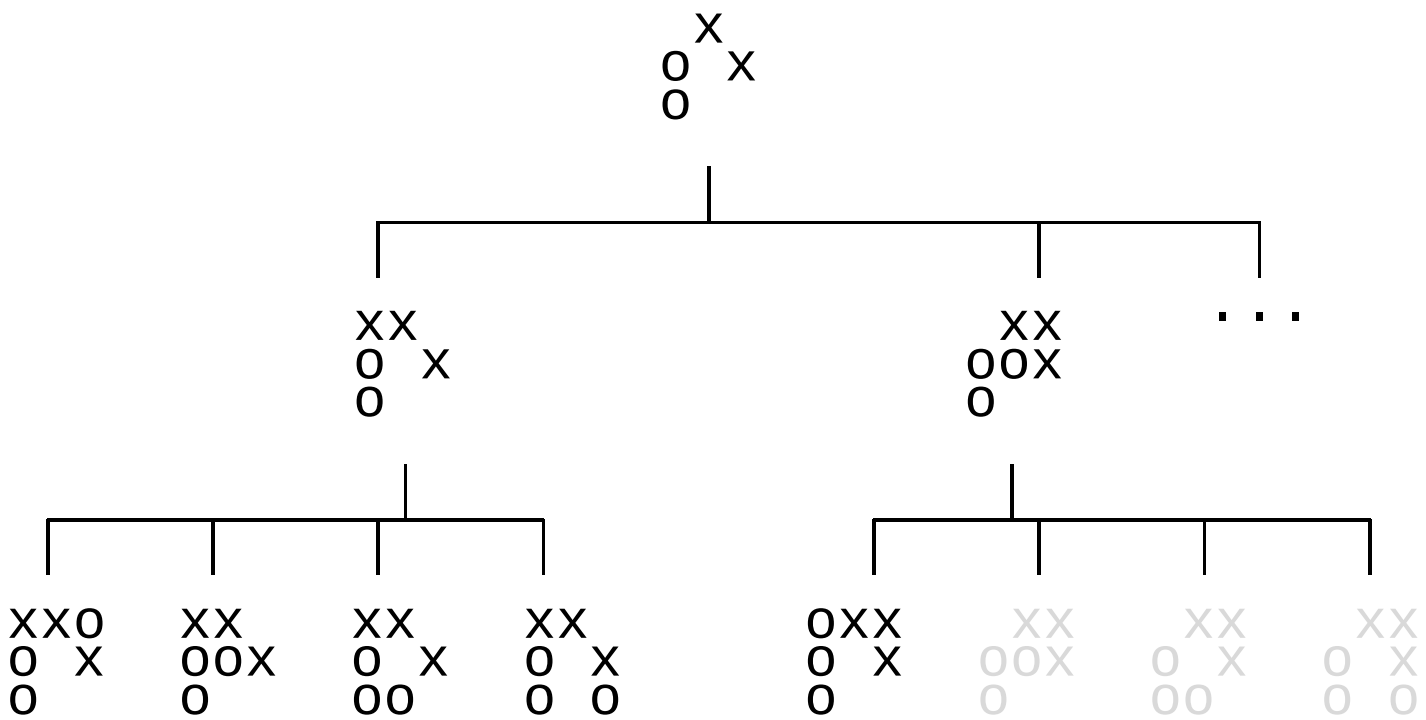
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

