CPSC 322
Introduction to Artificial Intelligence

October 20, 2004

Things...

No office hours today...will hold them Friday from 3:30 to 4:30 pm

Bring me your first midterms before the second midterm if you want to be retested on problem 3

Two obstacles to overcome

For really interesting (i.e., big) problems, we need to find ways to...

1. reduce the amount of time spent searching
   heuristic search
2. reduce the amount of time spent pre-building the graph
   generate the graph as you need it

By the way, those reductions have to be significant (i.e., big)

Move generation

```
move([0,X,C1,C2,C3,C4,C5,C6,C7],[X,0,C1,C2,C3,C4,C5,C6,C7]).
move([0,C1,C2,X,C3,C4,C5,C6,C7],[X,C1,C2,0,C3,C4,C5,C6,C7]).

/* ... and so on ... */

/* might be more useful as... */

move([0,X,C1,Y,C2,C3,C4,C5,C6],[[X,0,C1,Y,C2,C3,C4,C5,C6],
                                [Y,X,C1,0,C2,C3,C4,C5,C6]]).
```

More to come Wednesday...

add path collection

add the ability to beat the world’s best chess players

Why return paths?

Knowing that we can get from point A to point B is nice, but it doesn't tell us how to get from point A to point B.

The search algorithm does most of the work already, but it needs to keep track of the paths to the frontier nodes, not just the frontier nodes themselves.

Returning a path

goal: j

$$\begin{tikzcd}
 & & & & & & & a \arrow[dllll] \arrow[drrrr] & & & & & & & \\
 & & & b \arrow[dll] \arrow[drr] & & & & & & & & c \arrow[dll] \arrow[drr] & & & \\
 & d \arrow[dl] \arrow[dr] & & & & e \arrow[dl] \arrow[dr] & & & & f \arrow[dl] \arrow[dr] & & & & g \arrow[dl] \arrow[dr] & \\
h & & i & & j & & k & & l & & m & & n & & o
\end{tikzcd}$$

frontier:[[a]]

path being considered:

goal: j

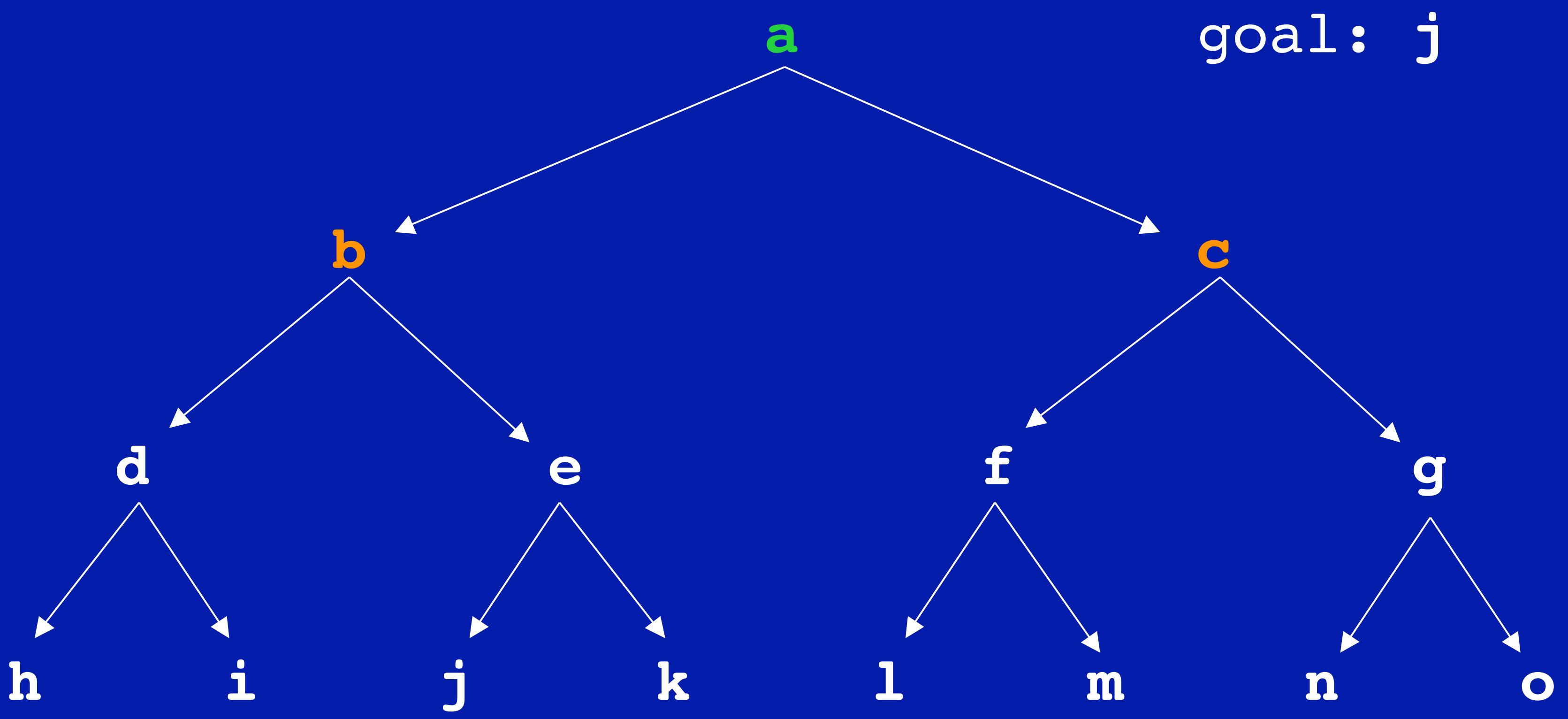
$$\begin{tikzcd}
& & & & & & & a \arrow[dllll] \arrow[drrrr] & & & & & & & \\
& & & b \arrow[dll] \arrow[drr] & & & & & & & & c \arrow[dll] \arrow[drr] & & & \\
& d \arrow[dl] \arrow[dr] & & & & e \arrow[dl] \arrow[dr] & & & & f \arrow[dl] \arrow[dr] & & & & g \arrow[dl] \arrow[dr] & \\
h & & i & & j & & k & & l & & m & & n & & o
\end{tikzcd}$$

```
frontier:[ ]
path being considered:[a]
```

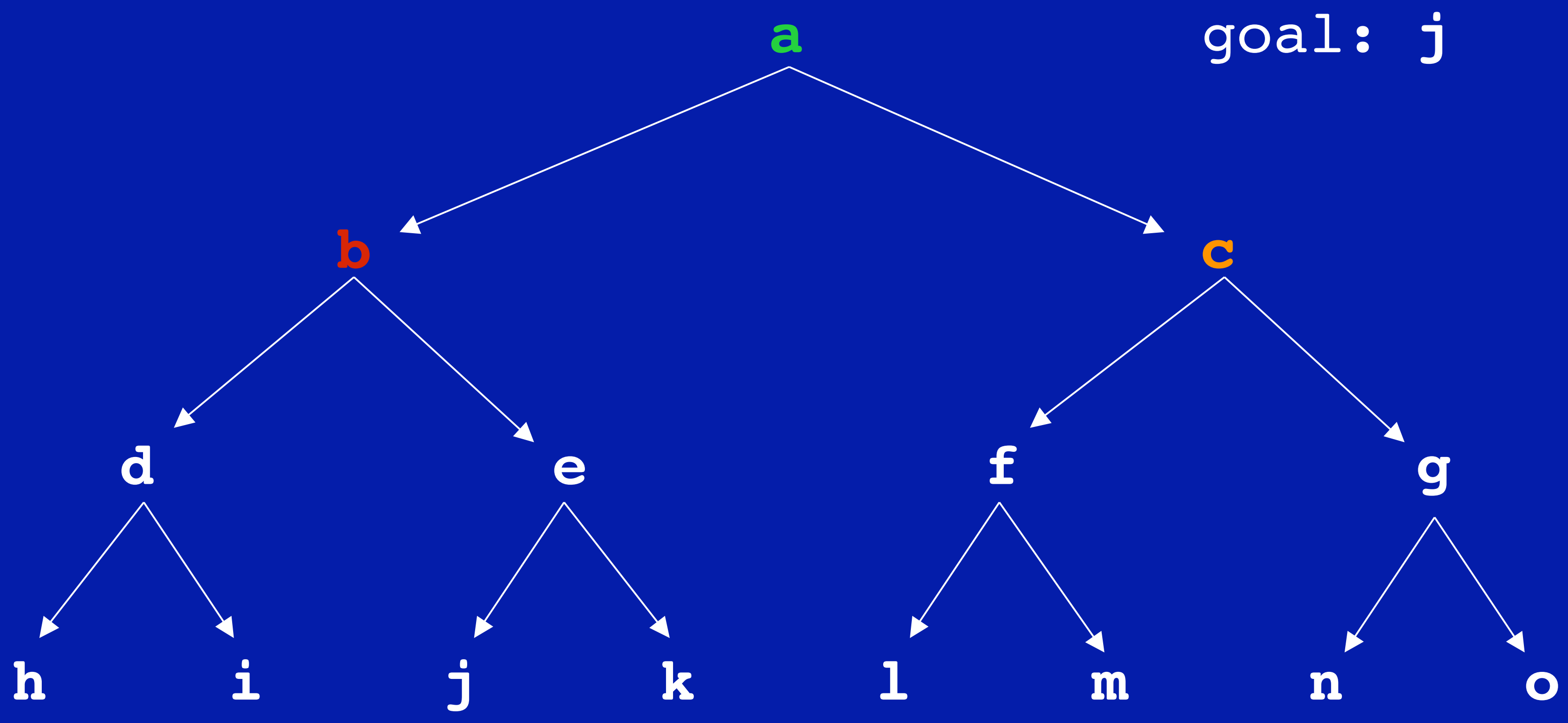
goal: j

$$\begin{tikzcd}
& & & & & & & a \arrow[dllll] \arrow[drrrr] \\
& & & b \arrow[dll] \arrow[drr] & & & & & & & & c \arrow[dll] \arrow[drr] \\
& d \arrow[dl] \arrow[dr] & & & & e \arrow[dl] \arrow[dr] & & & & f \arrow[dl] \arrow[dr] & & & & g \arrow[dl] \arrow[dr] \\
h & & i & & j & & k & & l & & m & & n & & o
\end{tikzcd}$$

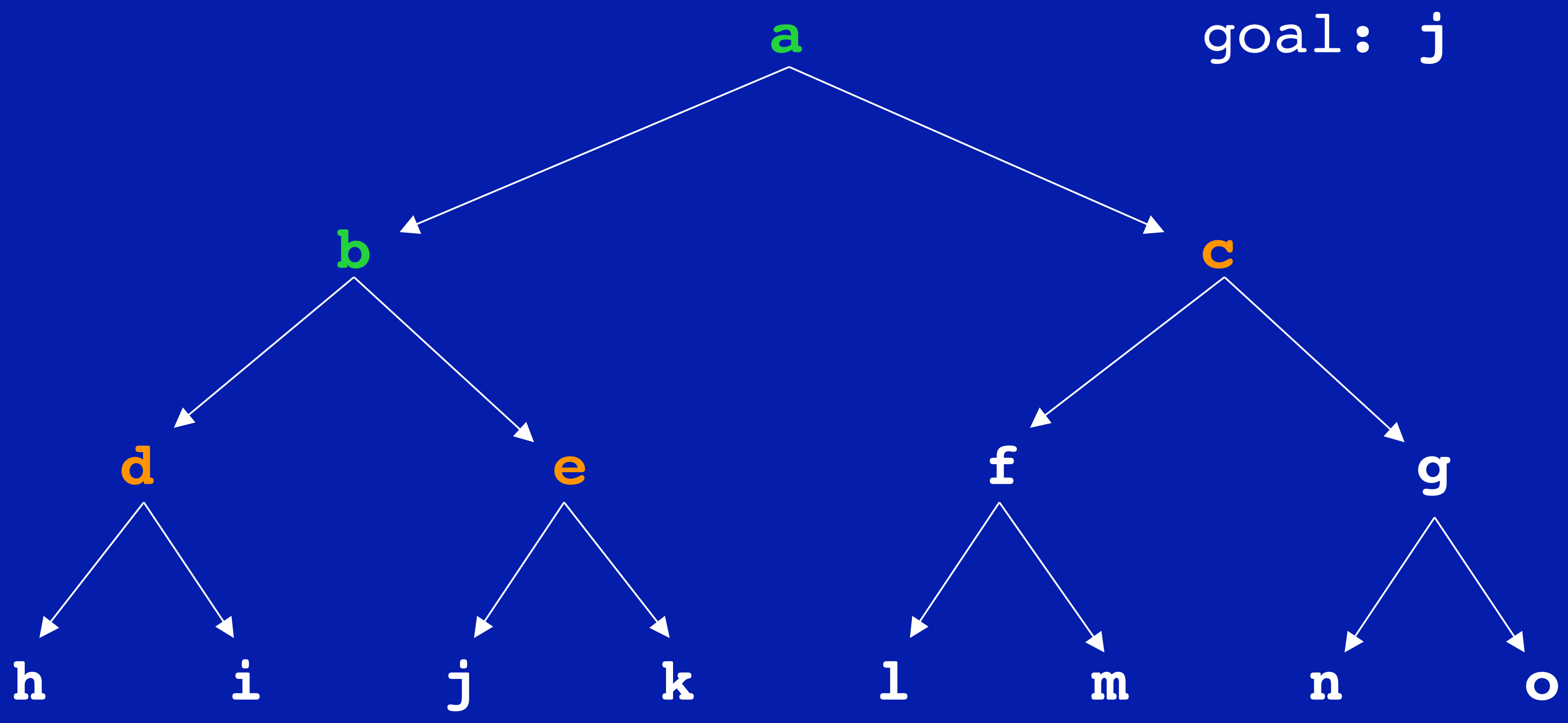
```
frontier:[[b,a],[c,a]]
path being considered:
```



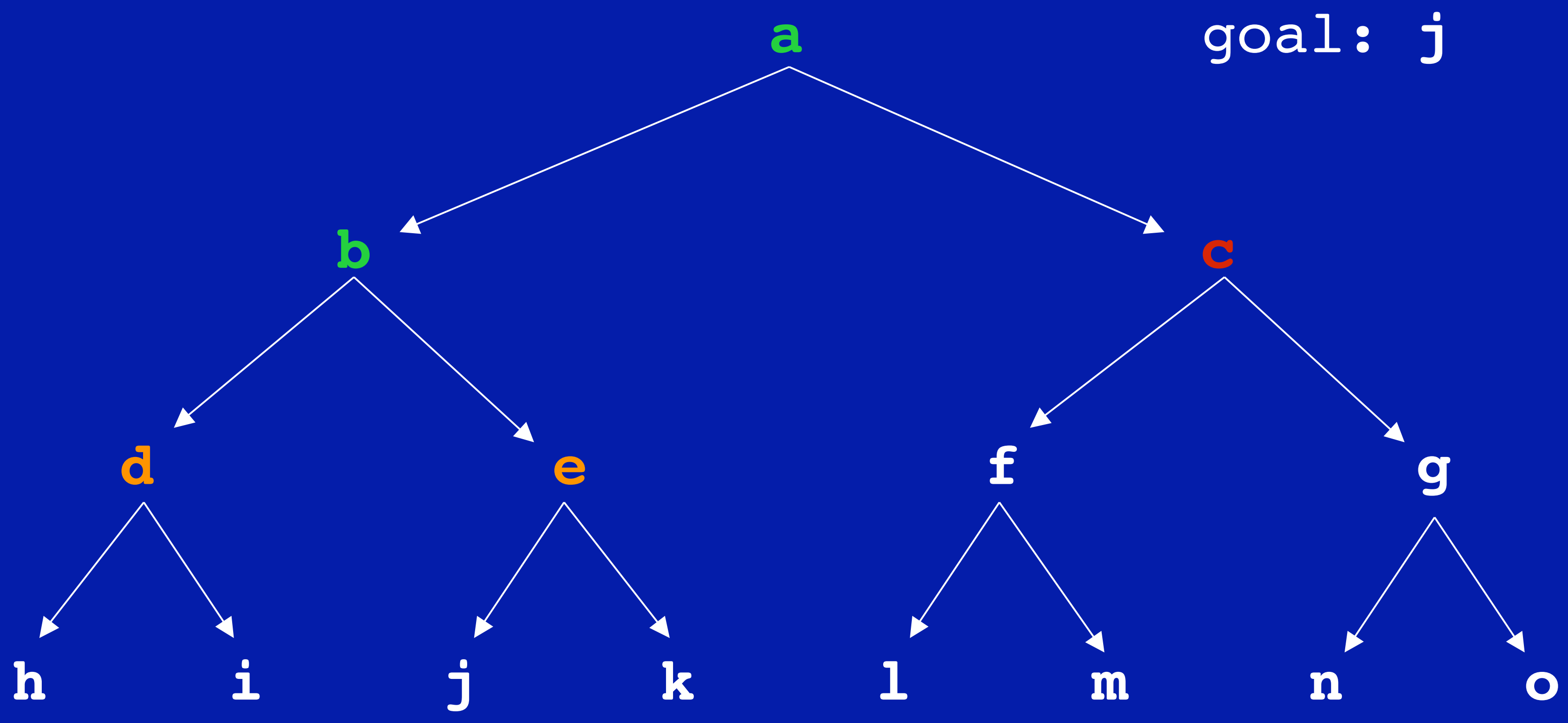
frontier:[[c,a]]
path being considered:[b,a]



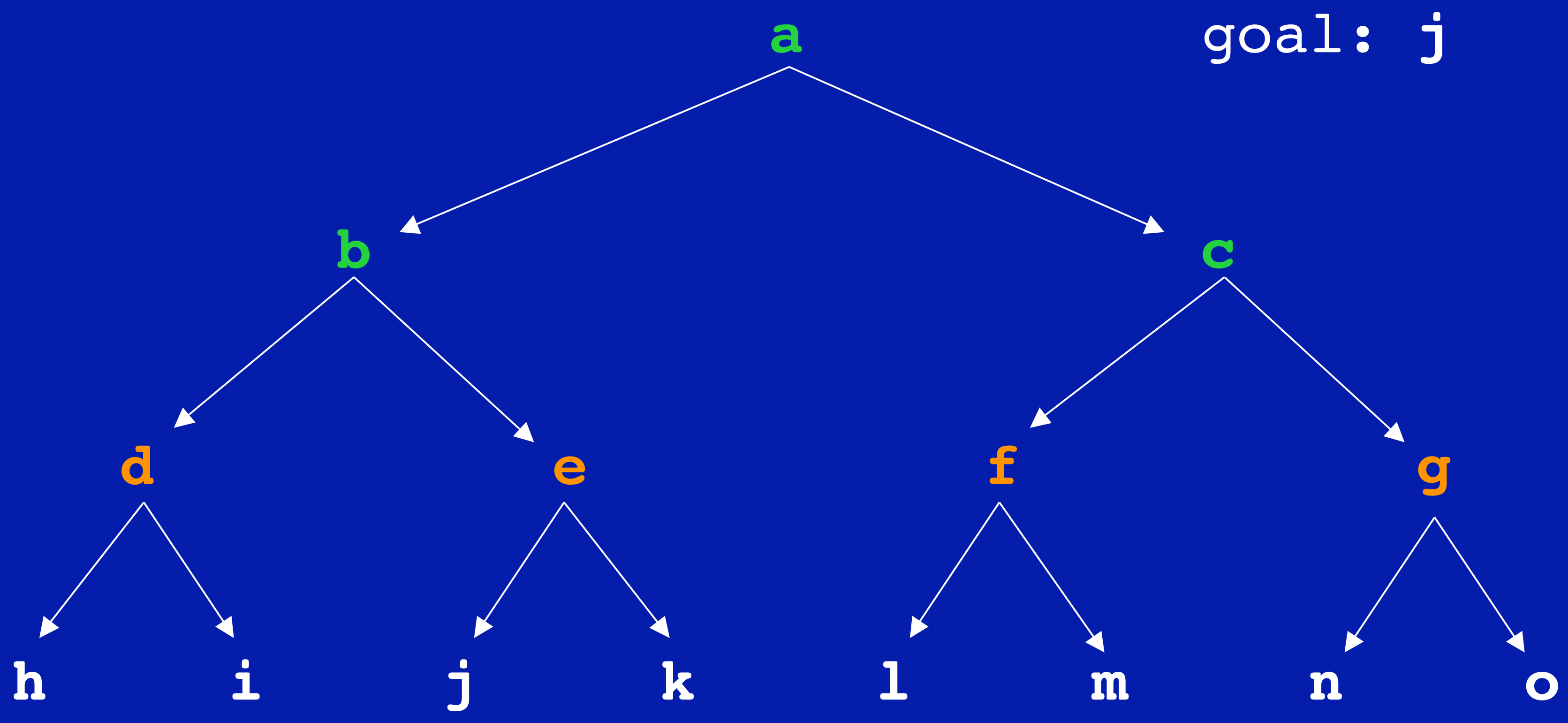
```
frontier:[[c,a],[d,b,a],[e,b,a]]
path being considered:
```