An Example

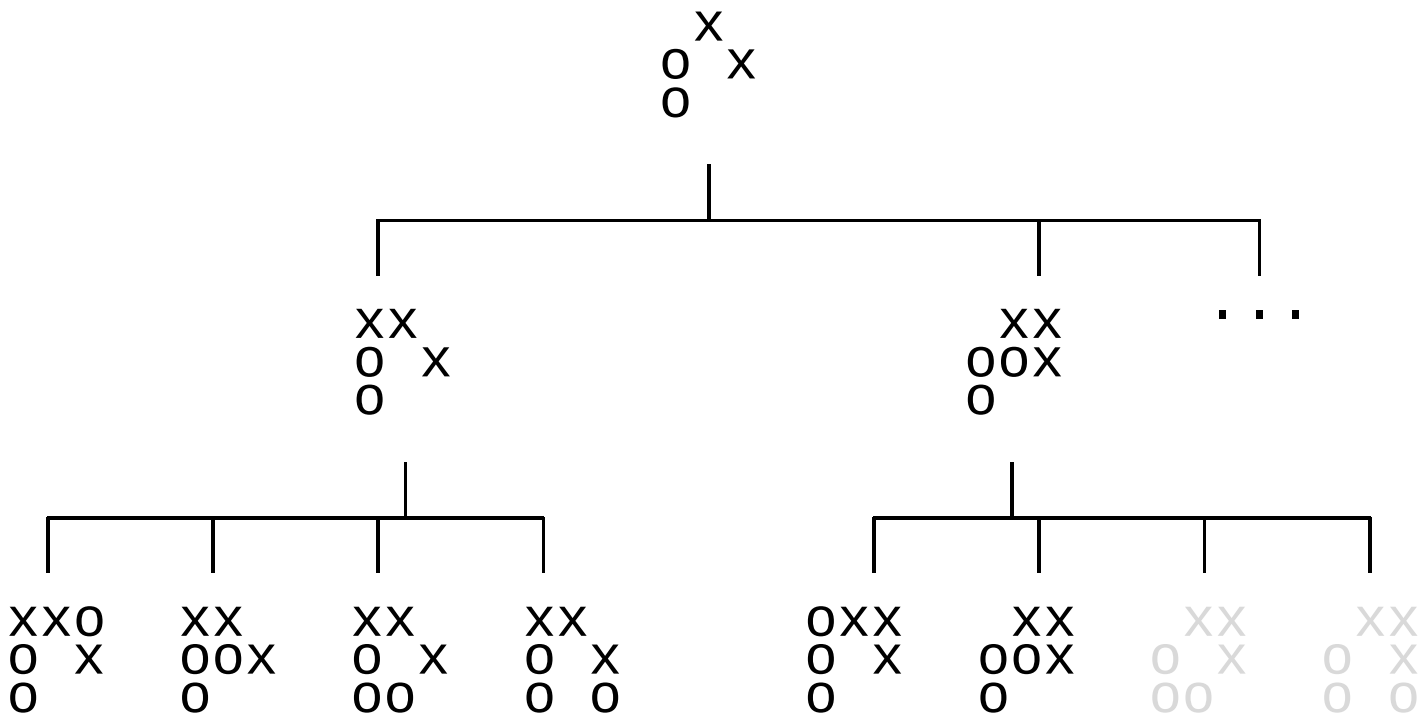
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

An Example

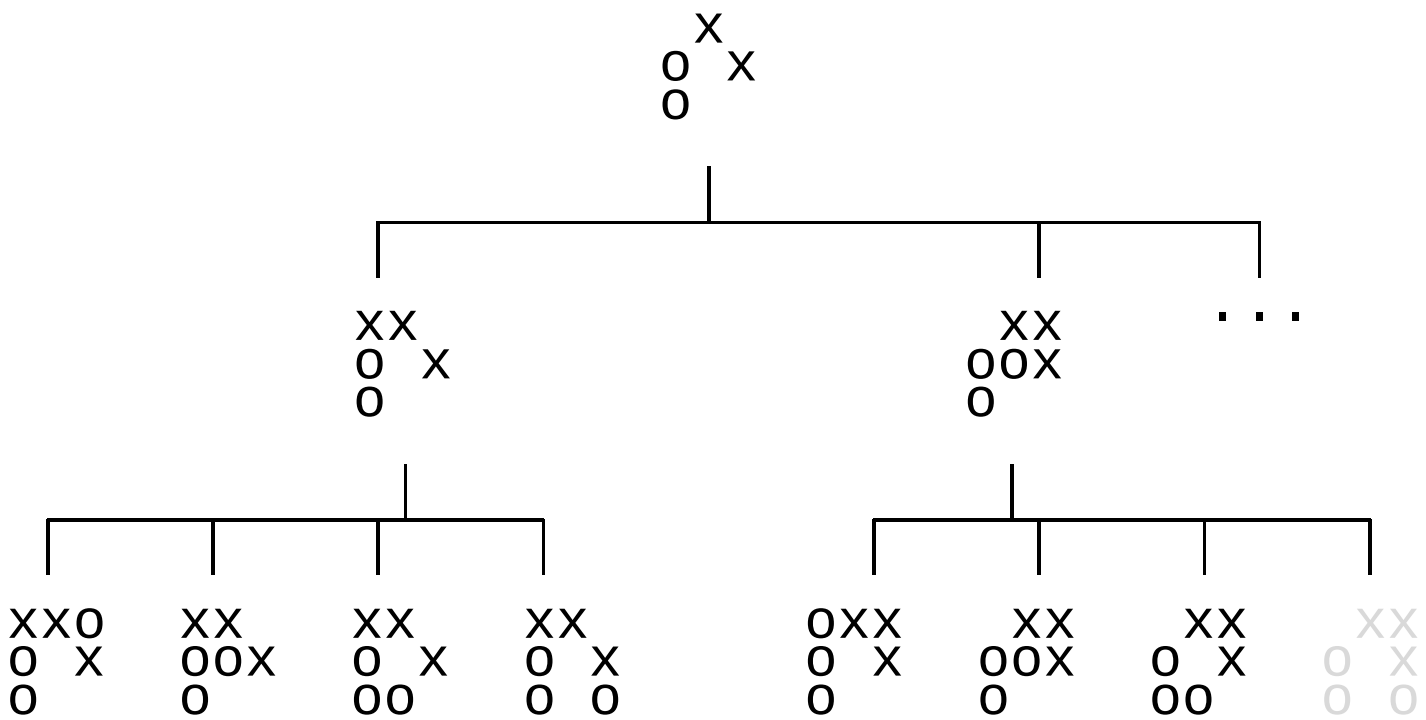
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