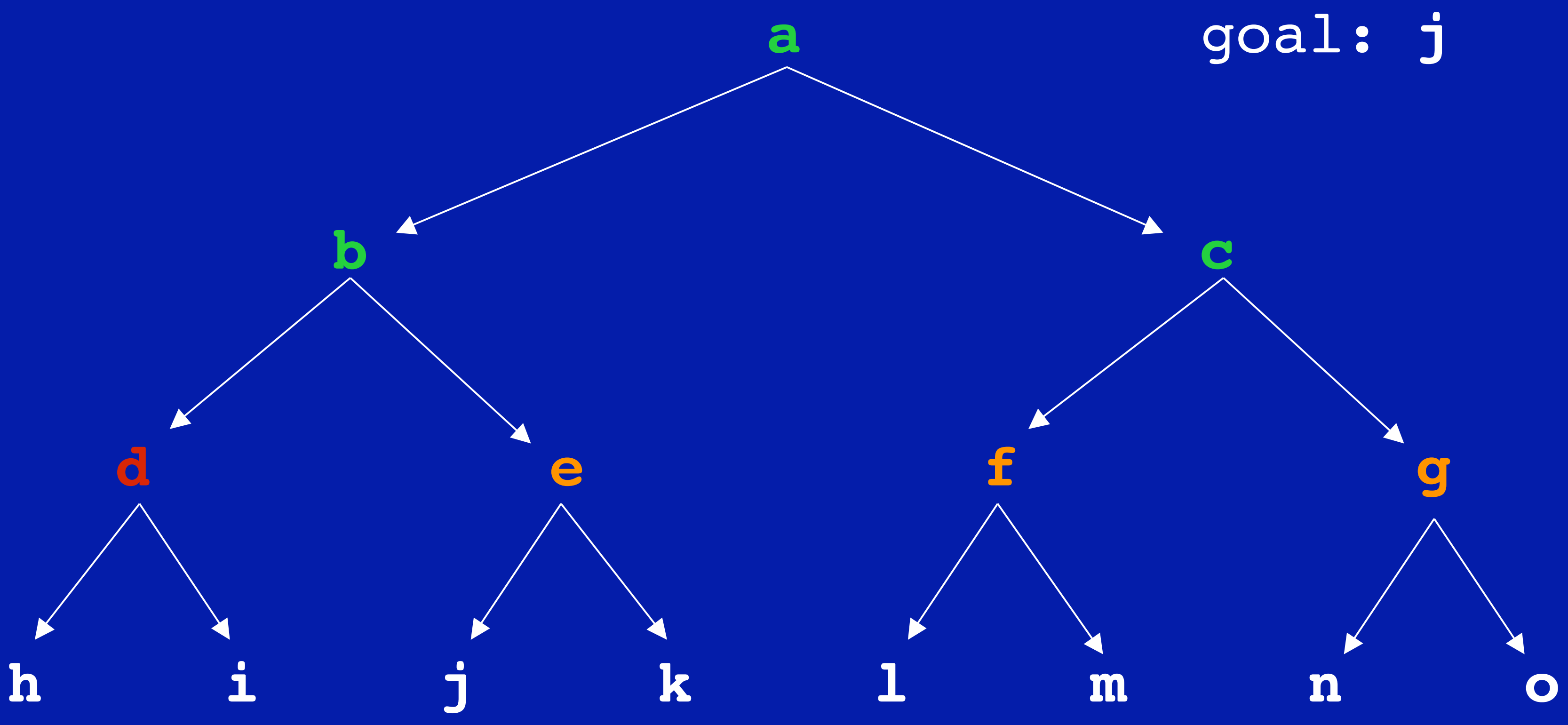
```
frontier:[[d,b,a],[e,b,a]]
path being considered:[c,a]
```



```
frontier:[[d,b,a],[e,b,a],[f,c,a],[g,c,a]]
path being considered:
```

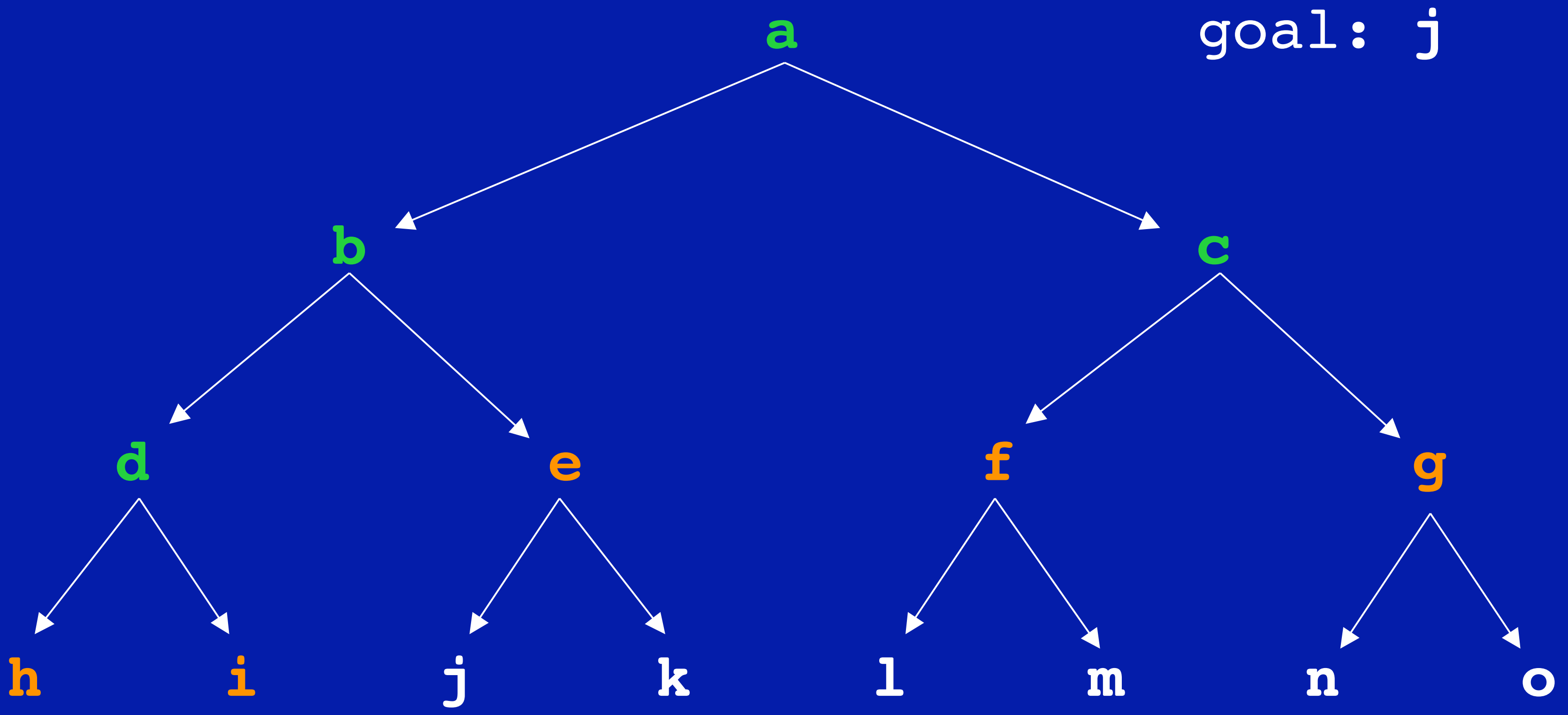


frontier:[[e,b,a],[f,c,a],[g,c,a]]
path being considered:[d,b,a]

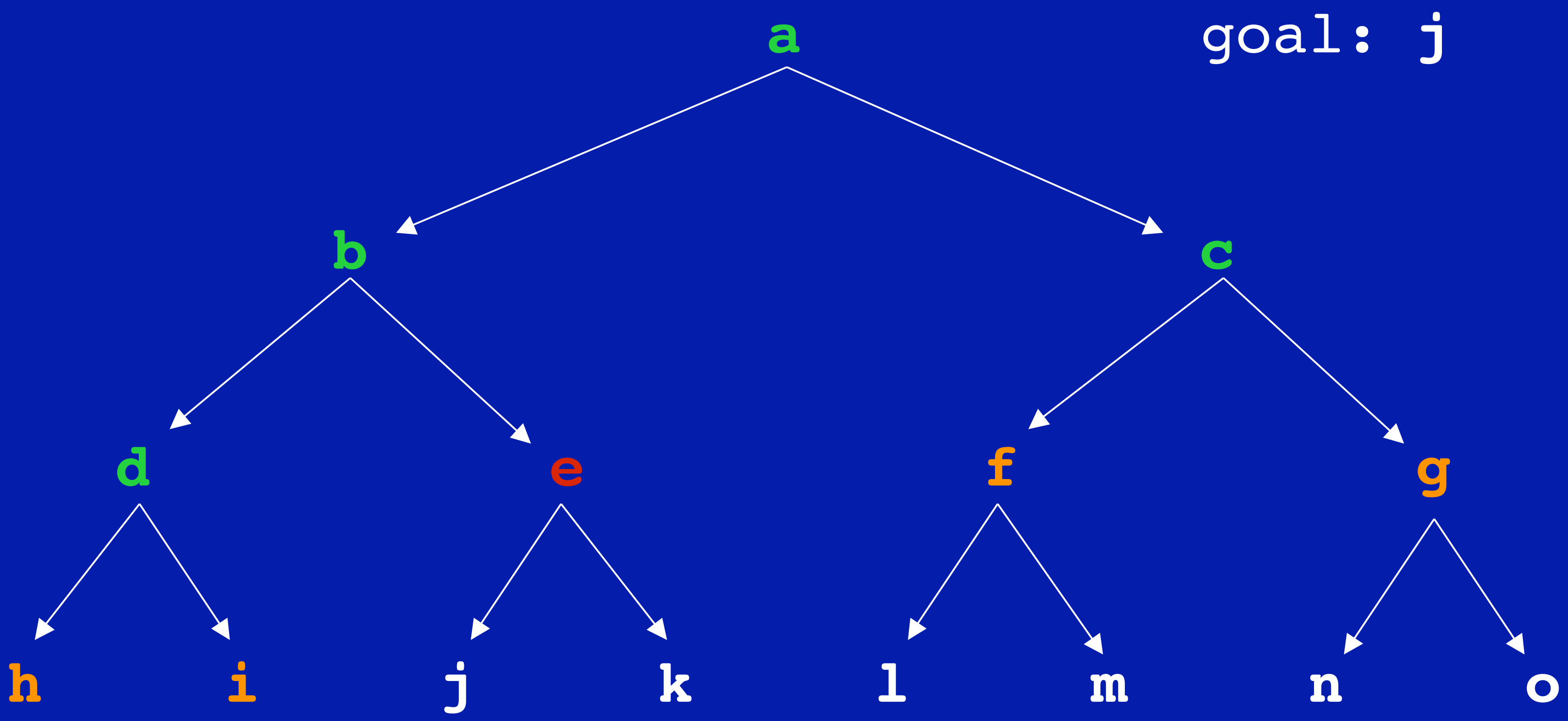


frontier:[[e,b,a],[f,c,a],[g,c,a],[h,d,b,a],[i,d,b,a]]
path being considered:

goal: j



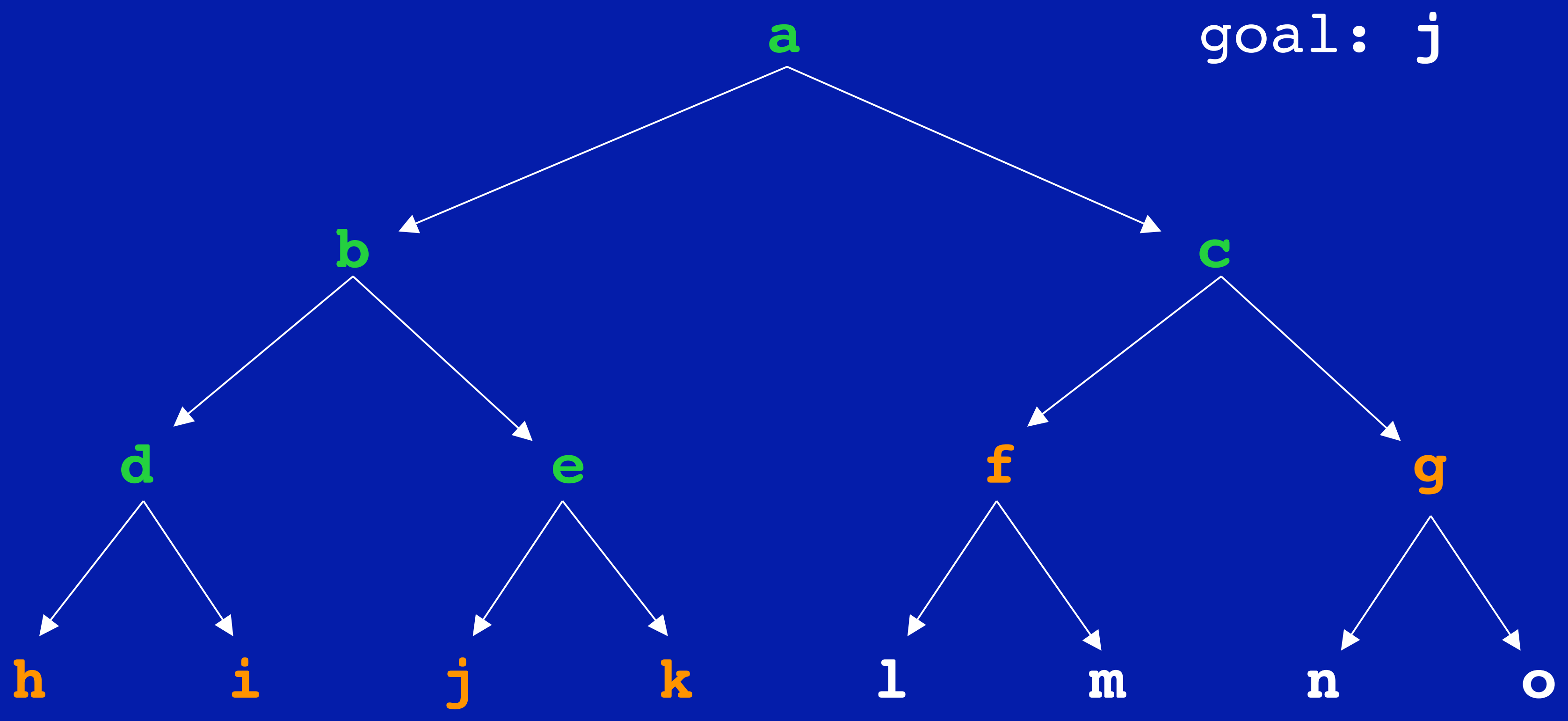
```
frontier:[[f,c,a],[g,c,a],[h,d,b,a],[i,d,b,a]]
path being considered:[e,b,a]
```



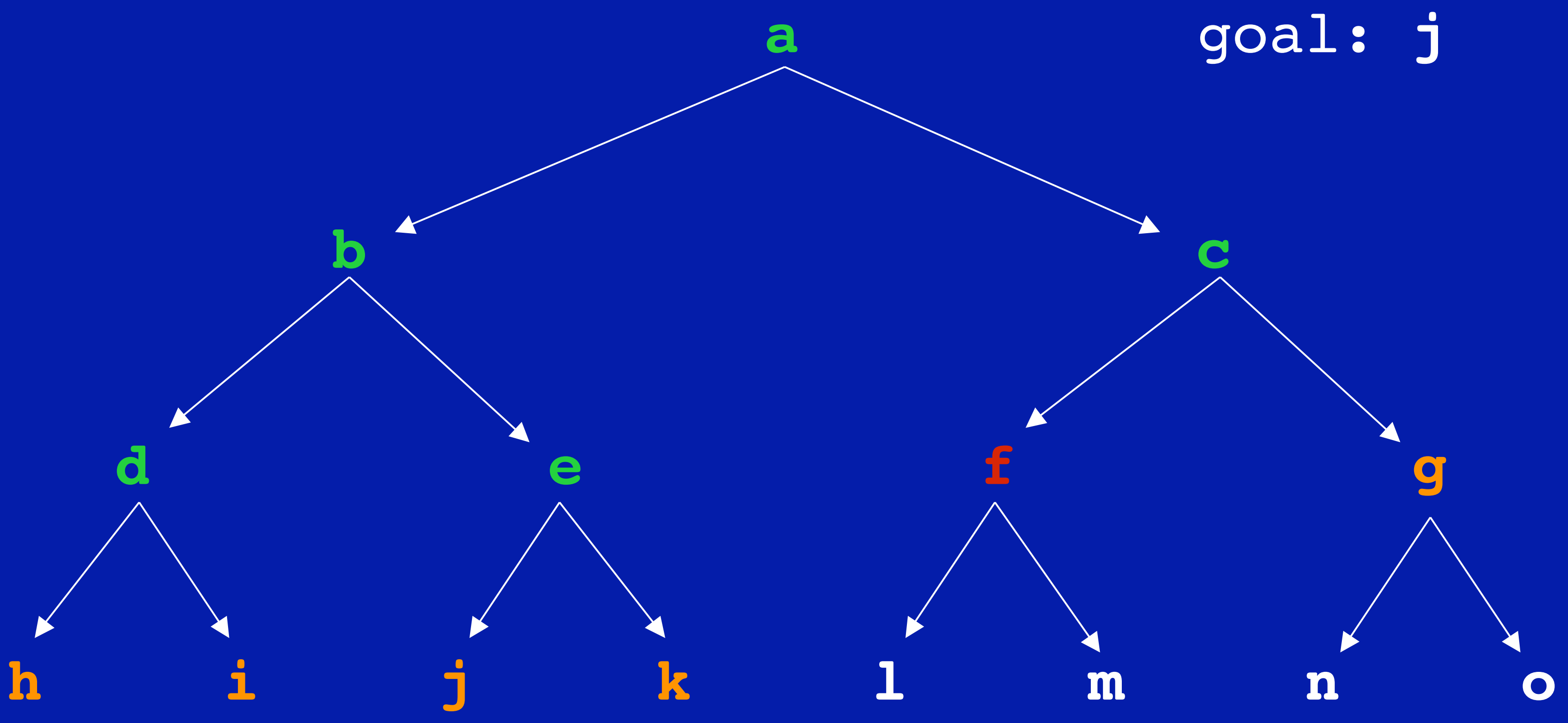
frontier:[[f,c,a],[g,c,a],[h,d,b,a],[i,d,b,a],[j,e,b,a],[k,e,b,a]]

path being considered:

goal: j

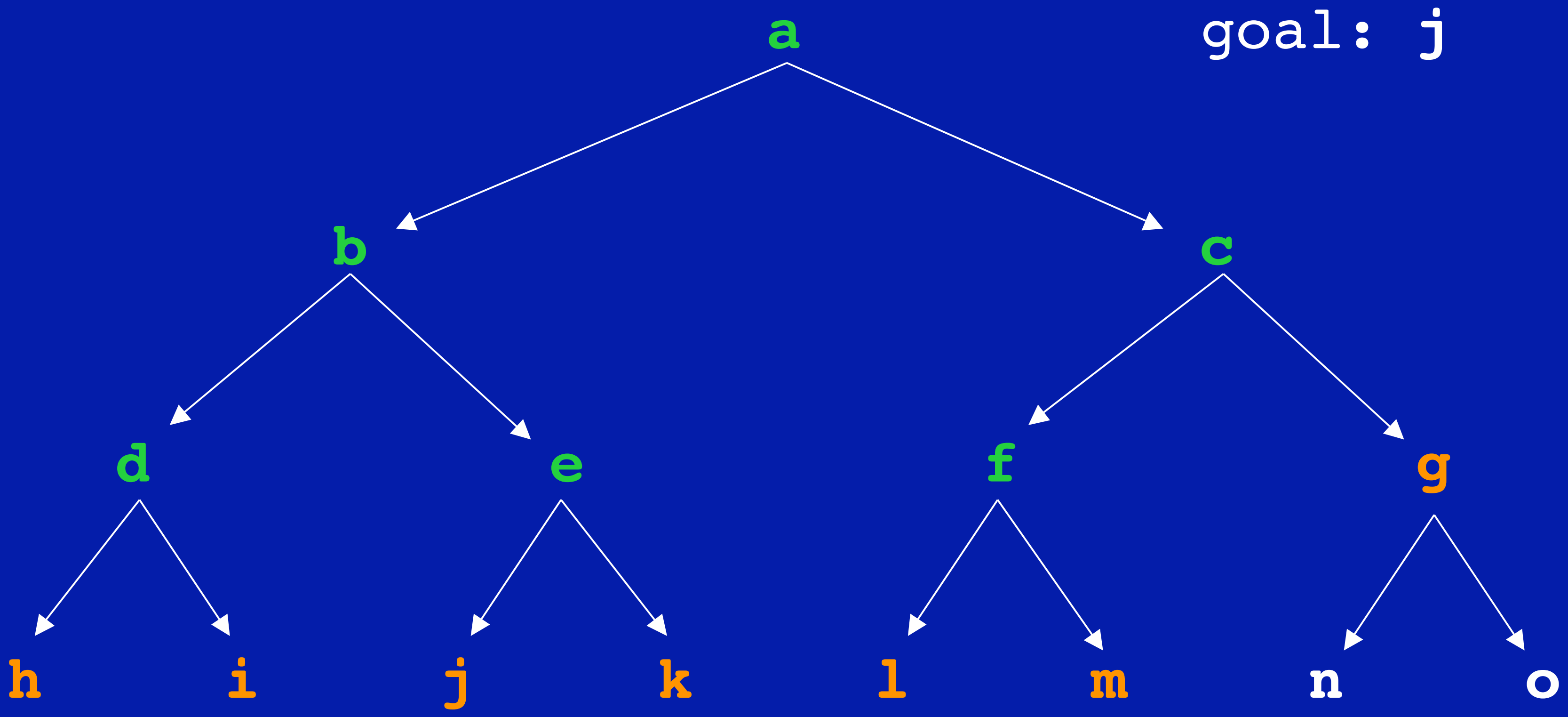


frontier:[[g,c,a],[h,d,b,a],[i,d,b,a],[j,e,b,a],[k,e,b,a]]
path being considered:[f,c,a]



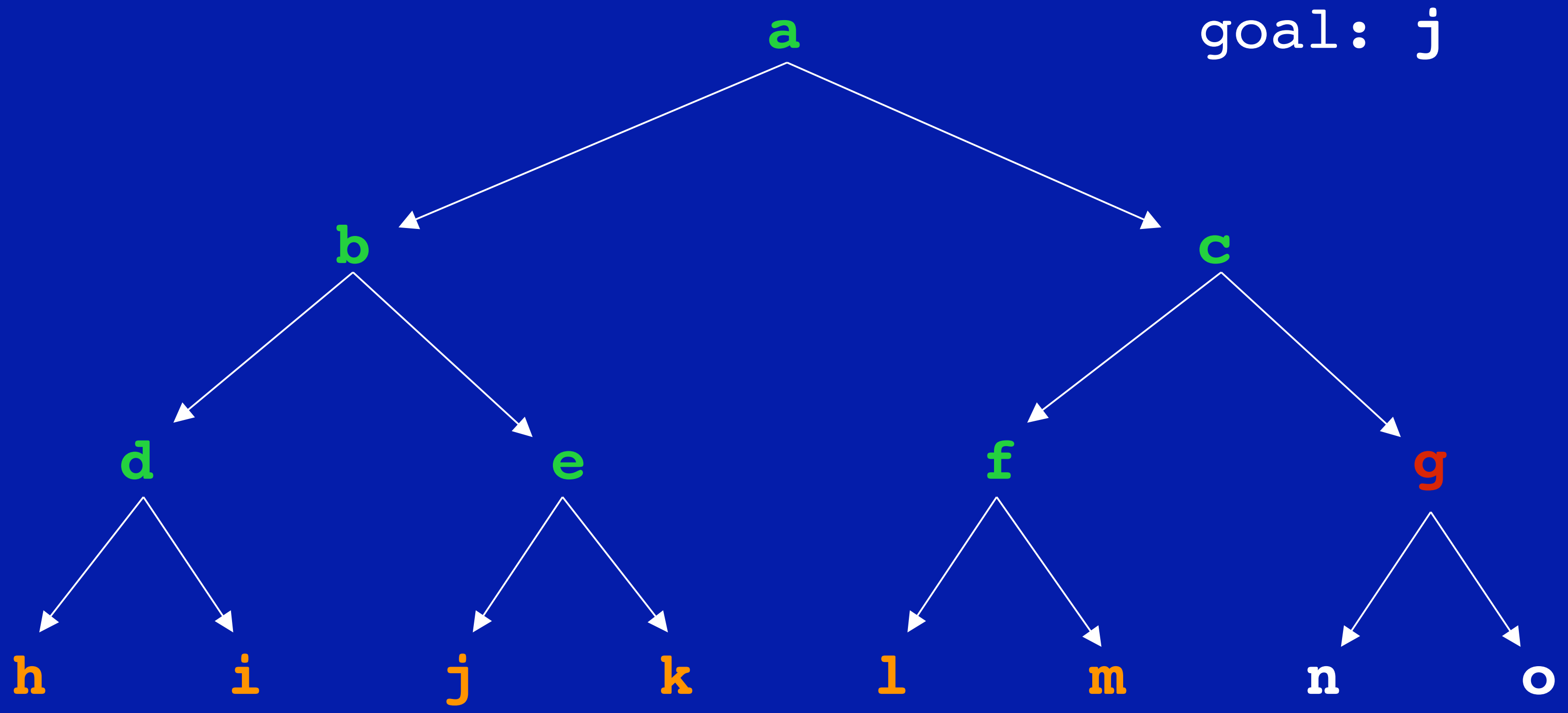
```
frontier:[[g,c,a],[h,d,b,a],[i,d,b,a],[j,e,b,a],[k,e,b
,a],[l,f,c,a],[m,f,c,a]]
path being considered:
```

goal: j



frontier:[[h,d,b,a],[i,d,b,a],[j,e,b,a],[k,e,b,a],[l,f,c,a],[m,f,c,a]]
path being considered:[g,c,a]

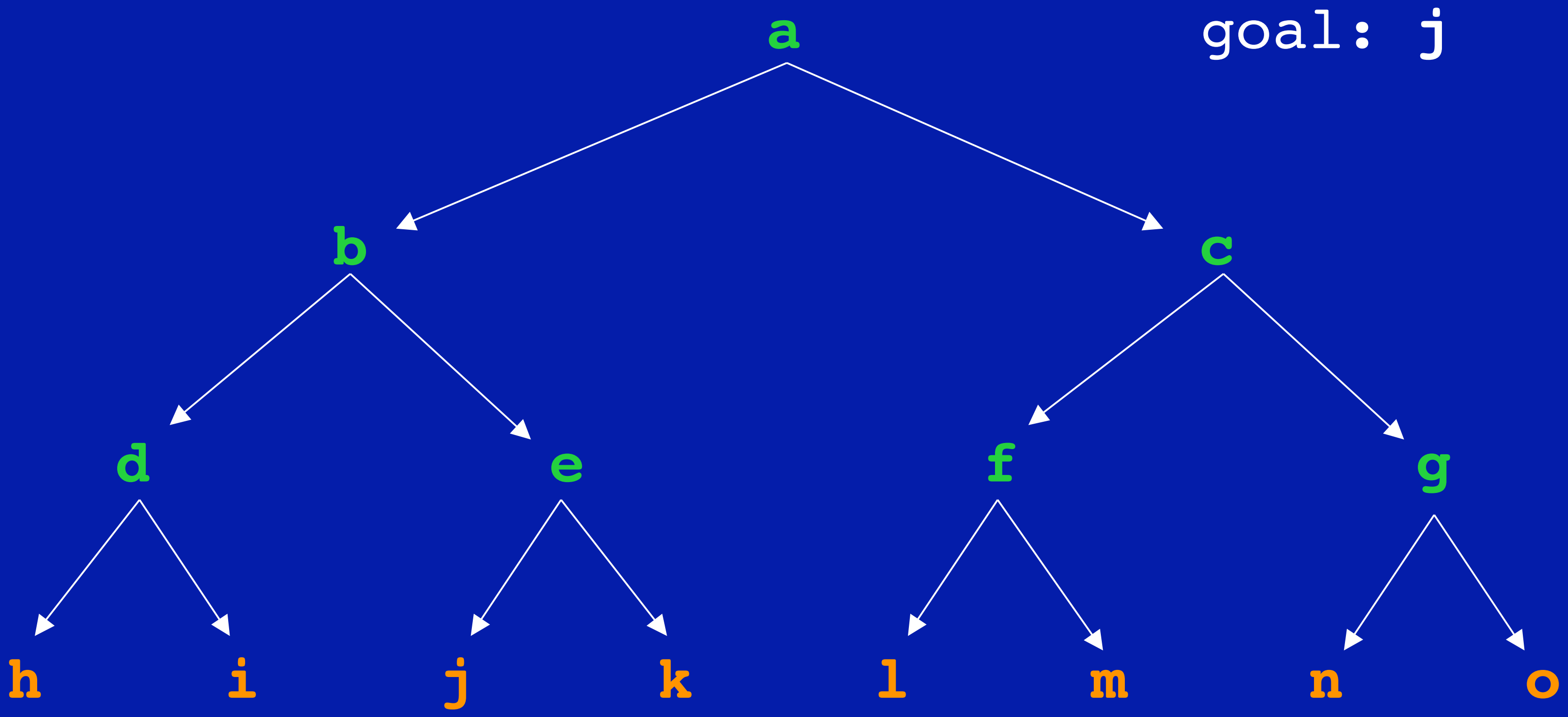
goal: j



frontier:[[h,d,b,a],[i,d,b,a],[j,e,b,a],[k,e,b,a],[l,f,c,a],[m,f,c,a],[n,g,c,a],[o,g,c,a]]

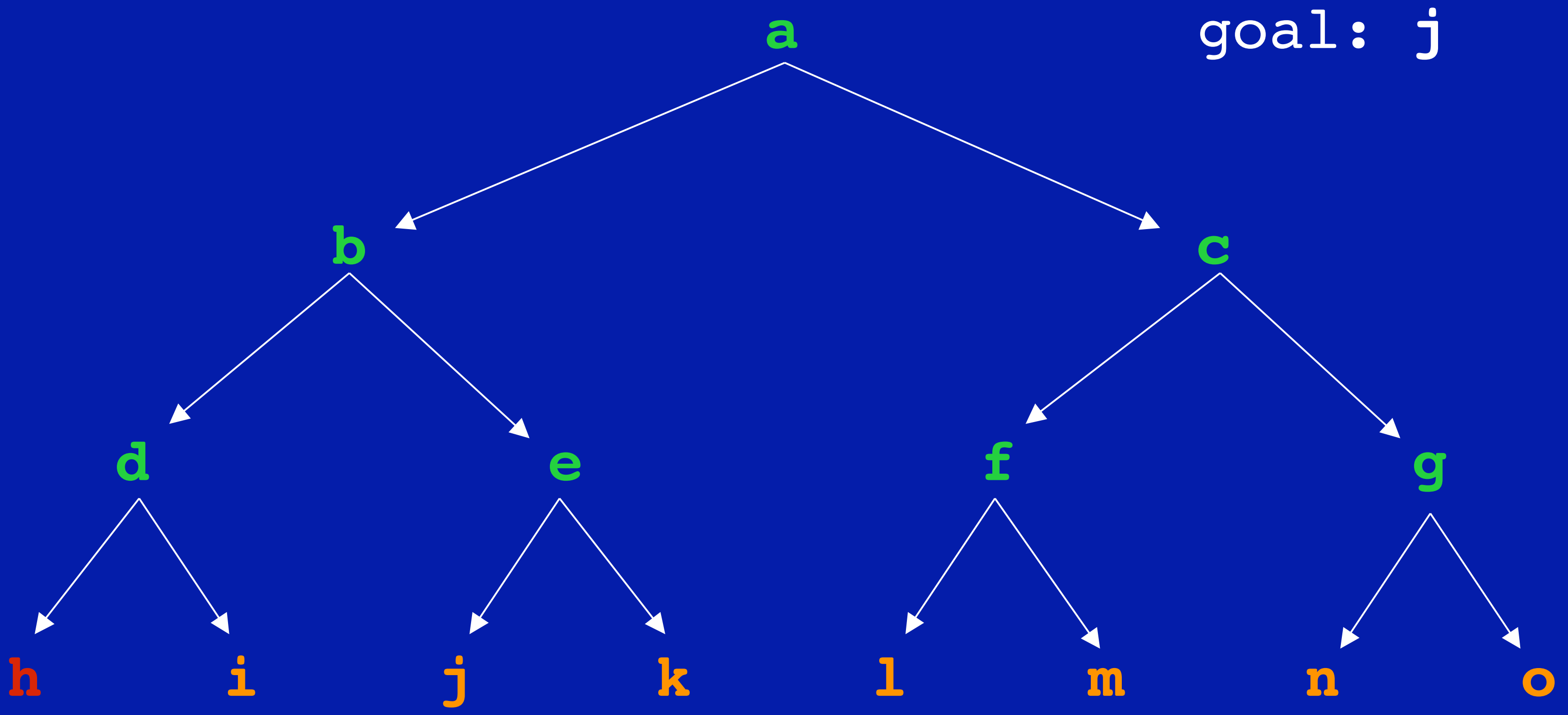
path being considered:

goal: j



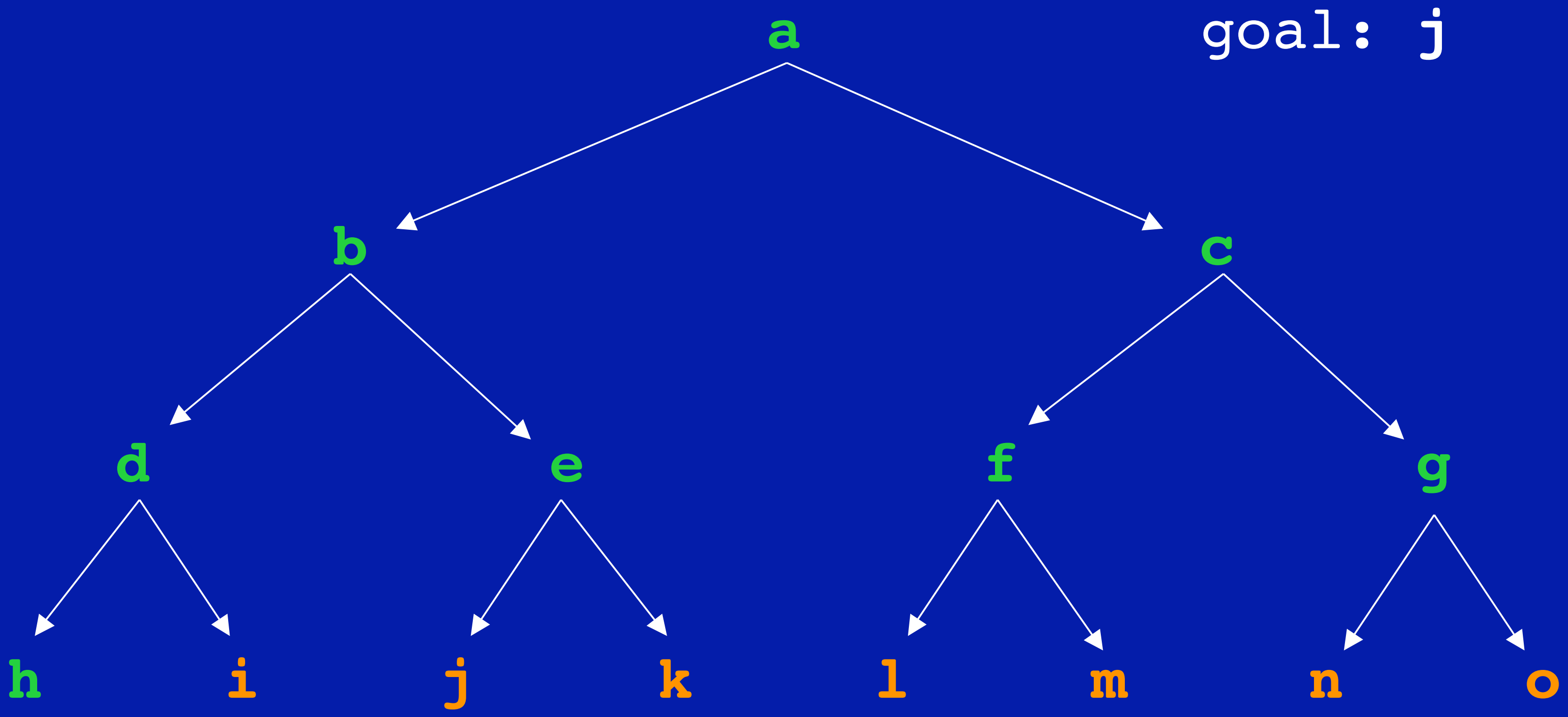
```
frontier:[[i,d,b,a],[j,e,b,a],[k,e,b,a],[l,f,c,a],[m,f
,c,a],[n,g,c,a],[o,g,c,a]]
path being considered:[h,d,b,a]
```

goal: j

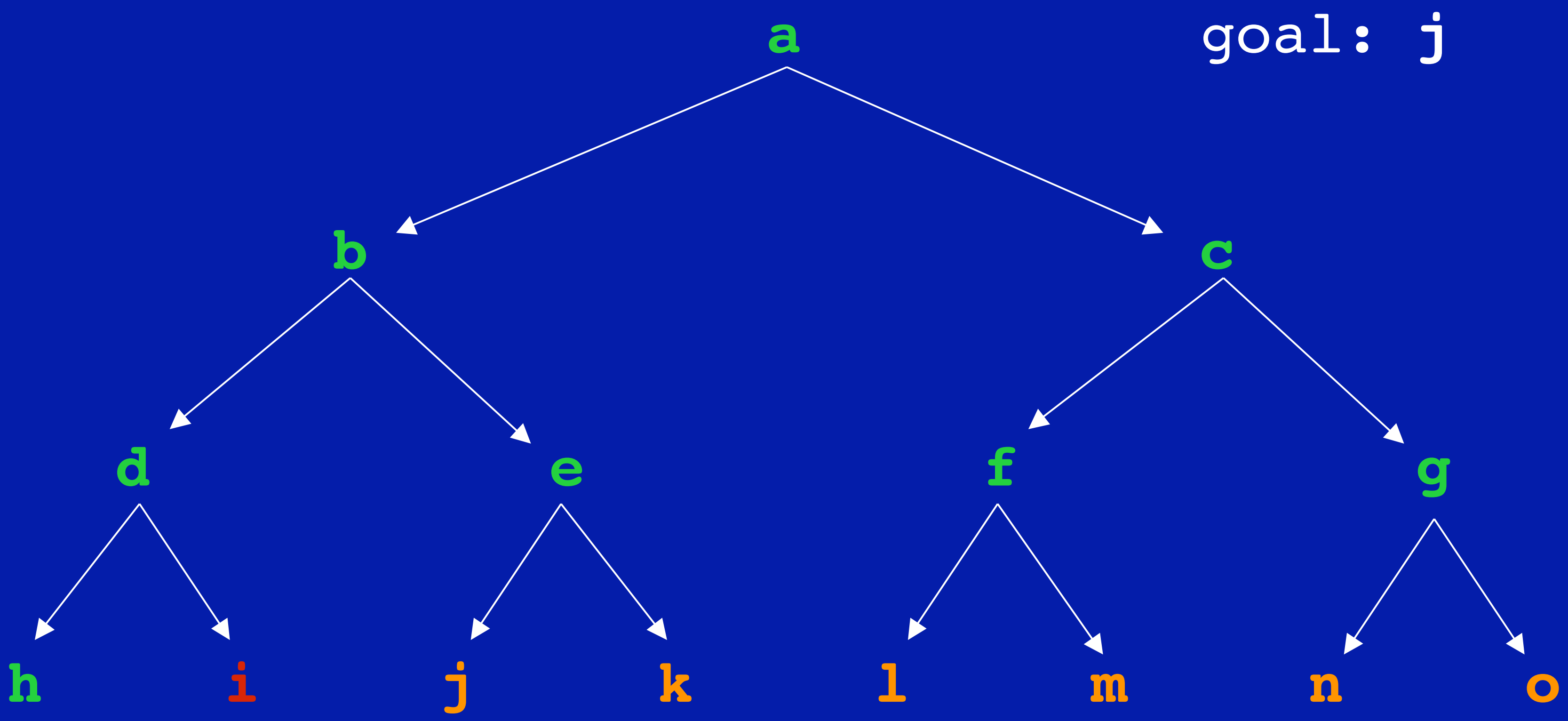


frontier:[[i,d,b,a],[j,e,b,a],[k,e,b,a],[l,f,c,a],[m,f,c,a],[n,g,c,a],[o,g,c,a]]
path being considered:

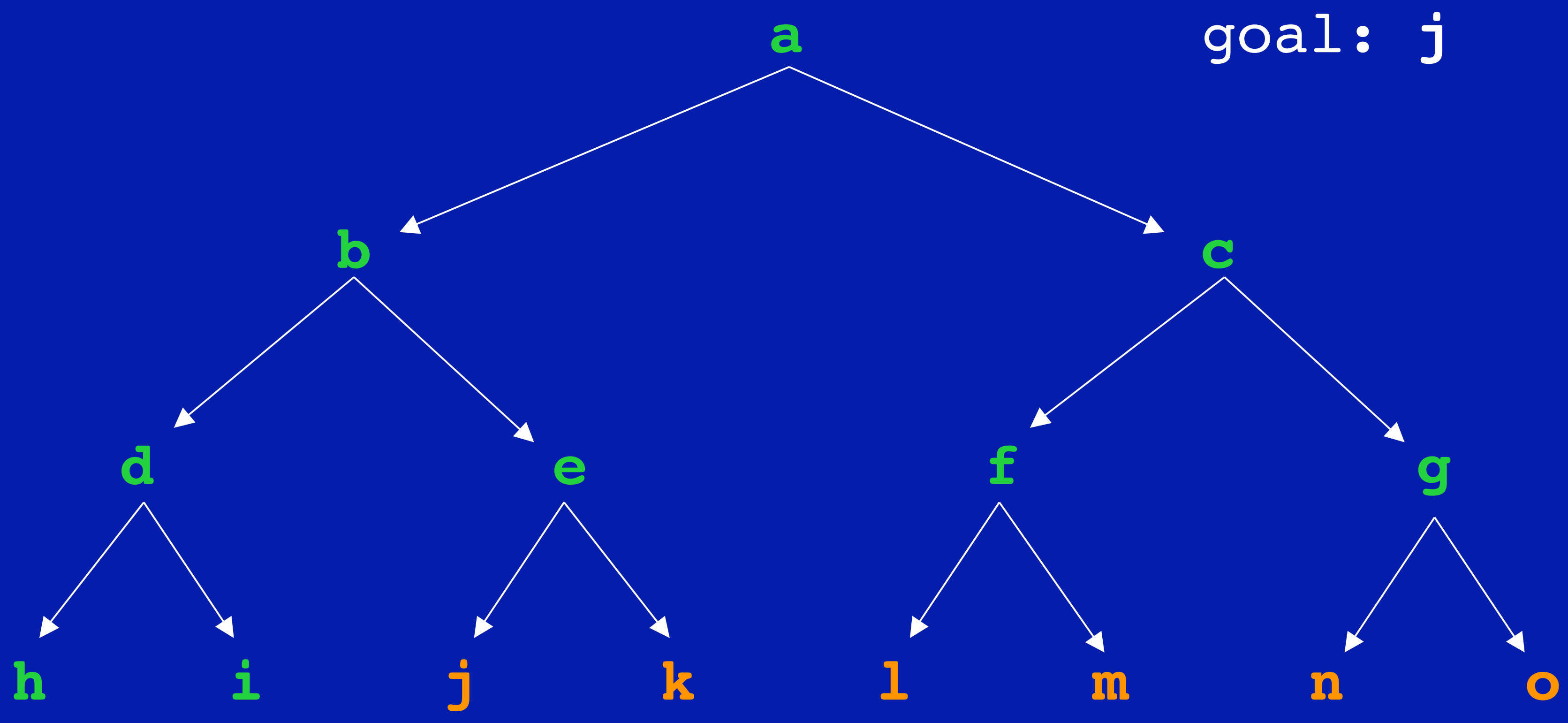
goal: j



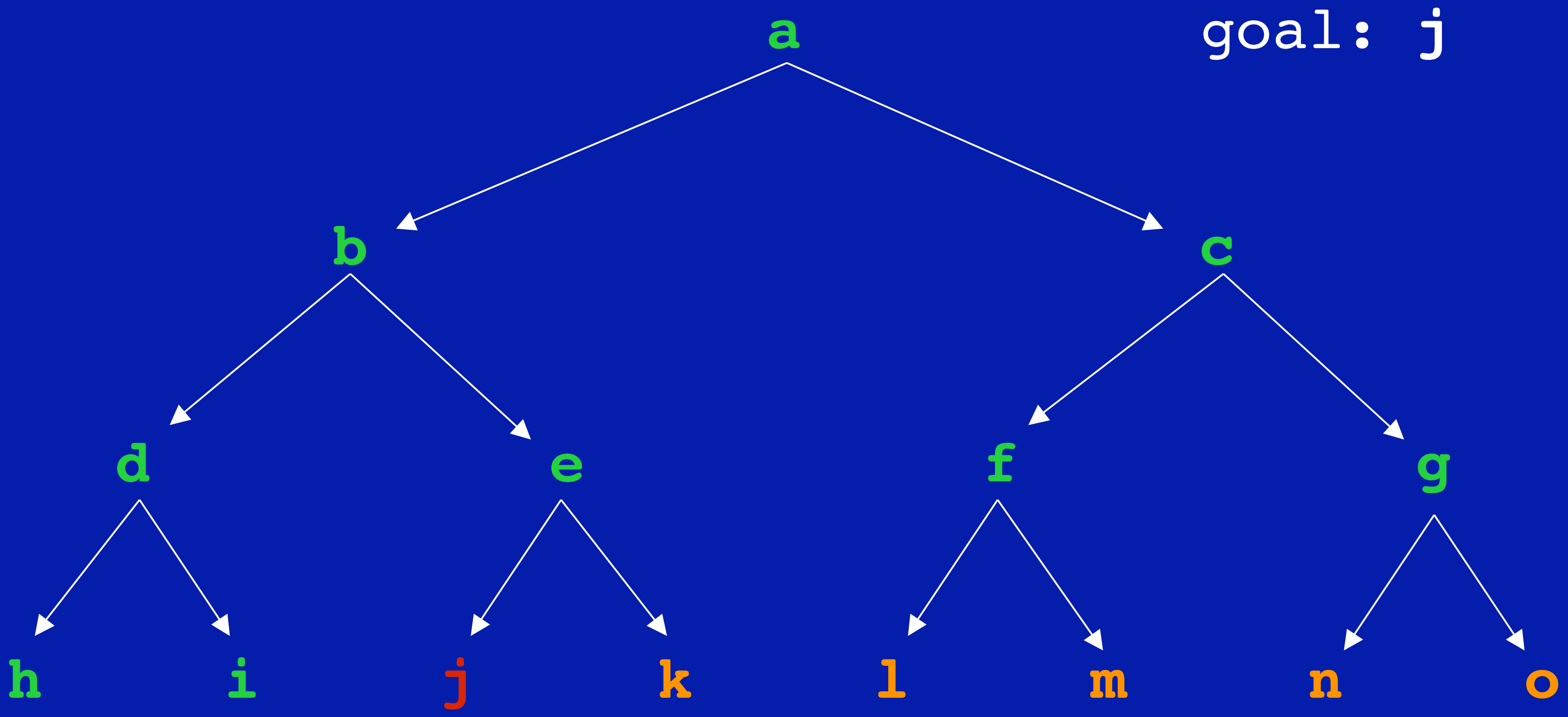
frontier:[[j,e,b,a],[k,e,b,a],[l,f,c,a],[m,f,c,a],[n,g,c,a],[o,g,c,a]]
path being considered:[i,d,b,a]



```
frontier:[[j,e,b,a],[k,e,b,a],[l,f,c,a],[m,f,c,a],[n,g
,c,a],[o,g,c,a]]
path being considered:
```



```
frontier:[[k,e,b,a],[l,f,c,a],[m,f,c,a],[n,g,c,a],[o,g
,c,a]]
path being considered:[j,e,b,a]
```



Path search algorithm

```
psearch(F,[N|P]) <- choose([N|P],F,_) &
                    is_goal(N).

psearch(F,S) <- choose([N|P],F,F1) &
                neighbors(N,NN) &
                add_paths(NN,[N|P],NN2) &
                add_to_frontier(NN2,F1,F2) &
                psearch(F2,S).
```

Path search algorithm

```
psearch(F,[N|P]) <- choose([N|P],F,_) &
                    is_goal(N).

psearch(F,S) <- choose([N|P],F,F1) &
                neighbors(N,NN) &
                add_paths(NN,[N|P],NN2) &
                add_to_frontier(NN2,F1,F2) &
                psearch(F2,S).
```

psearch(F,S) is true if the search from the end of a path on the frontier F results in a path S to the goal. Remember that elements of the frontier F are paths in reverse order, not individual nodes.