An Example

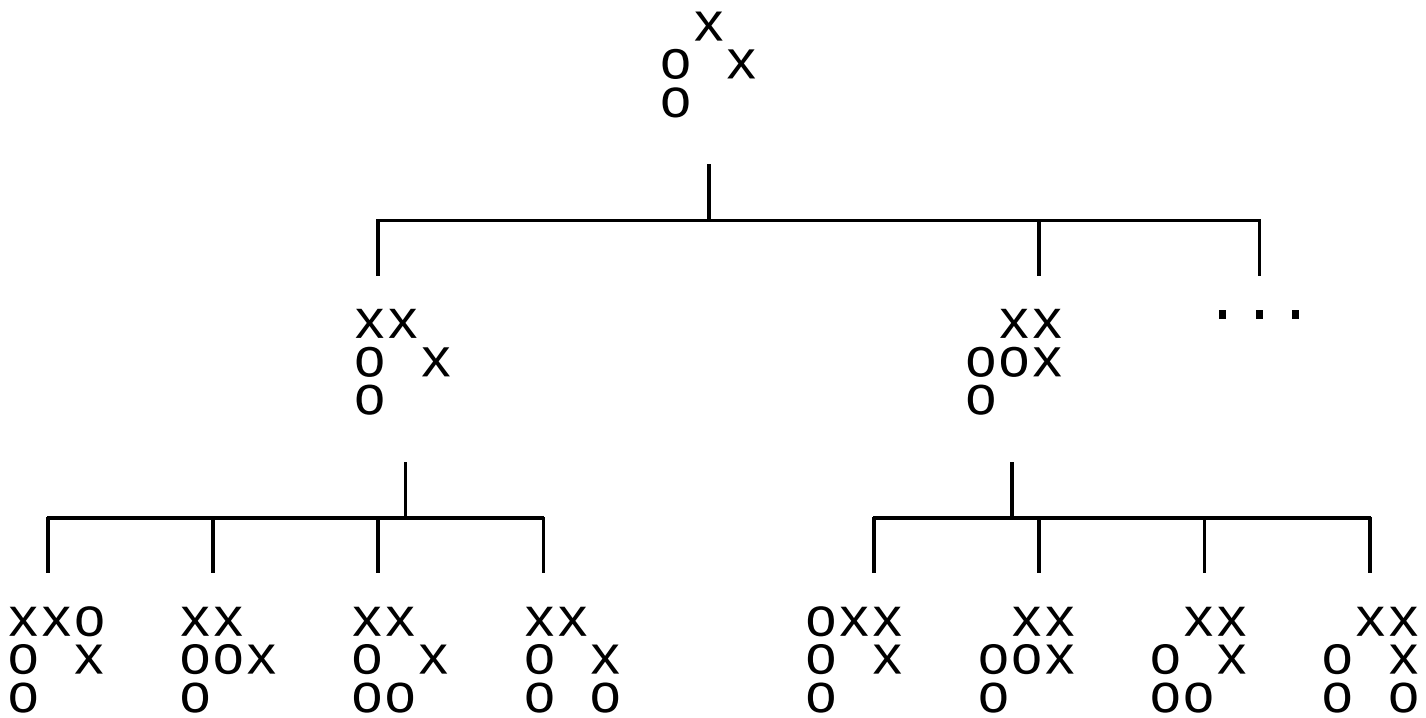
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

An Example

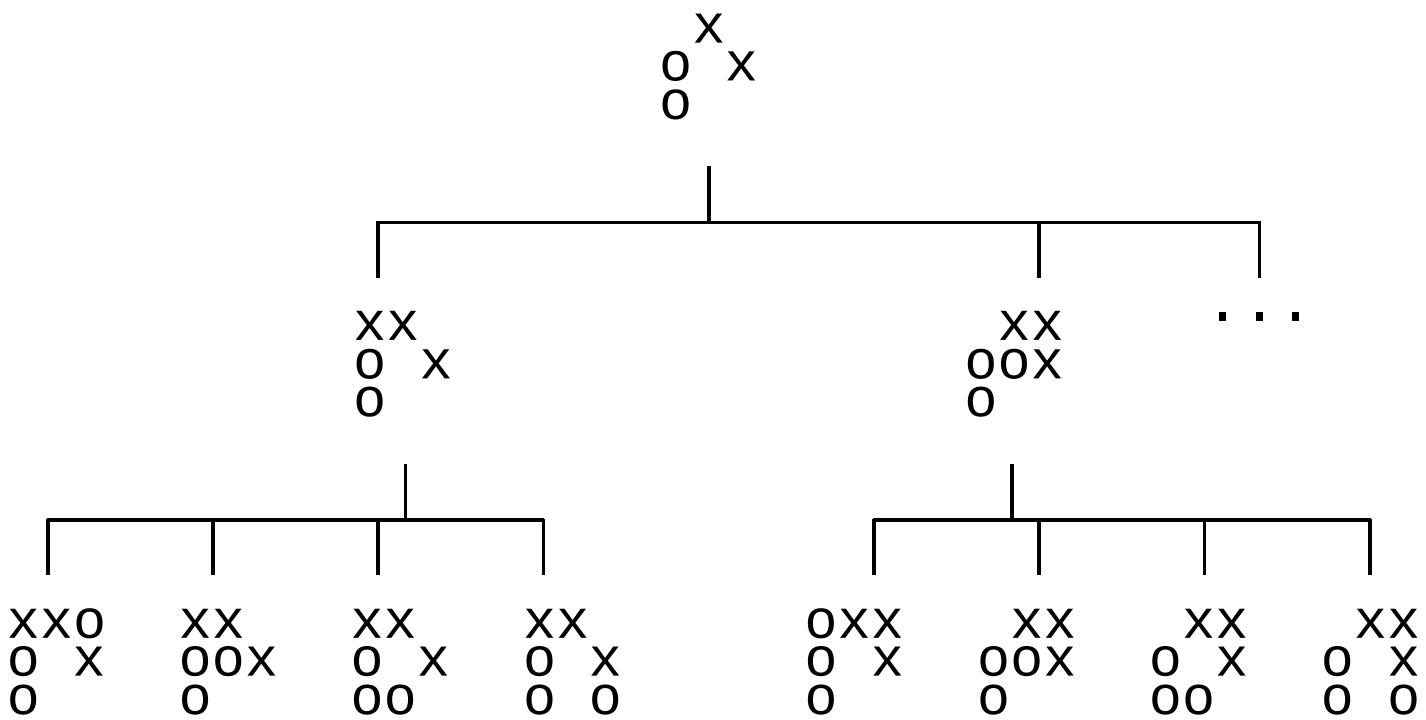
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