Path search algorithm

```
psearch(F,[N|P]) <- choose([N|P],F,_) &
                    is_goal(N).

psearch(F,S) <- choose([N|P],F,F1) &
                neighbors(N,NN) &
                add_paths(NN,[N|P],NN2) &
                add_to_frontier(NN2,F1,F2) &
                psearch(F2,S).
```

is_goal(N) is true if N is a goal node

Path search algorithm

```
psearch(F,[N|P]) <- choose([N|P],F,_) &
                    is_goal(N).

psearch(F,S) <- choose([N|P],F,F1) &
                neighbors(N,NN) &
                add_paths(NN,[N|P],NN2) &
                add_to_frontier(NN2,F1,F2) &
                psearch(F2,S).
```

choose([N|P],F,F1) means [N|P] is some path chosen from F - the frontier - and F1 is the set of paths remaining when [N|P] is removed. This fails if F is empty.

Path search algorithm

```
psearch(F,[N|P]) <- choose([N|P],F,_) &
                    is_goal(N).

psearch(F,S) <- choose([N|P],F,F1) &
                neighbors(N,NN) &
                add_paths(NN,[N|P],NN2) &
                add_to_frontier(NN2,F1,F2) &
                psearch(F2,S).
```

neighbors(N,NN) is true if NN is the list of neighbors of node N

Path search algorithm

```
psearch(F,[N|P]) <- choose([N|P],F,_) &
                    is_goal(N).

psearch(F,S) <- choose([N|P],F,F1) &
                neighbors(N,NN) &
                add_paths(NN,[N|P],NN2) &
                add_to_frontier(NN2,F1,F2) &
                psearch(F2,S).
```

add_paths(NN,[N|P],NN2) means that NN2 is the list of paths obtained by adding one element of NN to the front of the path [N|P]. (If NN has m elements, and the length of [N|P] is q, then NN2 has n paths of length q+1.)

Path search algorithm

```
psearch(F,[N|P]) <- choose([N|P],F,_) &
                    is_goal(N).

psearch(F,S) <- choose([N|P],F,F1) &
                neighbors(N,NN) &
                add_paths(NN,[N|P],NN2) &
                add_to_frontier(NN2,F1,F2) &
                psearch(F2,S).
```

```
cilog: ask add_paths([a,b,c],[x,y,z],X).
Answer: add_paths([a, b, c], [x, y, z], [[a, x, y, z],
[b, x, y, z], [c, x, y, z]]).
```

Path search algorithm

```
psearch(F,[N|P]) <- choose([N|P],F,_) &
                    is_goal(N).

psearch(F,S) <- choose([N|P],F,F1) &
                neighbors(N,NN) &
                add_paths(NN,[N|P],NN2) &
                add_to_frontier(NN2,F1,F2) &
                psearch(F2,S).
```

add_to_frontier(NN2,F1,F2) means that F2 is the new frontier made by adding the list of paths NN2 to the old frontier F1

CILOG path search

```
neighbors(a,[b,c]).
neighbors(b,[d,e]).
neighbors(c,[f,g]).
neighbors(d,[h,i]).
neighbors(e,[j,k]).
neighbors(f,[l,m]).
neighbors(g,[n,o]).
neighbors(h,[]).
neighbors(i,[]).
neighbors(j,[]).
neighbors(k,[]).
neighbors(l,[]).
neighbors(m,[]).
neighbors(n,[]).
neighbors(o,[]).


is_goal(j).

append([],Z,Z).
append([A|X],Y,[A|Z]) <- append(X,Y,Z).
```

CILOG path search

```
/* this is David Poole's code...I just stole it. */
/* to make it go, don't ask psearch(a,X).  You need to say
   ask psearch([[a]],X).

psearch(F,[N|P]) <- choose([N|P],F,_) &
                    is_goal(N).

psearch(F,S) <- choose([N|P],F,F1) &
                neighbors(N,NN) &
                add_paths(NN,[N|P],NN2) &
                add_to_frontier(NN2,F1,F2) &
                psearch(F2,S).

add_paths([],_,[]).
add_paths([M|R],P,[[M|P]|PR]) <- add_paths(R,P,PR).

choose(N,[N|Flist],Flist).


/* this is breadth-first search */
add_to_frontier(Nodelist,Flist1,Flist2) <- append(Flist1,Nodelist,Flist2).

/* this is depth-first search
add_to_frontier(Nodelist,Flist1,Flist2) <- append(Nodelist,Flist1,Flist2).
*/
```

Search Recap

We know basic principles of search

We know how to search an existing graph and how to generate the graph as we need it

We know how to use heuristic knowledge to help us choose what to do next

Search Recap

We know basic principles of search

We know how to search an existing graph and how to generate the graph as we need it

We know how to use heuristic knowledge to help us choose what to do next

This helps us find a goal in a non-hostile world.

But can we use this stuff to find a goal in the more realistic scenario where other agents are trying to prevent us from reaching our goal?

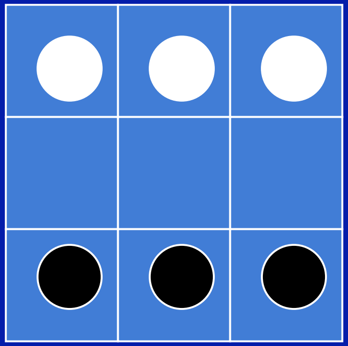
The Joy of Hex

The game of hexapawn

The Joy of Hex

The game of hexapawn

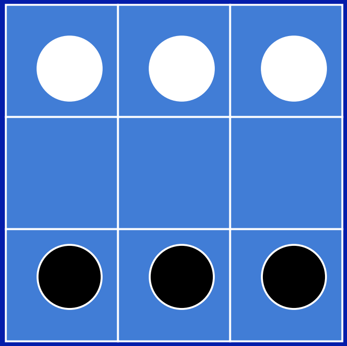
- 3 x 3 board
- 3 pawns on each side



The Joy of Hex

The game of hexapawn

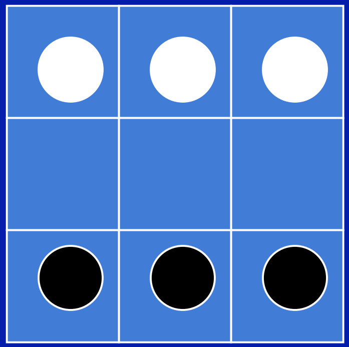
- 3 x 3 board
- 3 pawns on each side
- movement of pawns:



The Joy of Hex

The game of hexapawn

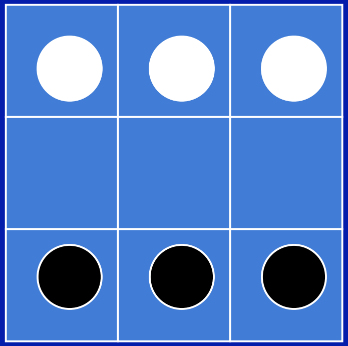
- 3 x 3 board
- 3 pawns on each side
- movement of pawns:
    - white moves first



The Joy of Hex

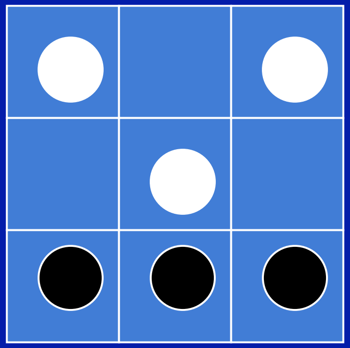
The game of hexapawn

- 3 x 3 board
- 3 pawns on each side
- movement of pawns:
    - white moves first
    - pawn can move straight ahead one space if that space is empty



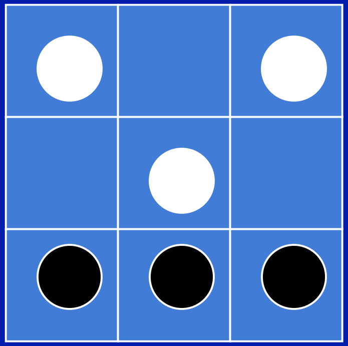
The Joy of Hex

The game of hexapawn



- 3 x 3 board
- 3 pawns on each side
- movement of pawns:
    - white moves first
    - pawn can move straight ahead one space if that space is empty

The Joy of Hex

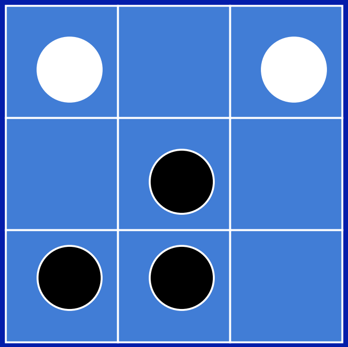


The game of hexapawn

- 3 x 3 board
- 3 pawns on each side
- movement of pawns:
  - white moves first
  - pawn can move straight ahead one space if that space is empty
  - pawn can move diagonally one space forward to capture opponent’s pawn occupying that space

The Joy of Hex

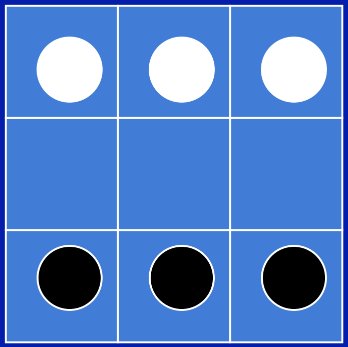
The game of hexapawn



- 3 x 3 board
- 3 pawns on each side
- movement of pawns:
  - white moves first
  - pawn can move straight ahead one space if that space is empty
  - pawn can move diagonally one space forward to capture opponent’s pawn occupying that space

The Joy of Hex

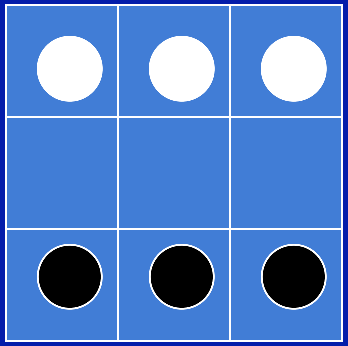
The game of hexapawn



- 3 ways to win:

The Joy of Hex

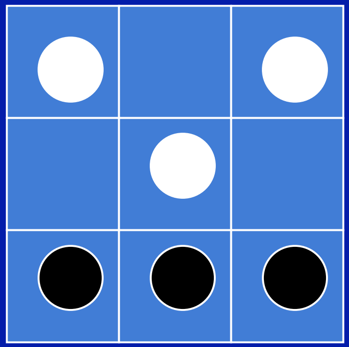
The game of hexapawn



- 3 ways to win:
    - capture all your opponent’s pawns

The Joy of Hex

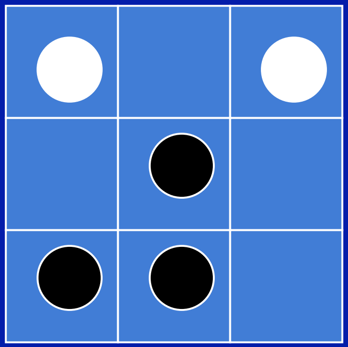
The game of hexapawn



- 3 ways to win:
    - capture all your opponent’s pawns

The Joy of Hex

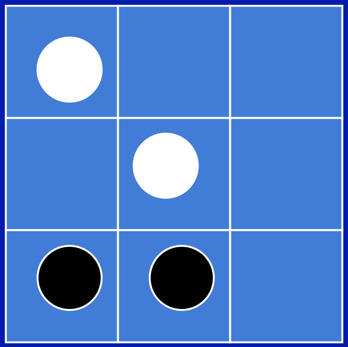
The game of hexapawn



- 3 ways to win:
    - capture all your opponent’s pawns

The Joy of Hex

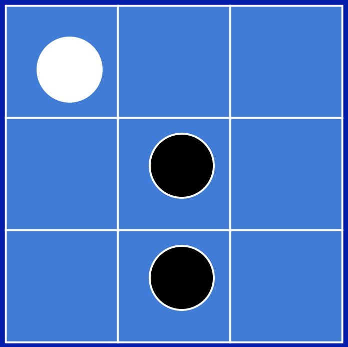
The game of hexapawn



- 3 ways to win:
    - capture all your opponent’s pawns

The Joy of Hex

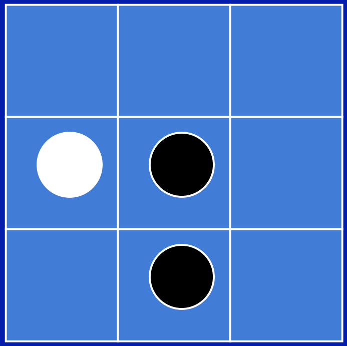
The game of hexapawn



- 3 ways to win:
    - capture all your opponent’s pawns

The Joy of Hex

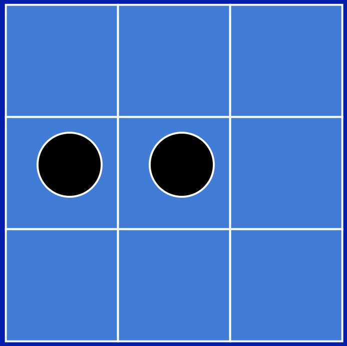
The game of hexapawn



- 3 ways to win:
    - capture all your opponent’s pawns

The Joy of Hex

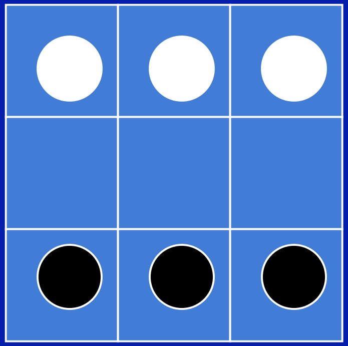
The game of hexapawn



- 3 ways to win:
  - capture all your opponent’s pawns

The Joy of Hex

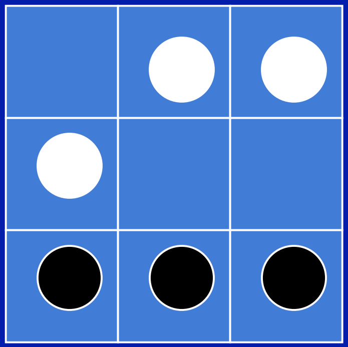
The game of hexapawn



- 3 ways to win:
    - capture all your opponent’s pawns
    - one of your pawns reaches the opposite end of the board

The Joy of Hex

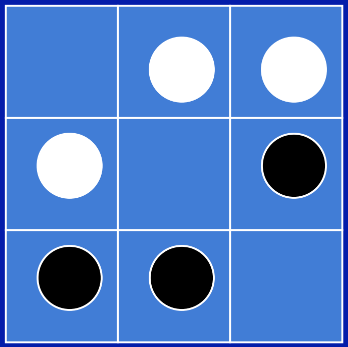
The game of hexapawn



- 3 ways to win:
  - capture all your opponent’s pawns
  - one of your pawns reaches the opposite end of the board

The Joy of Hex

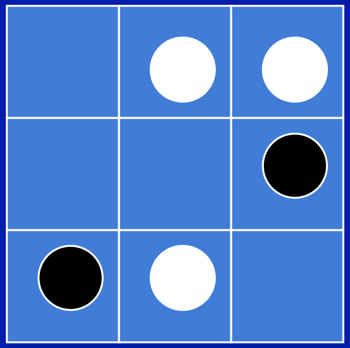
The game of hexapawn



- 3 ways to win:
    - capture all your opponent’s pawns
    - one of your pawns reaches the opposite end of the board

The Joy of Hex

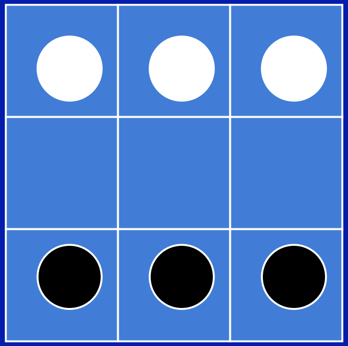
The game of hexapawn



- 3 ways to win:
  - capture all your opponent’s pawns
  - one of your pawns reaches the opposite end of the board

The Joy of Hex

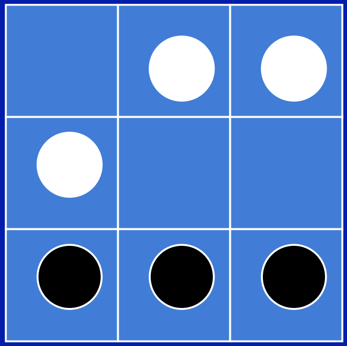
The game of hexapawn



- 3 ways to win:
    - capture all your opponent's pawns
    - one of your pawns reaches the opposite end of the board
    - it's your opponent's turn but your opponent can't move

The Joy of Hex

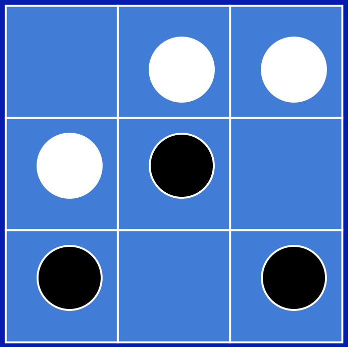
The game of hexapawn



- 3 ways to win:
    - capture all your opponent’s pawns
    - one of your pawns reaches the opposite end of the board
    - it’s your opponent’s turn but your opponent can’t move

The Joy of Hex

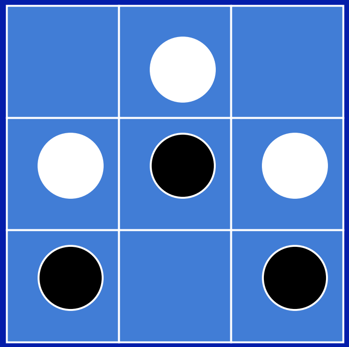
The game of hexapawn



- 3 ways to win:
  - capture all your opponent’s pawns
  - one of your pawns reaches the opposite end of the board
  - it’s your opponent’s turn but your opponent can’t move

The Joy of Hex

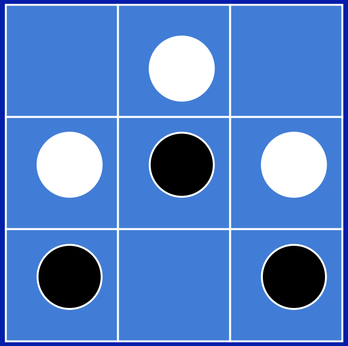
The game of hexapawn



- 3 ways to win:
  - capture all your opponent’s pawns
  - one of your pawns reaches the opposite end of the board
  - it’s your opponent’s turn but your opponent can’t move

The Joy of Hex

The game of hexapawn



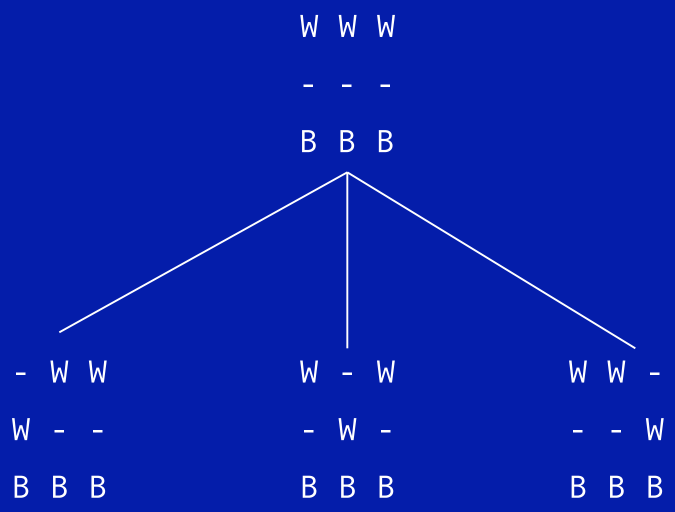
- 3 ways to win:
  - capture all your opponent’s pawns
  - one of your pawns reaches the opposite end of the board
  - it’s your opponent’s turn but your opponent can’t move

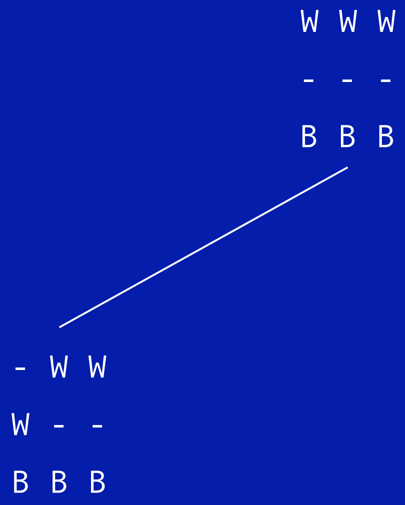
Now it’s time to look at a game in Kurt-o-vision...
(we’re pushing the black pawns)

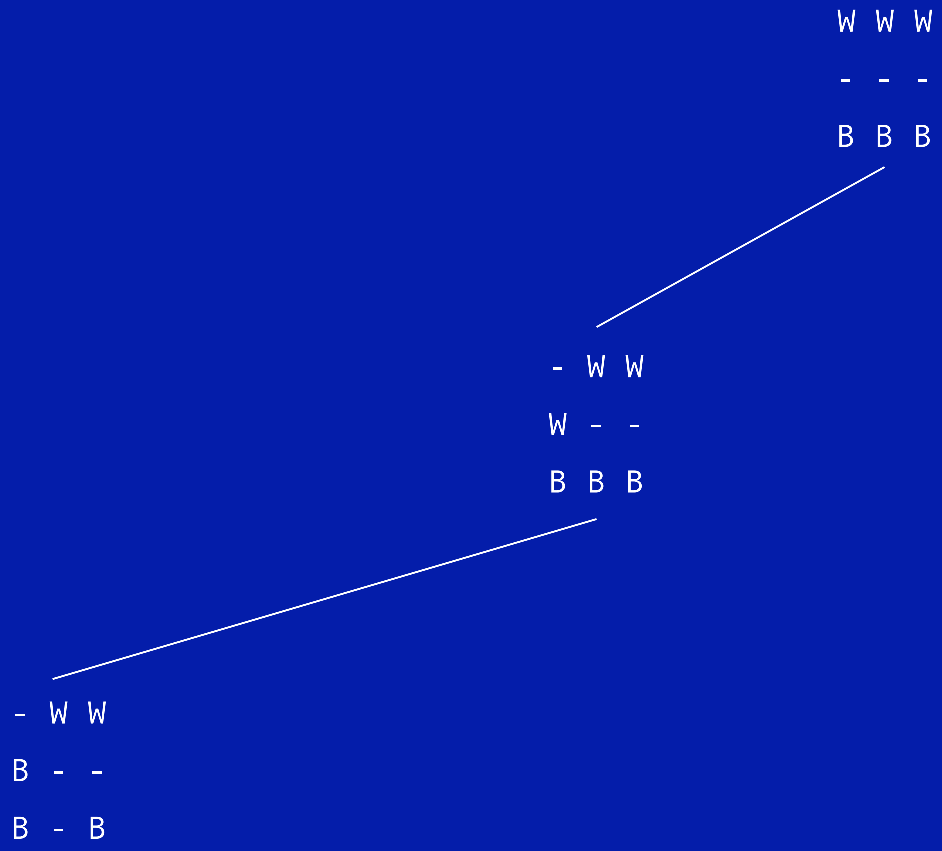
W W W

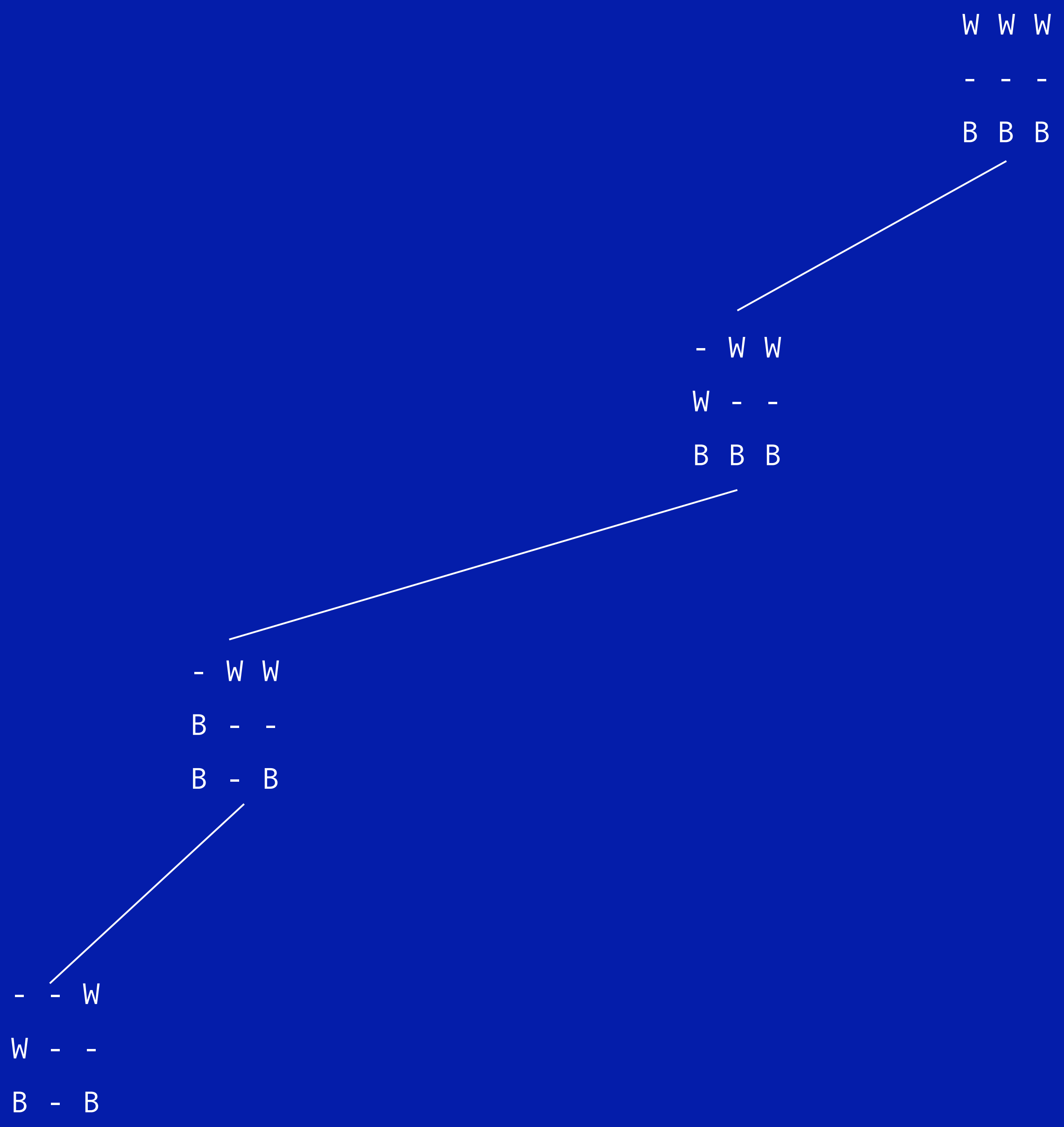
- - -

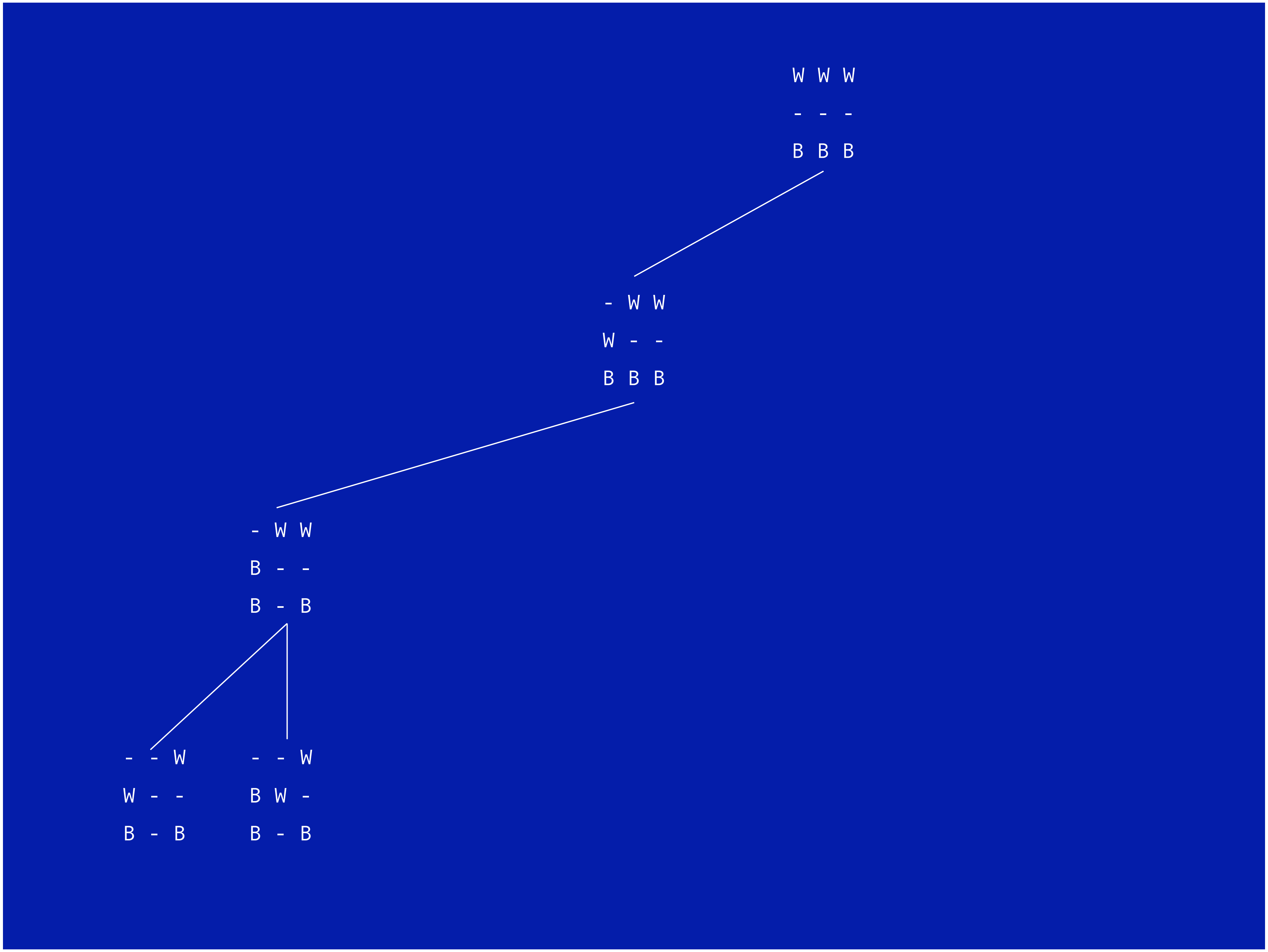
B B B

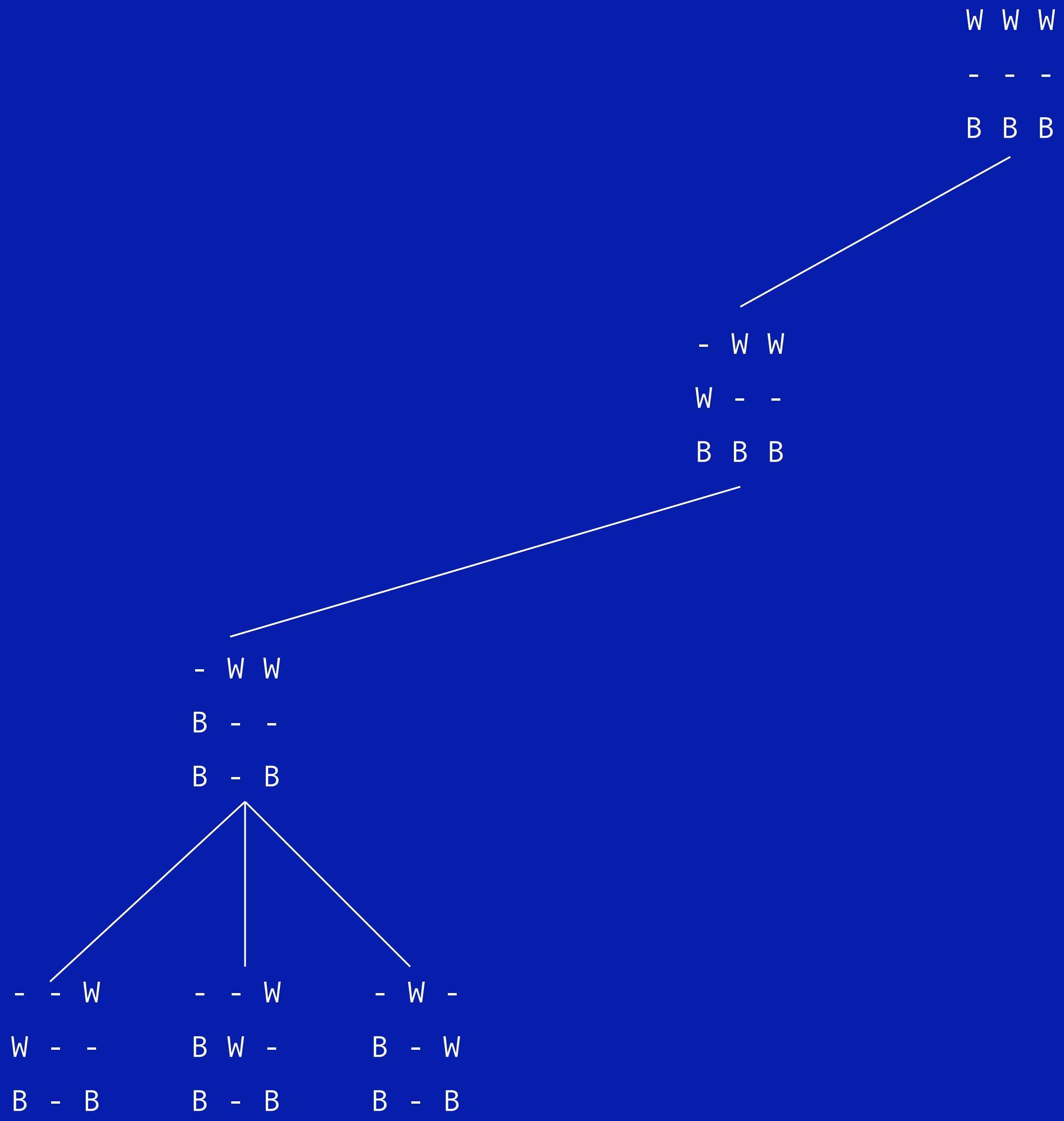

W W W
- - -
B B B
- W W
W - -
B B B
W - W
- W -
B B B
W W -
- - W
B B B

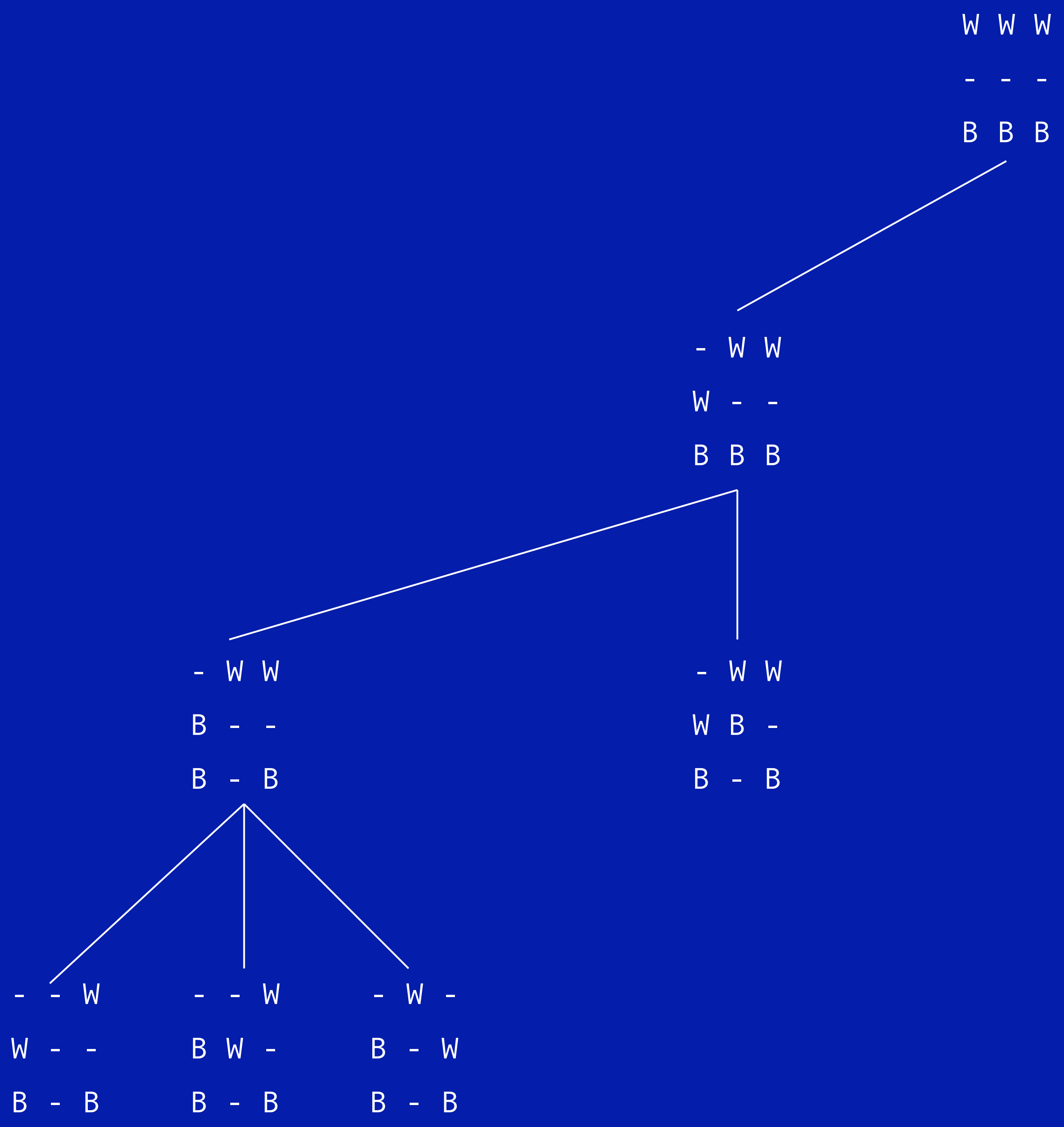

W W W
- - -
B B B
- W W
W - -
B B B

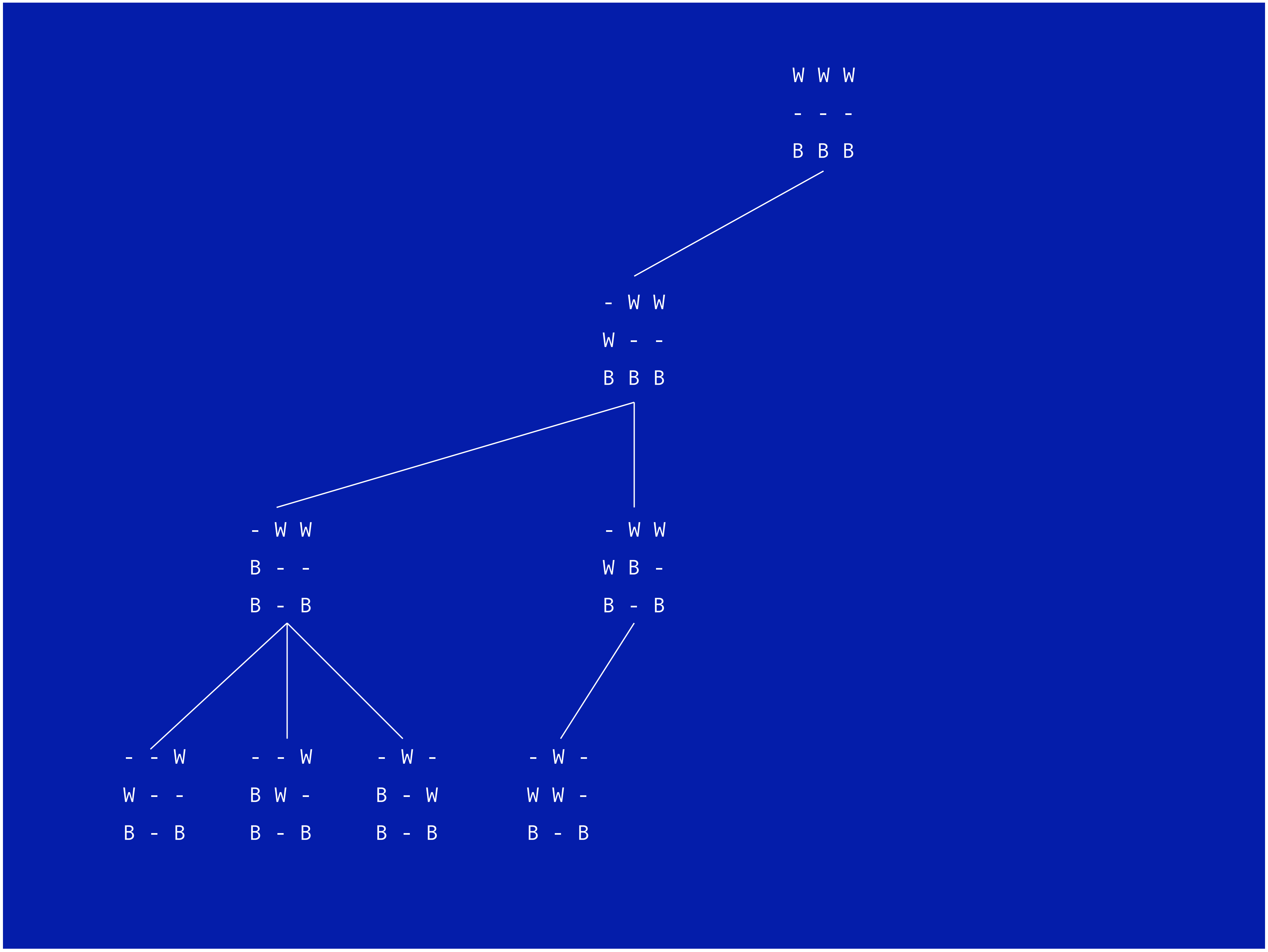

W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B

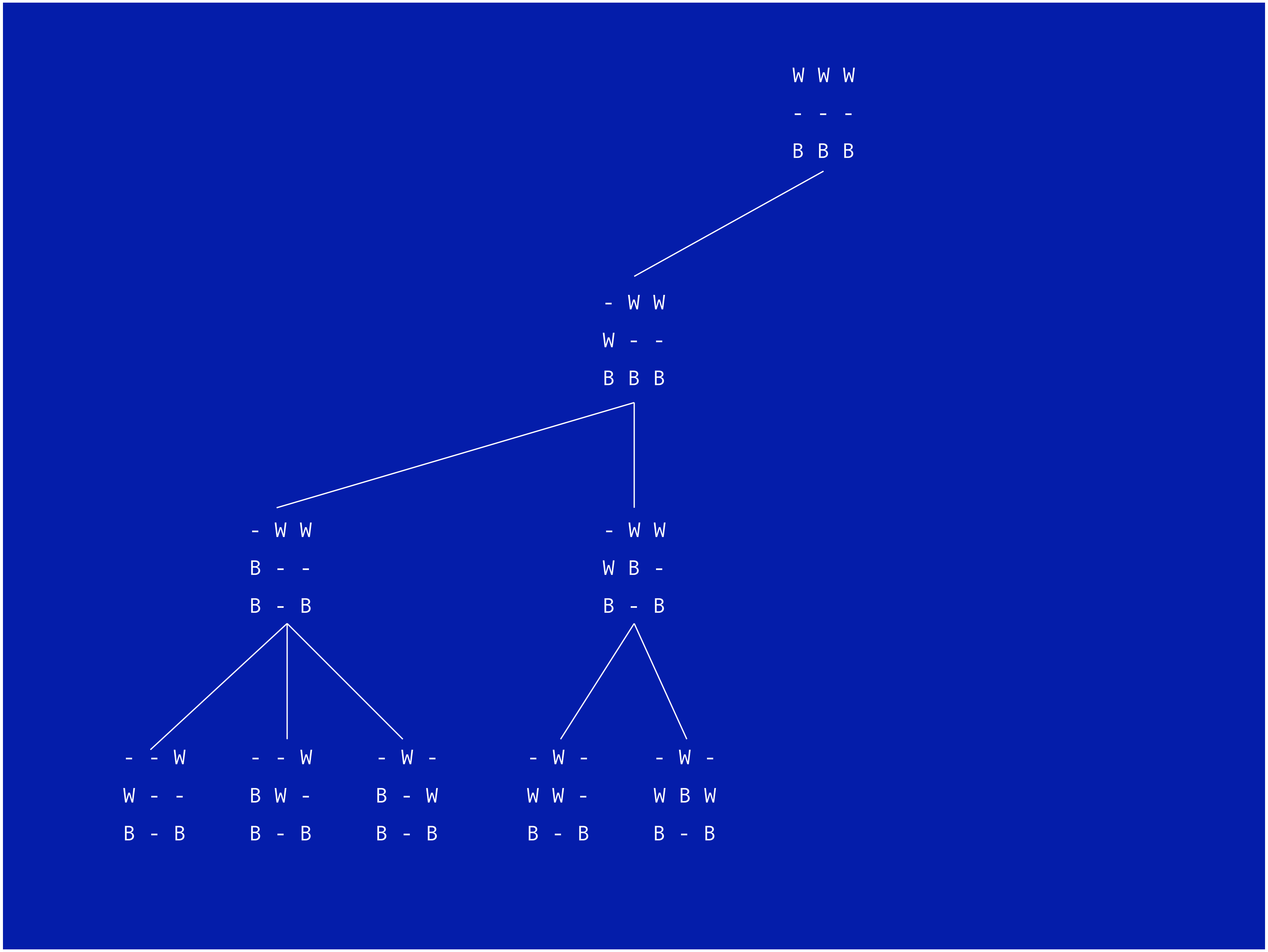

W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- - W
W - -
B - B

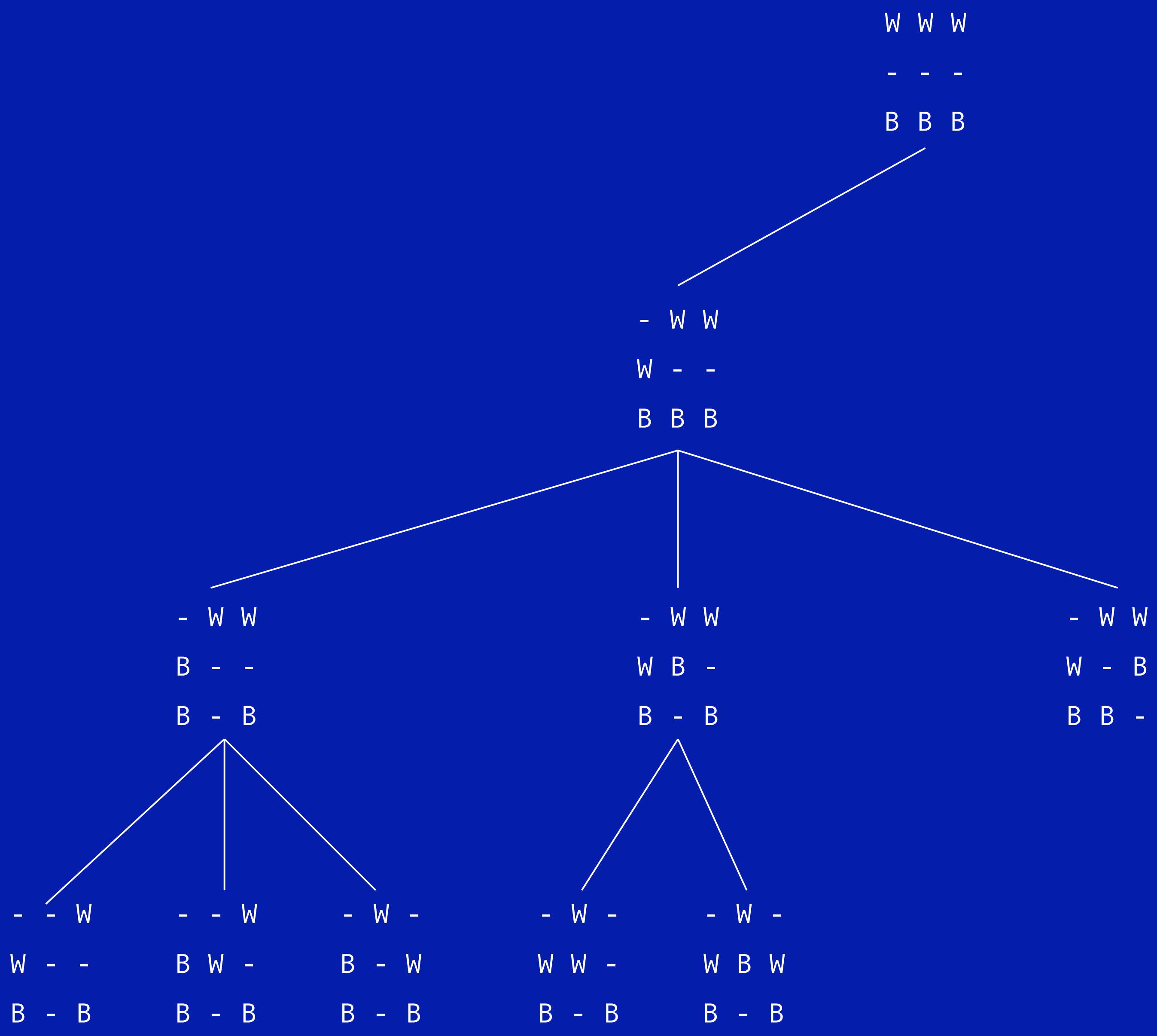

W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- - W
W - -
B - B
- - W
B W -
B - B

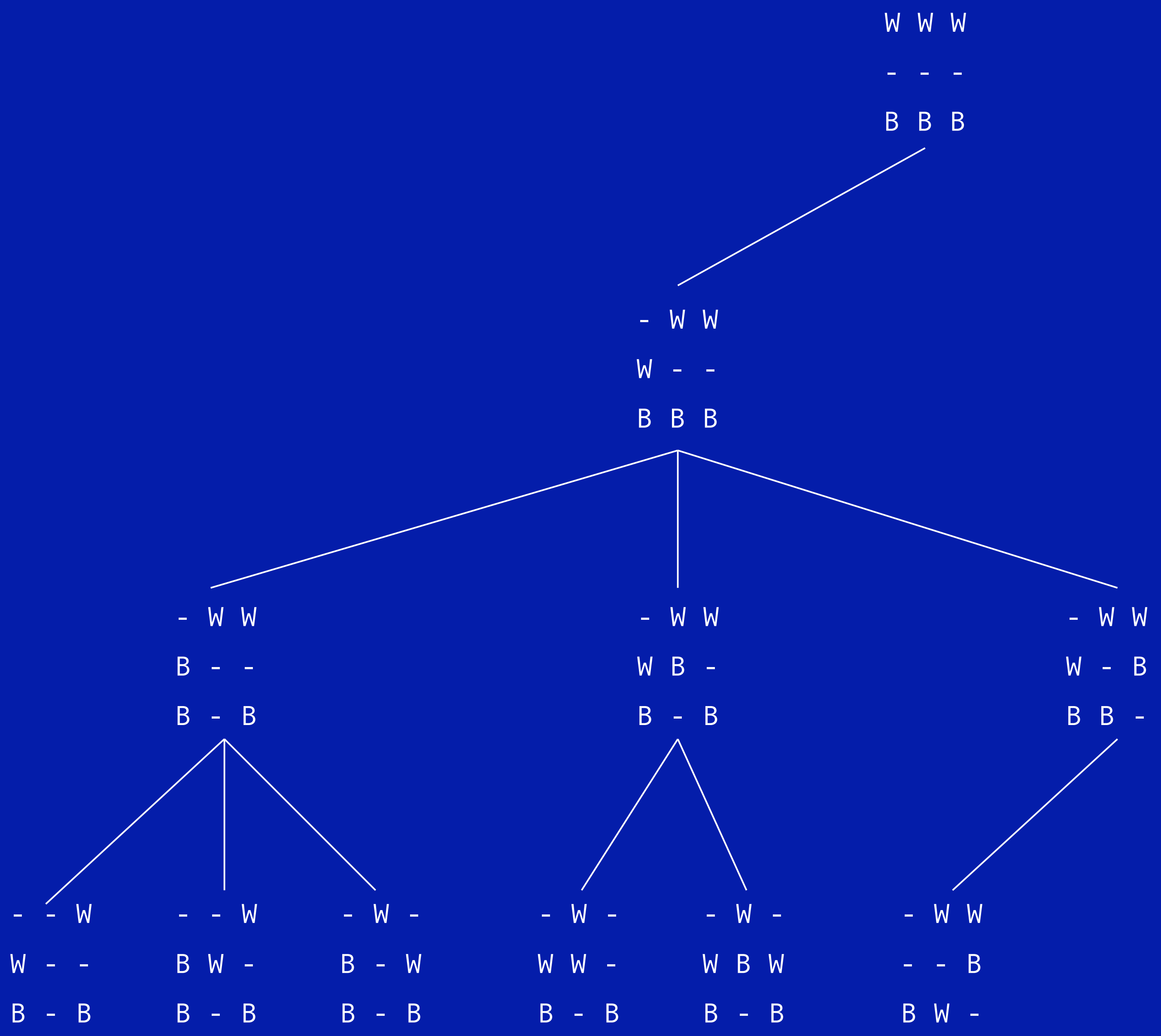

W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- - W
W - -
B - B
- - W
B W -
B - B
- W -
B - W
B - B

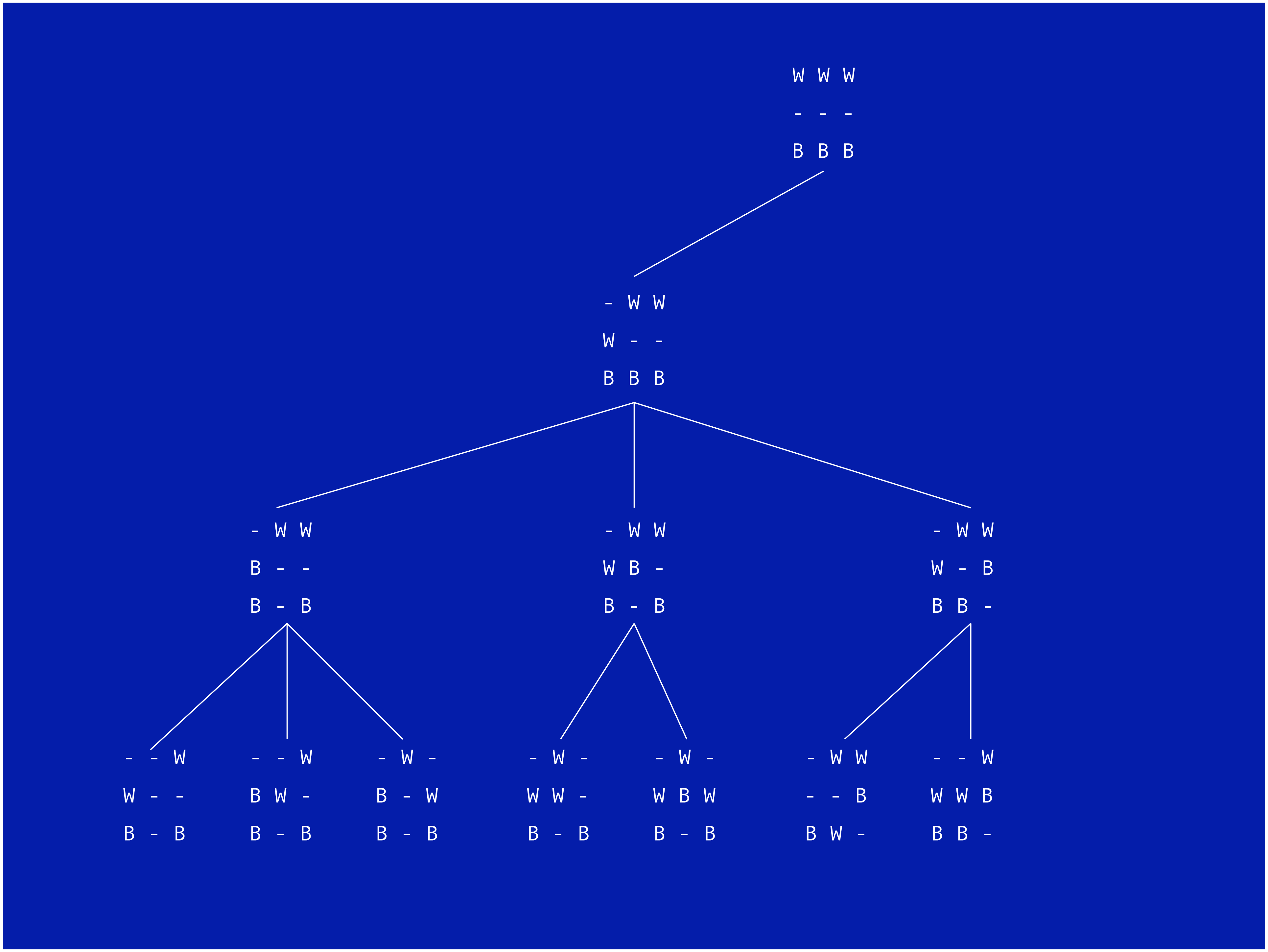

W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- - W
W - -
B - B
- - W
B W -
B - B
- W -
B - W
B - B


W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- - W
W - -
B - B
- - W
B W -
B - B
- W -
B - W
B - B
- W -
W W -
B - B

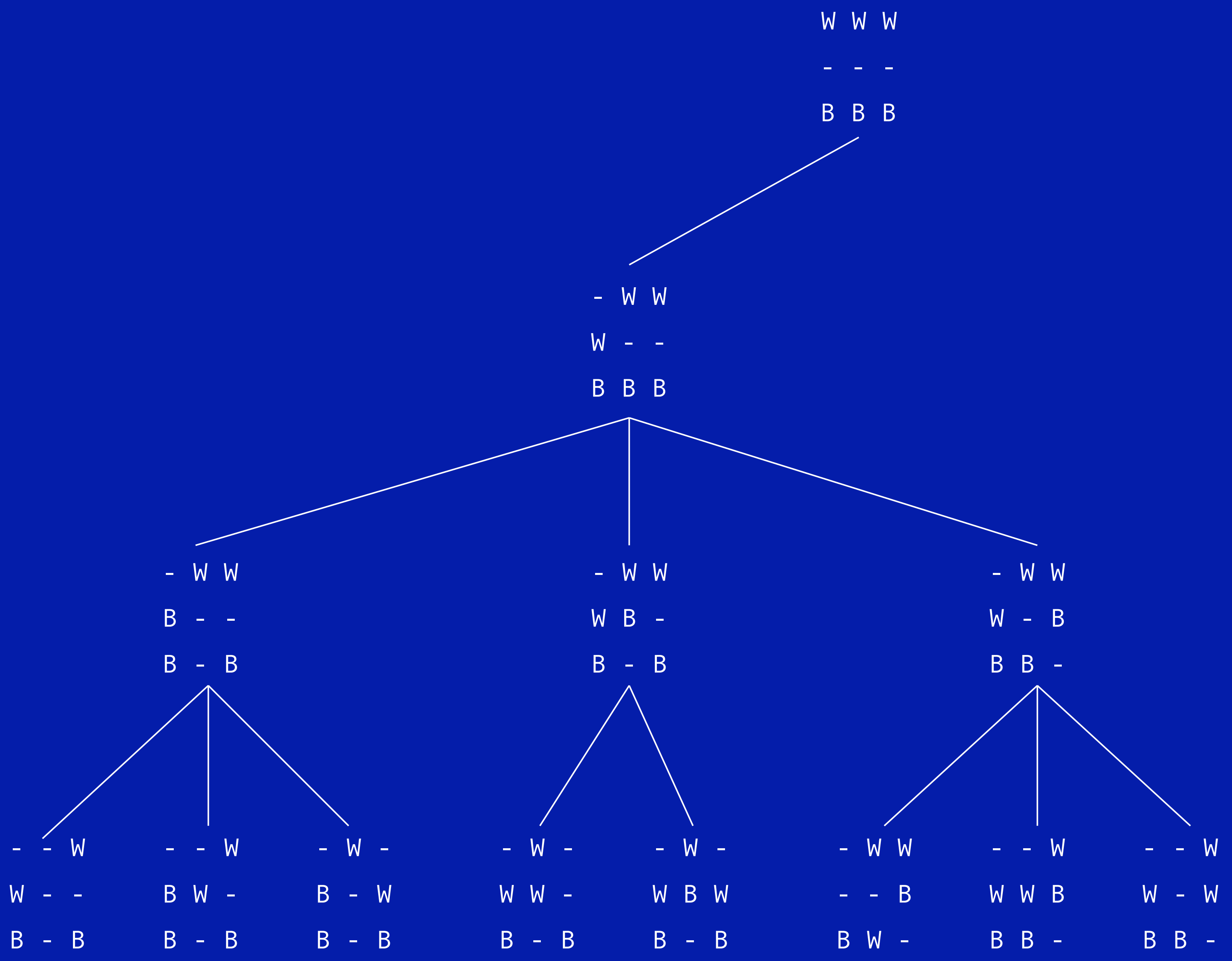

W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- - W
W - -
B - B
- - W
B W -
B - B
- W -
B - W
B - B
- W -