An Example

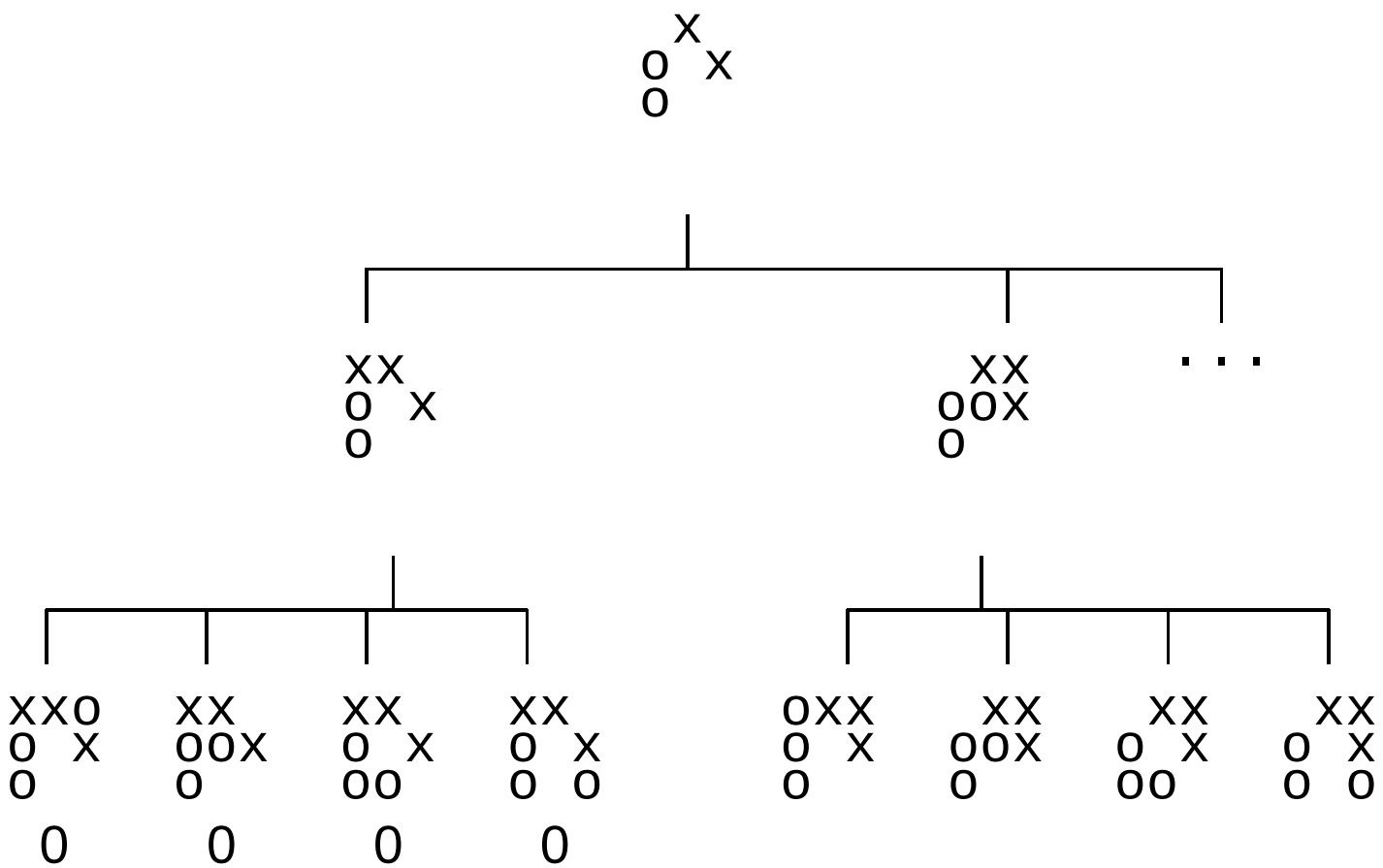
generating all moves



5 possibilities for x, 4 moves for o: 20 leaves

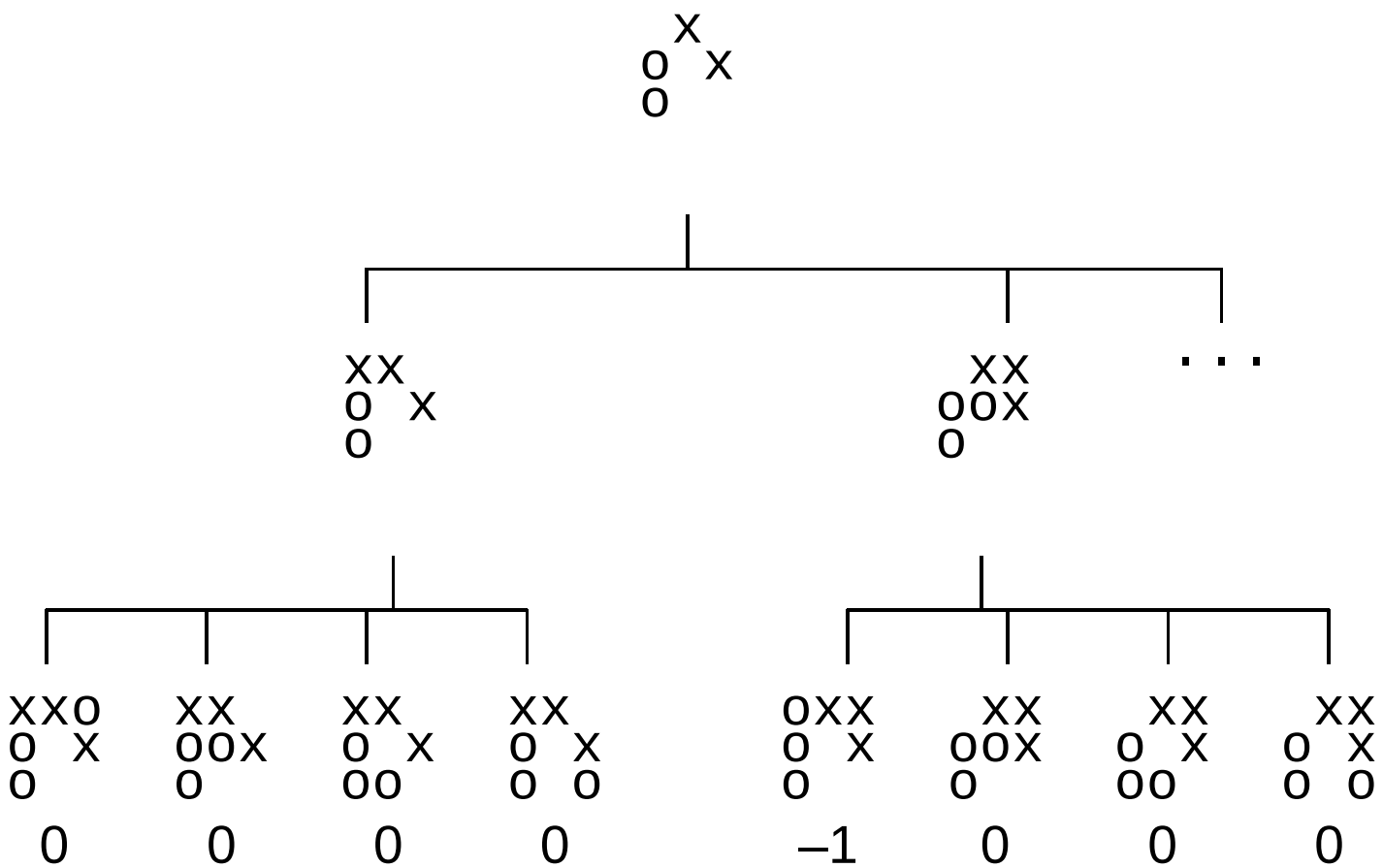
An Example

evaluating the positions



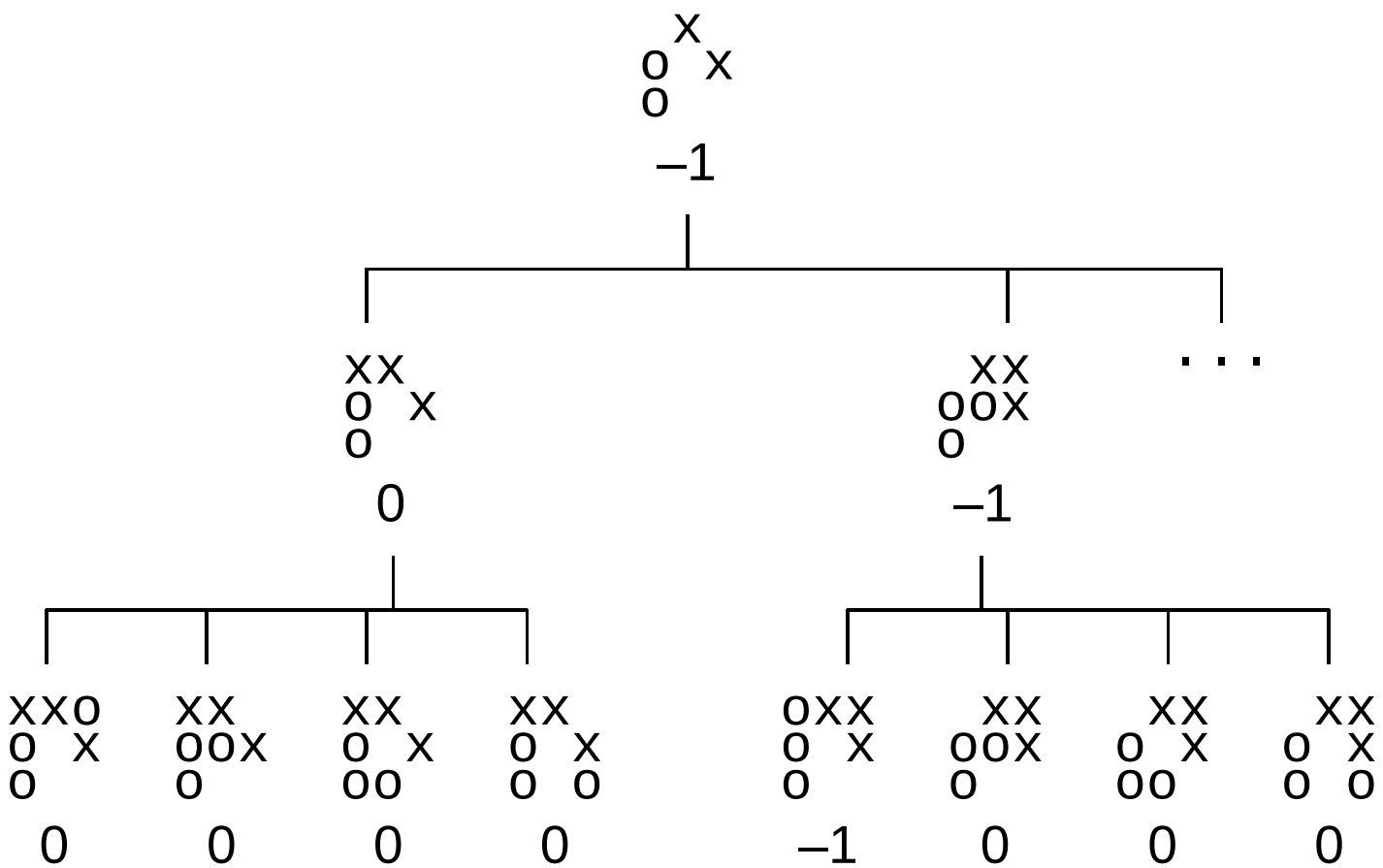
An Example

first all the leaves



An Example

looking ahead to compute the next move



Game Trees

the Class BoardTree

```
from classboard import *

class BoardTree():
    """
    Exports a tree of tic tac toe boards.
    """
    def __init__(self,b):
        """
        Returns a tree with as root the
        given board.
        """
        self.b = b
        self.v = 0
        self.c = []
        self.best = 0
```

String Representations

```
def __str__(self):
    """
    Defines the string representation
    of a tree of boards.
    """
    s = self.b.__str__()
    s = s + '\nvalue = %d' % self.v
    if len(self.c) > 0:
        s = s + '\nchildren :\n'
        for i in range(0,len(self.c)):
            s = s + '\n->child %d :\n' % i
            s = s + self.c[i].__str__()
    return s
def __repr__(self):
    """
    Takes the string representation as
    the representation of the tree.
    """
    return self.__str__()
```

String Representations

```
def __str__(self):
    """