W W -
B - B
- W -
W B W
B - B


W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- W W
W - B
B B -
- - W
W - -
B - B
- - W
B W -
B - B
- W -
B - W
B - B
- W -
W W -
B - B
- W -
W B W
B - B


W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- W W
W - B
B B -
- - W
W - -
B - B
- - W
B W -
B - B
- W -
B - W
B - B
- W -
W W -
B - B
- W -
W B W
B - B
- W W
- - B
B W -



W W W / - - - / B B B
- W W / W - - / B B B
- W W / B - - / B - B
- W W / W B - / B - B
- W W / W - B / B B -
- - W / W - - / B - B
- - W / B W - / B - B
- W - / B - W / B - B
- W - / W W - / B - B
- W - / W B W / B - B
- W W / - - B / B W -
- - W / W W B / B B -


W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- W W
W - B
B B -
- - W
W - -
B - B
- - W
B W -
B - B
- W -
B - W
B - B
- W -
W W -
B - B
- W -
W B W
B - B
- W W
- - B
B W -
- - W
W W B
B B -
- - W
W - W
B B -

Two Questions

Two Questions

First, how deep do you search?

Two Questions

First, how deep do you search?

As deep as you can within computational constraints:

- time
- memory
- space on powerpoint slide

The deeper the search, the more informed is your answer to the next question…

Two Questions

Second, how do you know which move to make?

Two Questions

Second, how do you know which move to make?

Use heuristic knowledge, of course. In this case, we apply this very crude board evaluation function:

```
       if you have won then board value = +10
else   if opponent has won then board value = -10
else   board value = number of your pawns -
                      number of opponent’s pawns
```

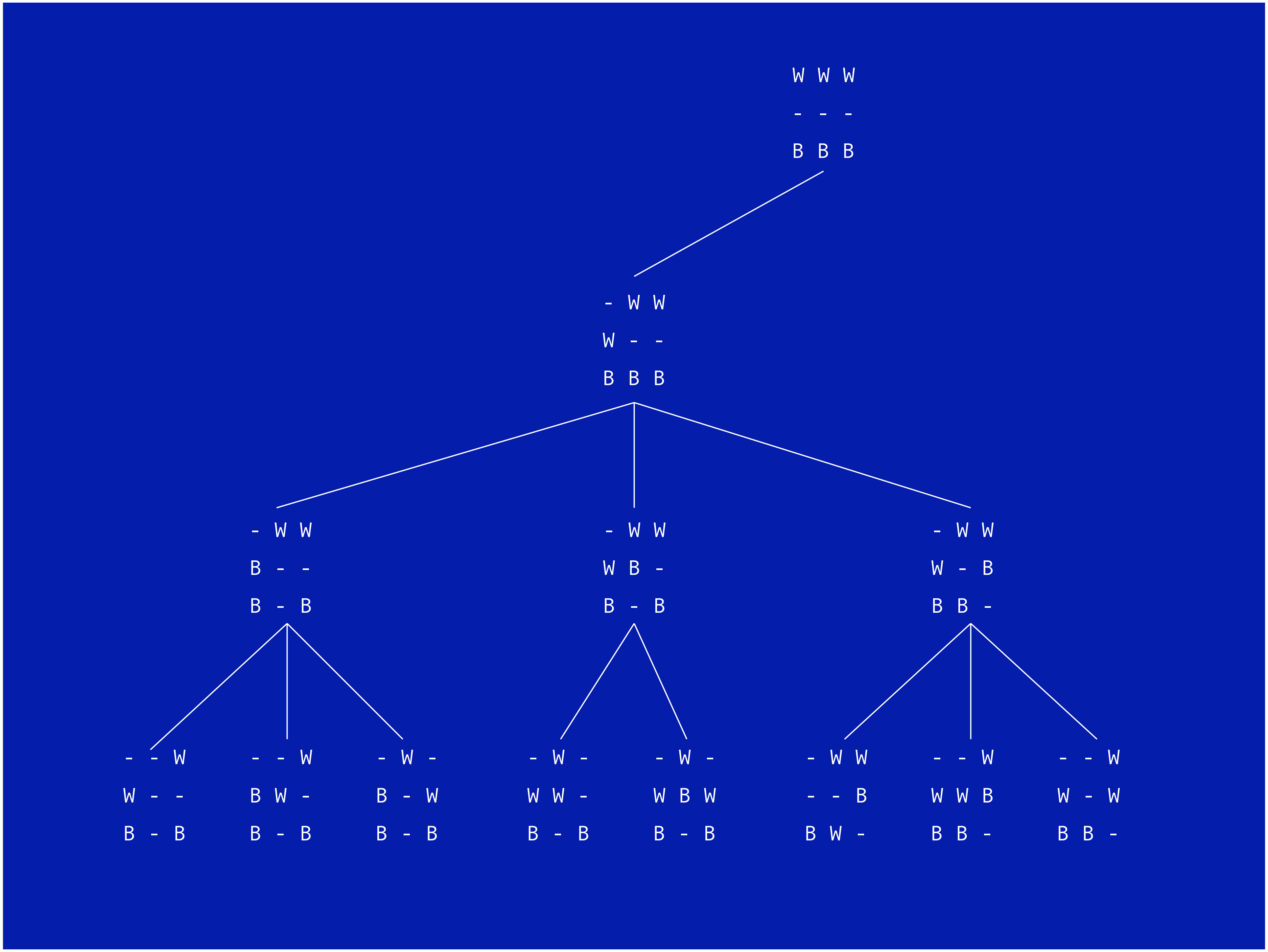
Two Questions

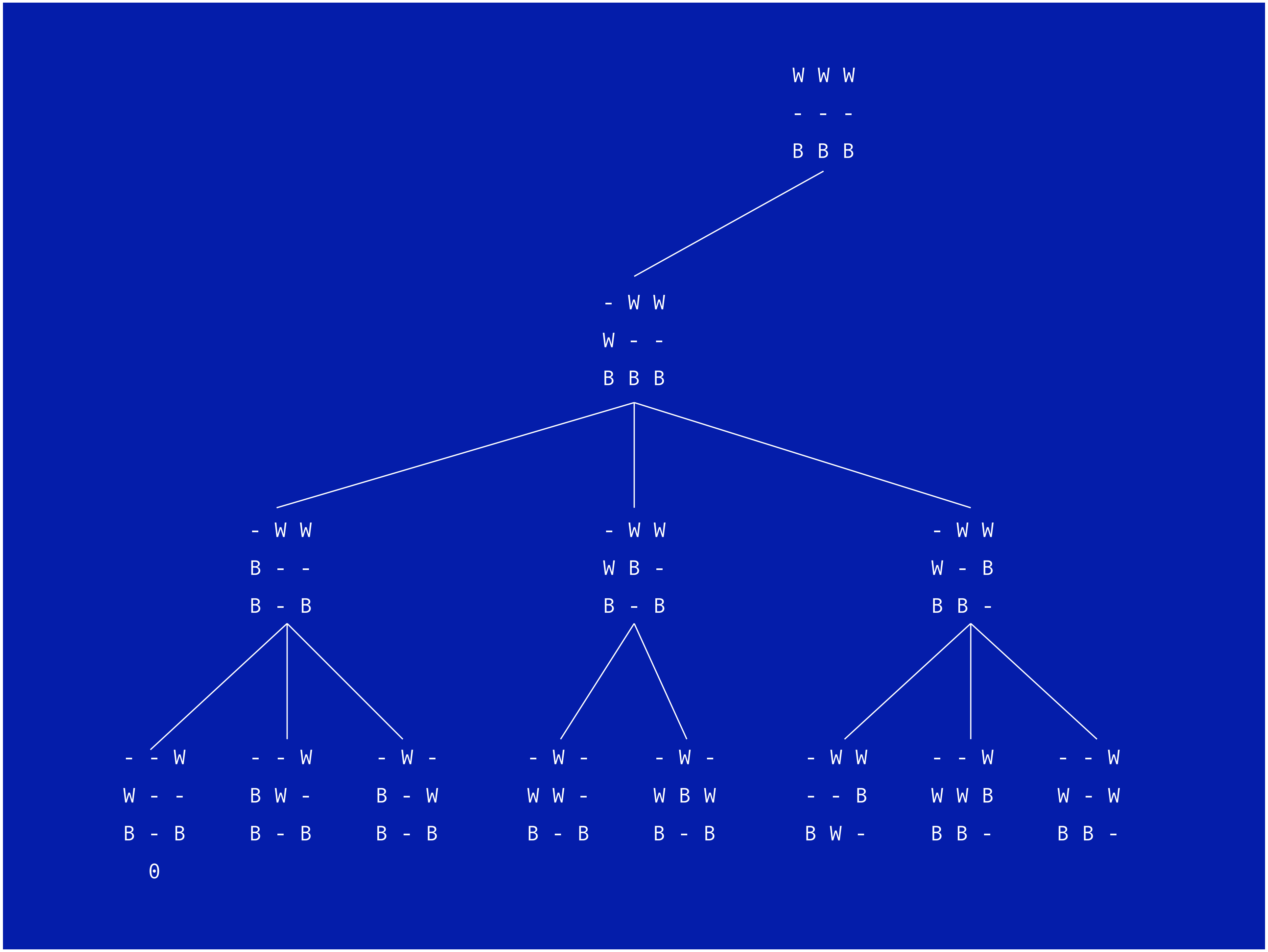
Second, how do you know which move to make?

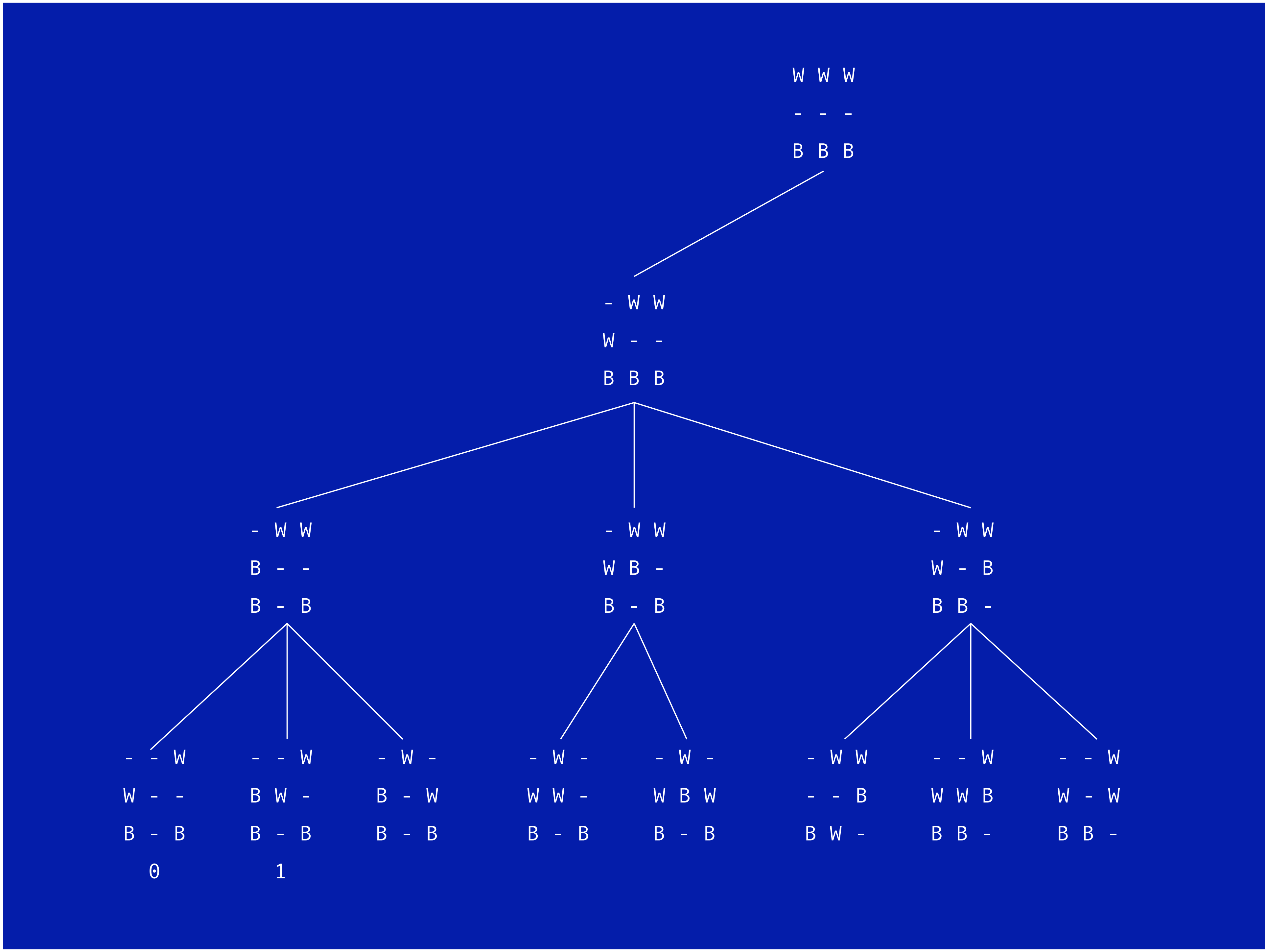
Use heuristic knowledge, of course! In this case, we apply this very crude board evaluation function:

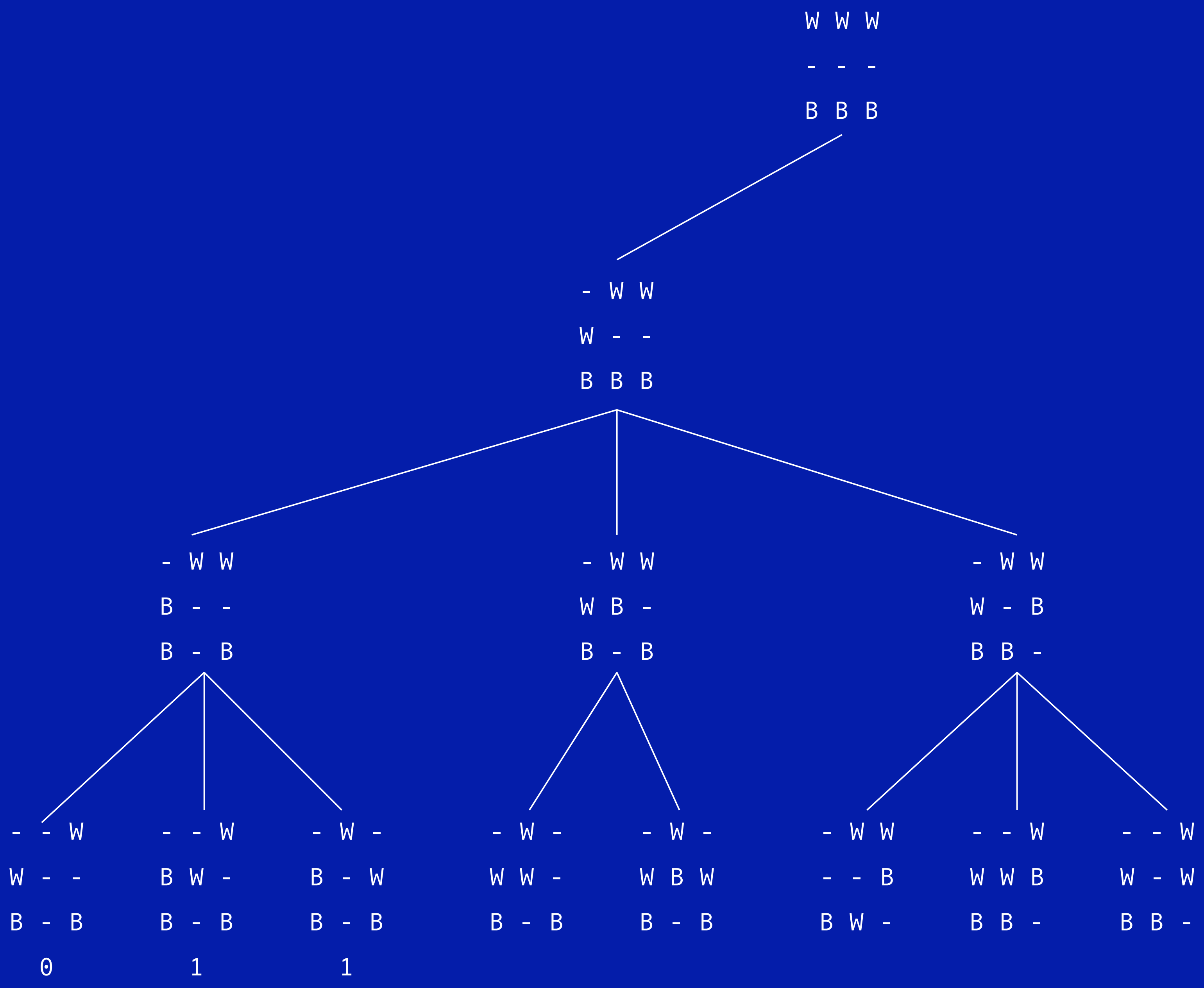
```
        if you have won then board value = +10
else    if opponent has won then board value = -10
else    board value = number of your pawns -
                          number of opponent’s pawns
```

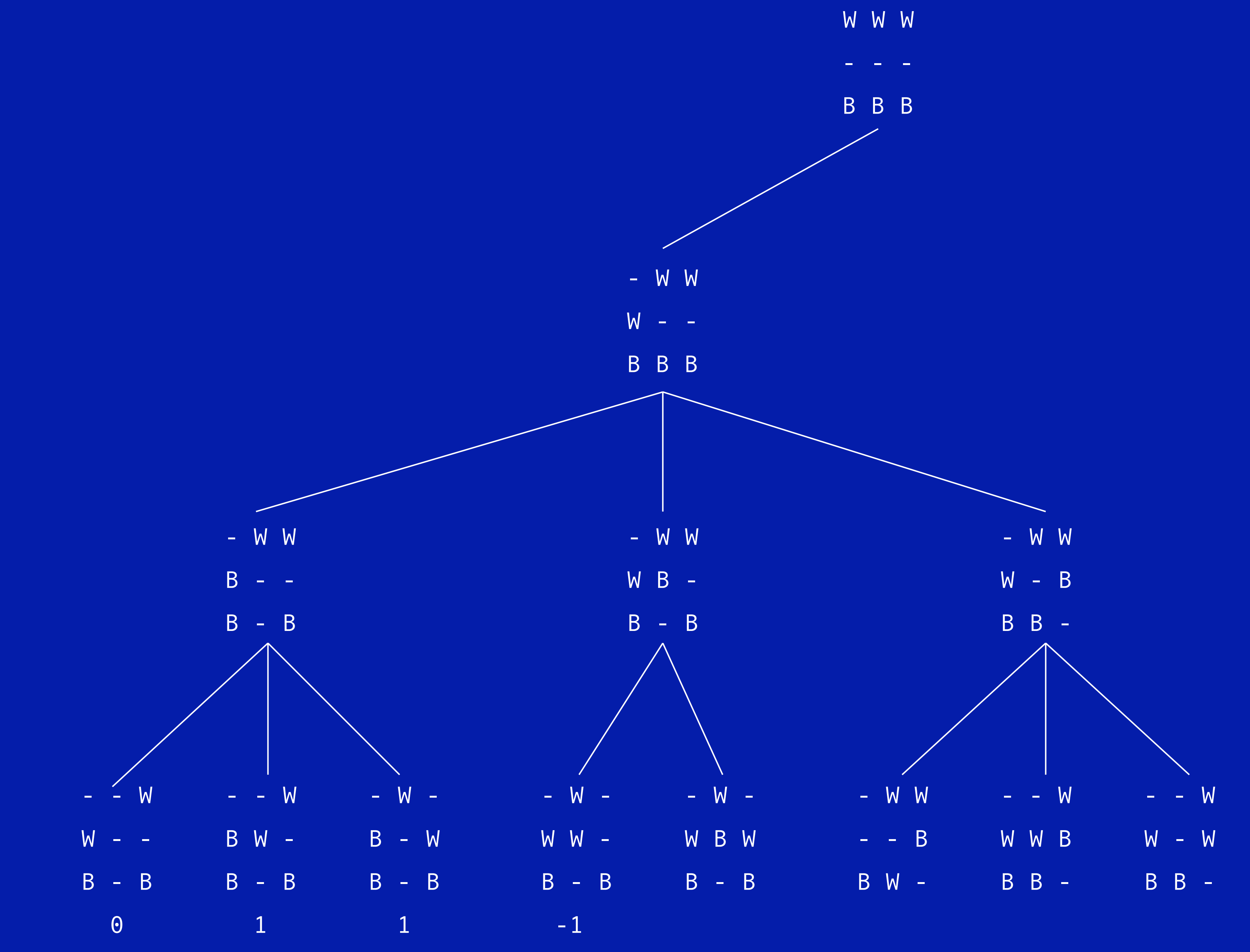
The board evaluation function is applied like this....


W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- W W
W - B
B B -
- - W
W - -
B - B
- - W
B W -
B - B
- W -
B - W
B - B
- W -
W W -
B - B
- W -
W B W
B - B
- W W
- - B
B W -
- - W
W W B
B B -
- - W
W - W
B B -

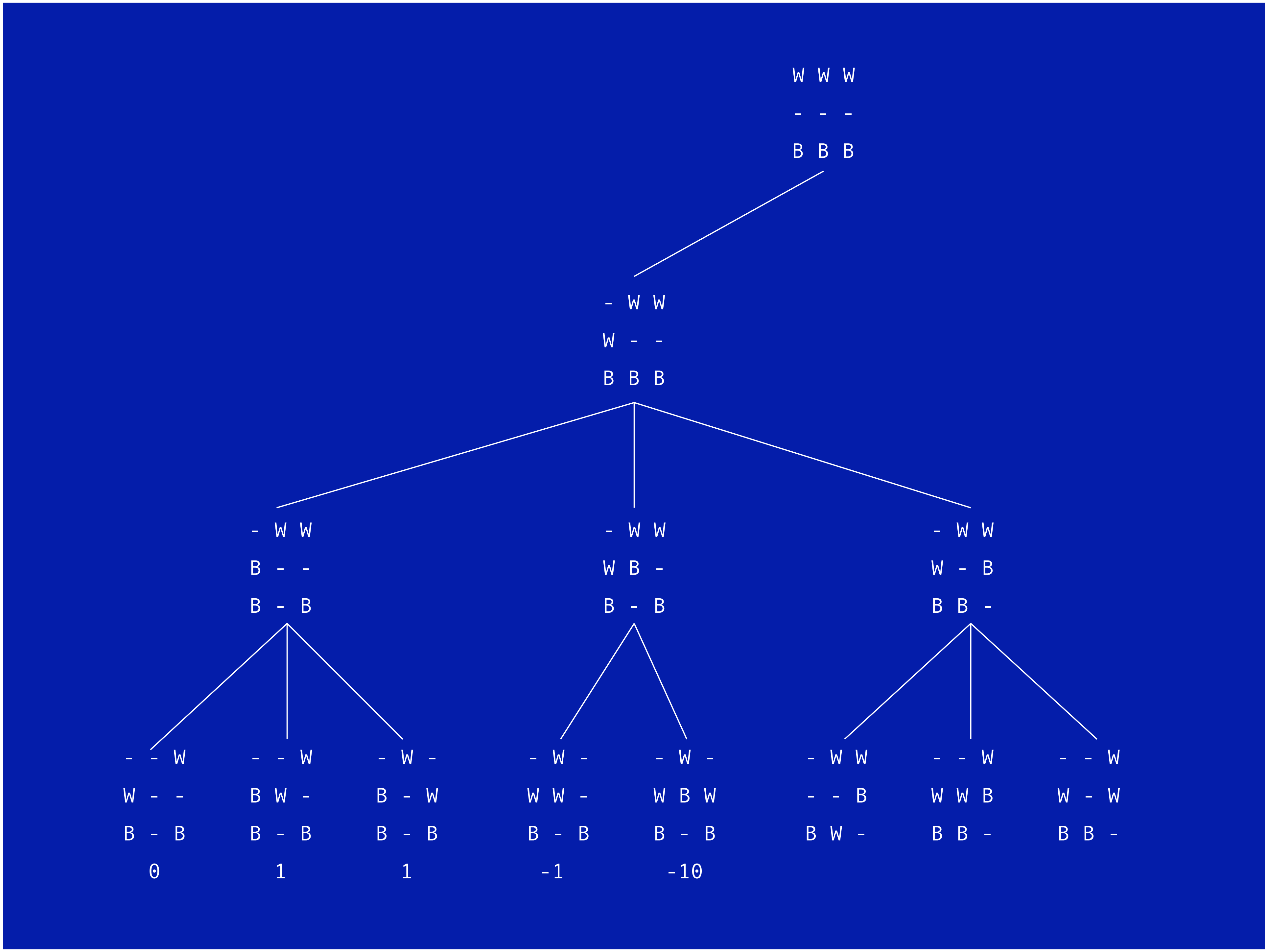

W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- W W
W - B
B B -
- - W
W - -
B - B
0
- - W
B W -
B - B
- W -
B - W
B - B
- W -
W W -
B - B
- W -
W B W
B - B
- W W
- - B
B W -
- - W
W W B
B B -
- - W
W - W
B B -