    Defines the string representation
    of a tree of boards.
    """
    s = self.b.__str__()
    s = s + '\nvalue = %d' % self.v
    if len(self.c) > 0:
        s = s + '\nchildren :\n'
        for i in range(0,len(self.c)):
            s = s + '\n->child %d :\n' % i
            s = s + self.c[i].__str__()
    return s
def __repr__(self):
    """
    Takes the string representation as
    the representation of the tree.
    """
    return self.__str__()
```

String Representations

```
def __str__(self):
    """
    Defines the string representation
    of a tree of boards.
    """
    s = self.b.__str__()
    s = s + '\nvalue = %d' % self.v
    if len(self.c) > 0:
        s = s + '\nchildren :\n'
        for i in range(0,len(self.c)):
            s = s + '\n->child %d :\n' % i
            s = s + self.c[i].__str__()
    return s
def __repr__(self):
    """
    Takes the string representation as
    the representation of the tree.
    """
    return self.__str__()
```

String Representations

```
def __str__(self):
    """
    Defines the string representation
    of a tree of boards.
    """
    s = self.b.__str__()
    s = s + '\nvalue = %d' % self.v
    if len(self.c) > 0:
        s = s + '\nchildren :\n'
        for i in range(0,len(self.c)):
            s = s + '\n->child %d :\n' % i
            s = s + self.c[i].__str__()
    return s
def __repr__(self):
    """
    Takes the string representation as
    the representation of the tree.
    """
    return self.__str__()
```

Generating the Moves

```
def moves(self,k,n):
    """
    Defines the children via all moves
    originating at self.b for pebble k,
    using n stages.
    """
    for i in range(0,3):
       for j in range(0,3):
          if self.b.m[i,j] == 0:
             nb = Board()
             nb.copyBoard(self.b.m)
             nt = BoardTree(nb)
             nt.b.m[i,j] = k
             if n > 1:
                nk = k % 2 + 1
                nt.moves(nk,n-1)
             self.c.append(nt)
```

Generating the Moves

```
def moves(self,k,n):
    """
    Defines the children via all moves
    originating at self.b for pebble k,
    using n stages.
    """
    for i in range(0,3):
        for j in range(0,3):
            if self.b.m[i,j] == 0:
                nb = Board()
                nb.copyBoard(self.b.m)
                nt = BoardTree(nb)
                nt.b.m[i,j] = k
                if n > 1:
                    nk = k % 2 + 1
                    nt.moves(nk,n-1)
                self.c.append(nt)
```

Generating the Moves

```
def moves(self,k,n):
    """
    Defines the children via all moves
    originating at self.b for pebble k,
    using n stages.
    """
    for i in range(0,3):
        for j in range(0,3):
            if self.b.m[i,j] == 0:
                nb = Board()
                nb.copyBoard(self.b.m)
                nt = BoardTree(nb)
                nt.b.m[i,j] = k
                if n > 1:
                    nk = k % 2 + 1
                    nt.moves(nk,n-1)
                self.c.append(nt)
```

Generating the Moves

```
def moves(self,k,n):
    """
    Defines the children via all moves
    originating at self.b for pebble k,
    using n stages.
    """
    for i in range(0,3):
        for j in range(0,3):
            if self.b.m[i,j] == 0:
                nb = Board()
                nb.copyBoard(self.b.m)
                nt = BoardTree(nb)
                nt.b.m[i,j] = k
                if n > 1:
                    nk = k % 2 + 1
                    nt.moves(nk,n-1)
                self.c.append(nt)
```

Evaluating the Moves

the base case

The computation starts at the leaves:

```
def values(self,k):
   """
   Assigns values to all the nodes
   in the tree of boards.
   """
   if len(self.c) == 0:
      ck = k % 2 + 1
      self.v = self.b.Value(k) - self.b.Value(ck)
```

Evaluating the Moves continued

```
else:
   for i in range(0,len(self.c)):
      self.c[i].values(k)
   max = self.c[0].v
   self.best = 0
   if max < 0:
      self.v = max
   else:
      for i in range(1,len(self.c)):
         if self.c[i].v < 0:
            max = self.c[i].v
         if max >= 0 and self.c[i].v > max:
            max = self.c[i].v
            self.best = i
      self.v = max
```

Evaluating the Moves continued

```
else:
   for i in range(0,len(self.c)):
      self.c[i].values(k)
   max = self.c[0].v
   self.best = 0
   if max < 0:
      self.v = max
   else:
      for i in range(1,len(self.c)):
         if self.c[i].v < 0:
            max = self.c[i].v
         if max >= 0 and self.c[i].v > max:
            max = self.c[i].v
            self.best = i
      self.v = max
```

Game Trees

Calculating Game Trees

running `gametree.py`

```
$ python gametree.py
give three characters per row
-> either X, O, or a space
give row 1 :  X
give row 2 : O X
give row 3 : O
 X
O X
O
[[0 1 0]
 [2 0 1]
 [2 0 0]]
player ? (1 or 2)
```

Calculating Game Trees

running `gametree.py`

```
$ python gametree.py
give three characters per row
-> either X, O, or a space
give row 1 :  X
give row 2 : O X
give row 3 : O
 X
O X
O
[[0 1 0]
 [2 0 1]
 [2 0 0]]
player ? (1 or 2)
```

Calculating Game Trees

running `gametree.py`

```
$ python gametree.py
give three characters per row
-> either X, O, or a space
give row 1 :  X
give row 2 : O X
give row 3 : O
 X
O X
O
[[0 1 0]
 [2 0 1]
 [2 0 0]]
player ? (1 or 2)
```

continued with the run

```
player ? (1 or 2) 1
 X
O X
O
value = -1
children :
->child 0 :
XX
O X
O
value = 0
children :
->child 0 :
XXO
O X
O
value = 0
...
```

continued with the run

```
player ? (1 or 2) 1
 X
O X
O
value = -1

children :
->child 0 :
XX
O X
O
value = 0

children :
->child 0 :
XXO
O X
O
value = 0
...
```

continued with the run

```
player ? (1 or 2) 1
 X
O X
O
value = -1

children :
->child 0 :
XX
O X
O
value = 0

children :
->child 0 :
XXO
O X
O
value = 0
...
```

the end of the run

```
...
->child 2 :
 X
OOX
O X
value = 0
->child 3 :
 X
O X
OOX
value = 0

recommended move :
XX
O X
O
```

the end of the run

```
...
->child 2 :
 X
OOX
O X
value = 0
->child 3 :
 X
O X
OOX
value = 0

recommended move :
XX
O X
O
```

Summary + Assignments

We considered an example of trees.
Assignments:

1. Develop a GUI for tic tac toe.
2. Use sockets for two players on remote computers.
3. Consider four in a row.