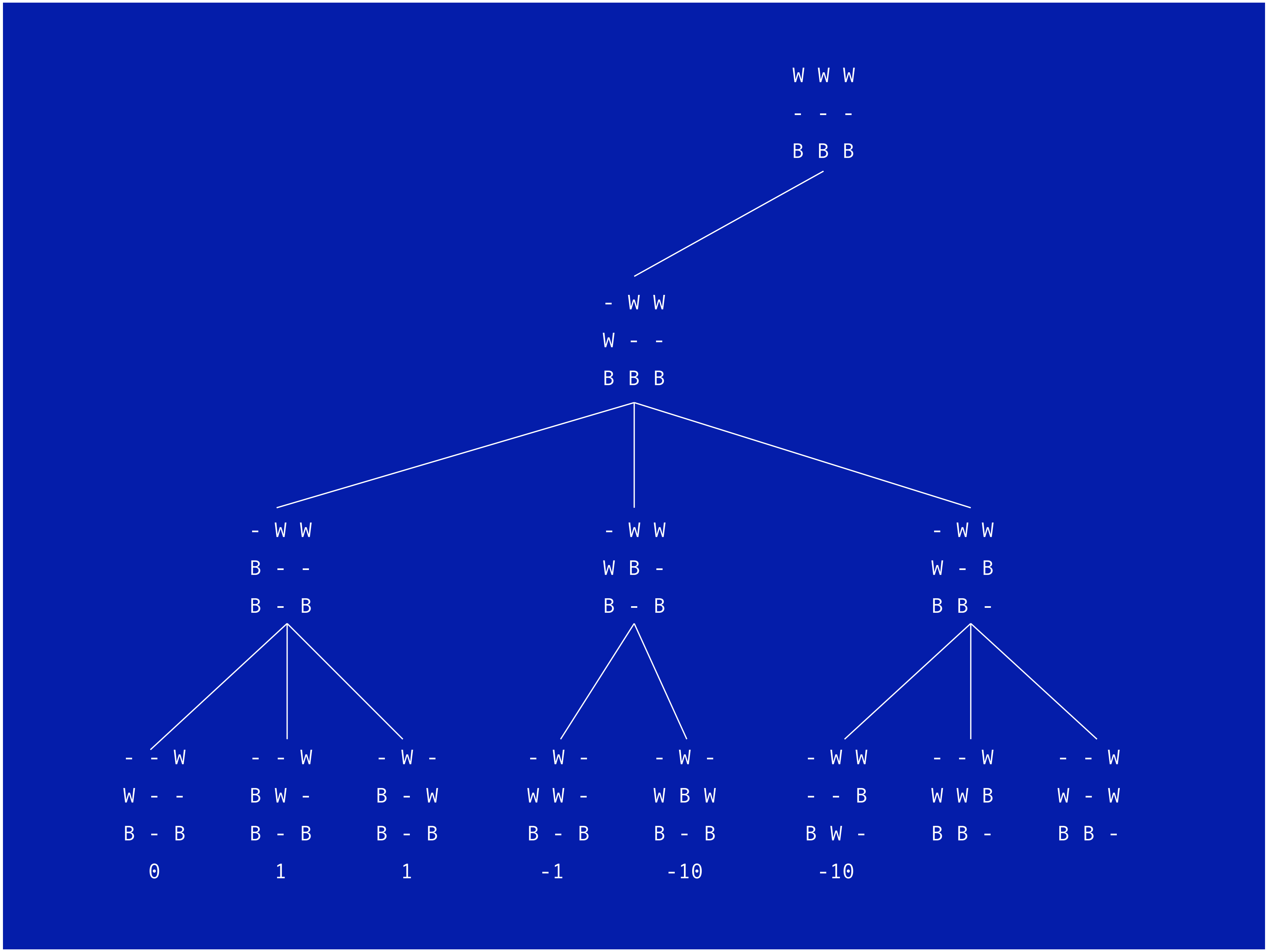

W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- W W
W - B
B B -
- - W
W - -
B - B
0
- - W
B W -
B - B
1
- W -
B - W
B - B
- W -
W W -
B - B
- W -
W B W
B - B
- W W
- - B
B W -
- - W
W W B
B B -
- - W
W - W
B B -

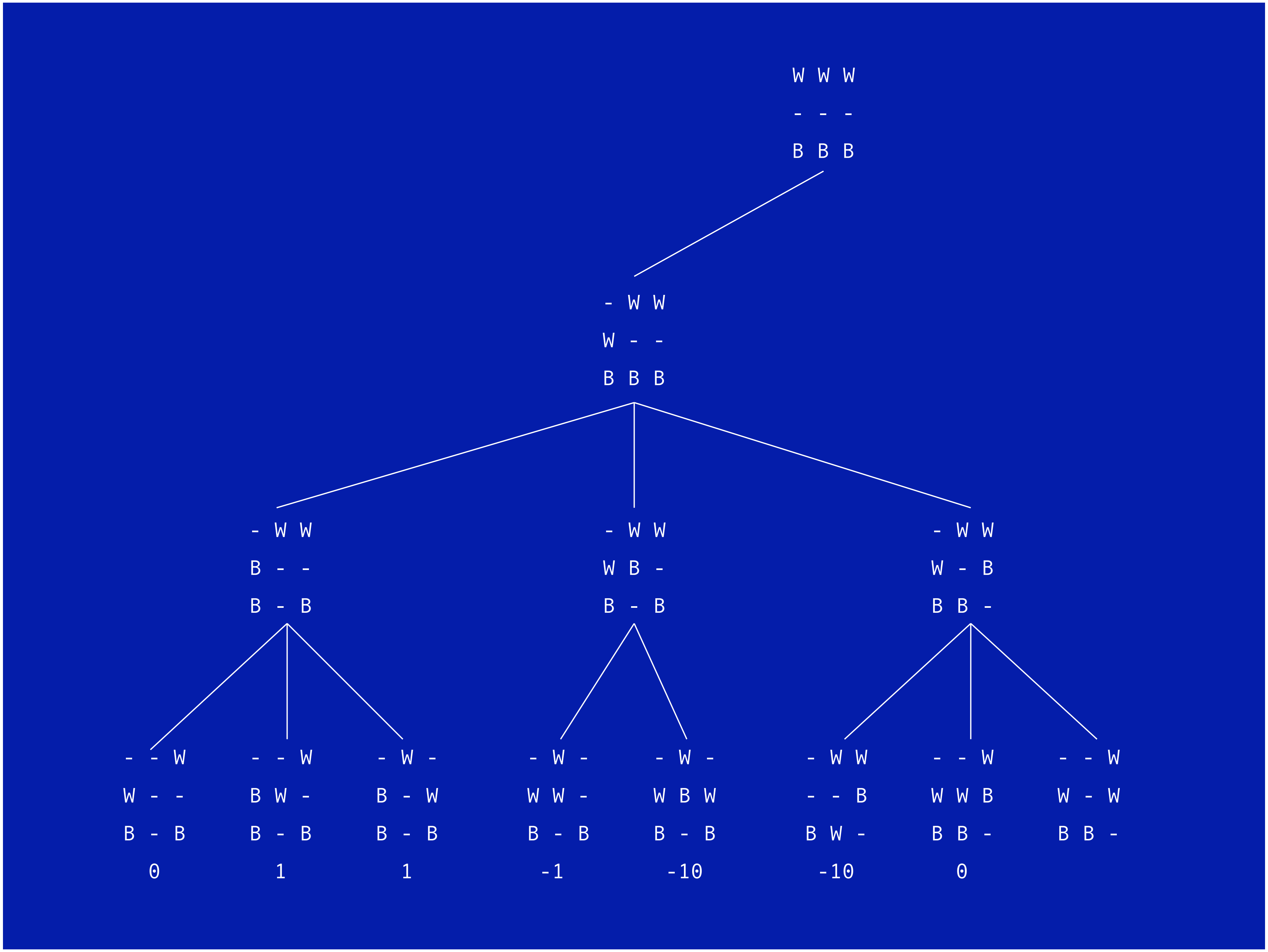

W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- W W
W - B
B B -
- - W
W - -
B - B
0
- - W
B W -
B - B
1
- W -
B - W
B - B
1
- W -
W W -
B - B
- W -
W B W
B - B
- W W
- - B
B W -
- - W
W W B
B B -
- - W
W - W
B B -

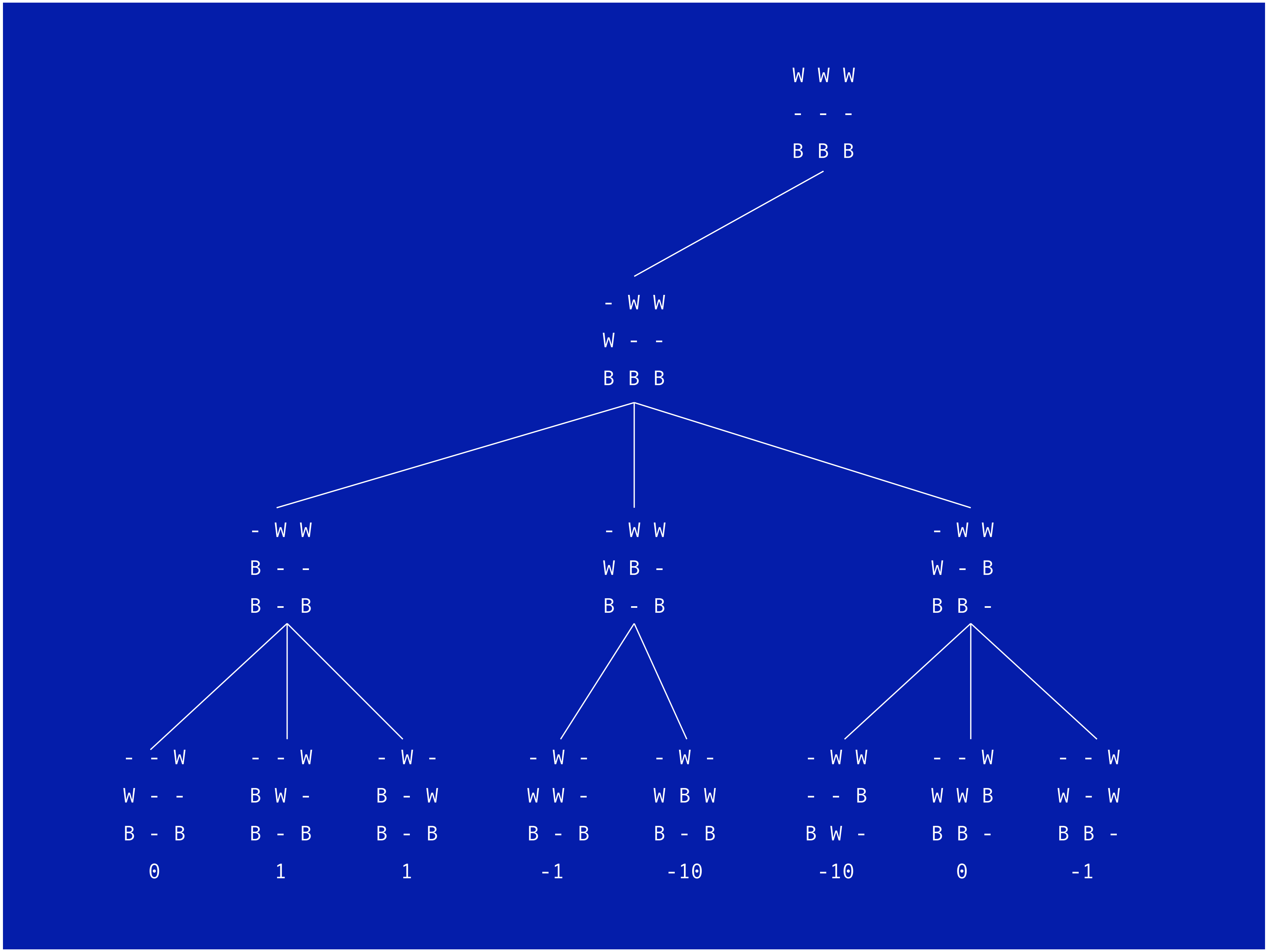


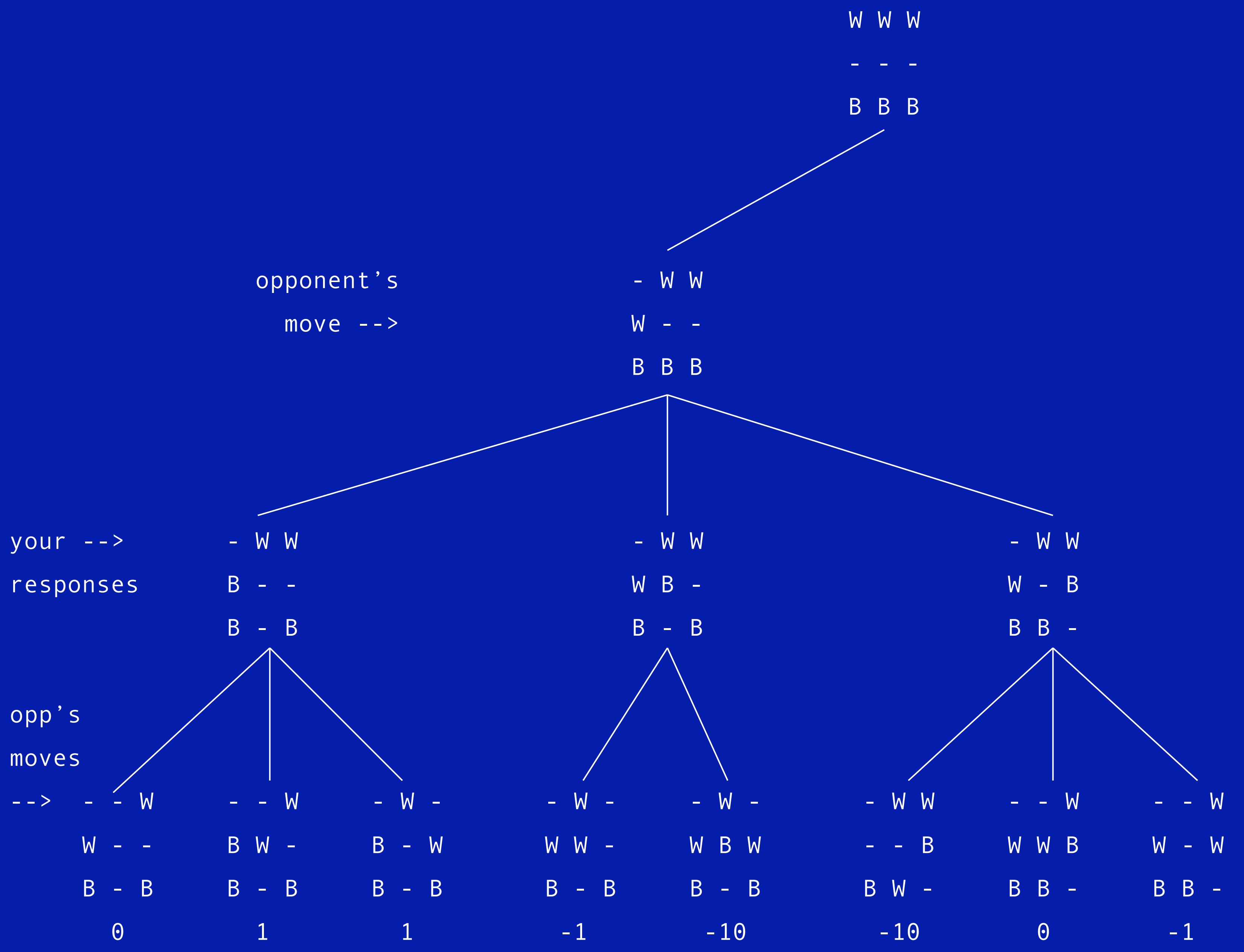
W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- W W
W - B
B B -
- - W
W - -
B - B
0
- - W
B W -
B - B
1
- W -
B - W
B - B
1
- W -
W W -
B - B
-1
- W -
W B W
B - B
- W W
- - B
B W -
- - W
W W B
B B -
- - W
W - W
B B -

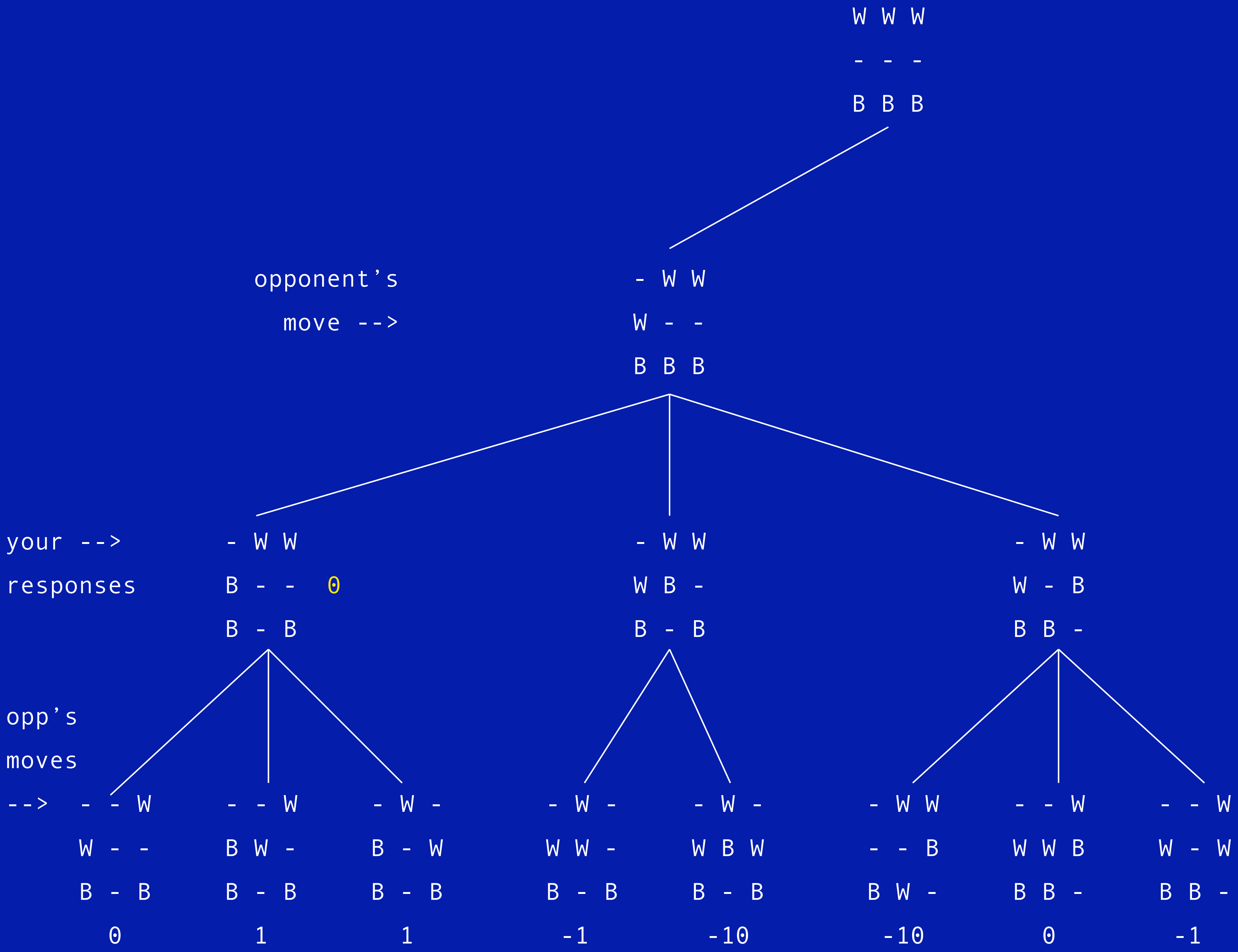

W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- W W
W - B
B B -
- - W
W - -
B - B
0
- - W
B W -
B - B
1
- W -
B - W
B - B
1
- W -
W W -
B - B
-1
- W -
W B W
B - B
-10
- W W
- - B
B W -
- - W
W W B
B B -
- - W
W - W
B B -

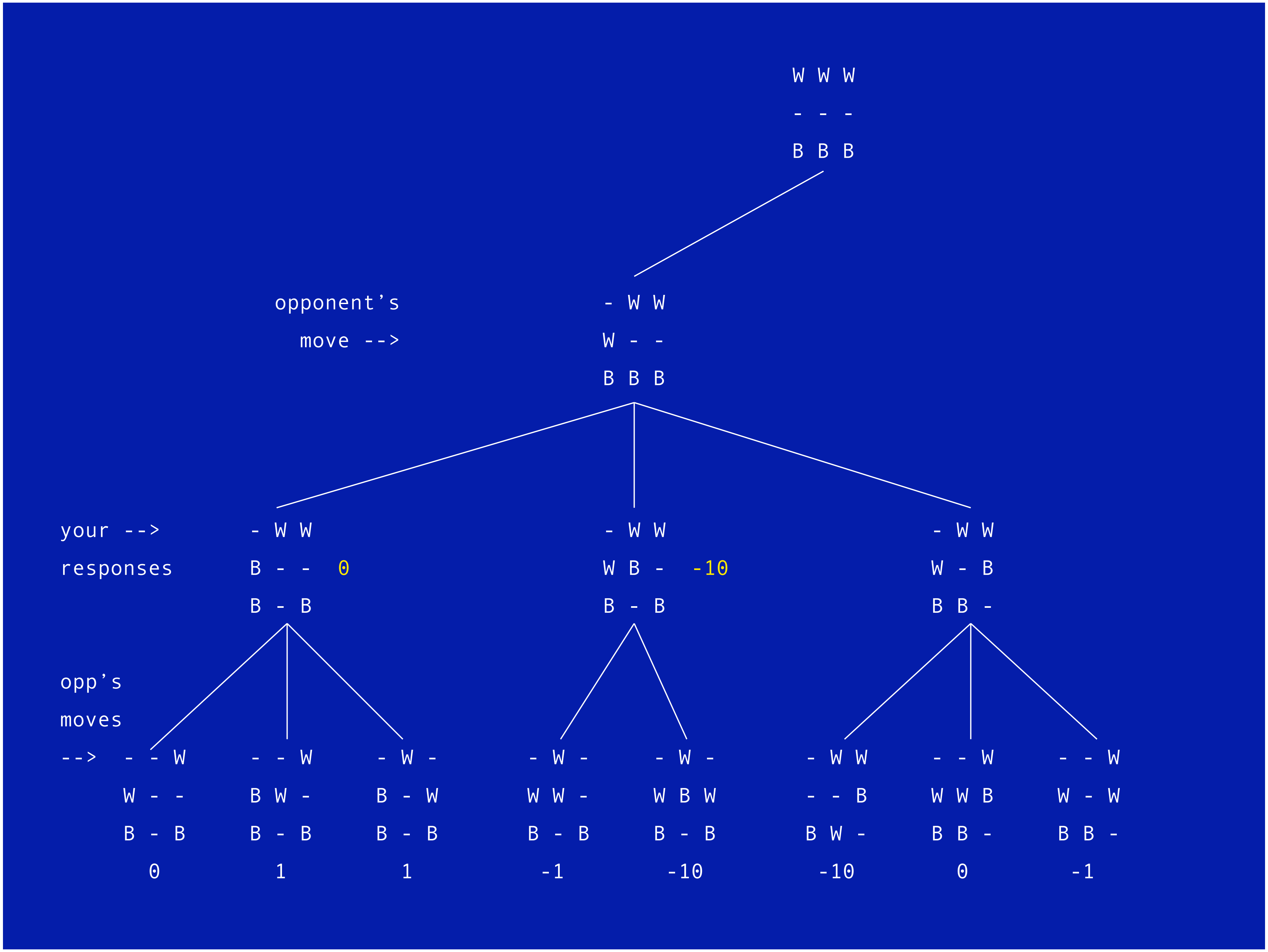

W W W / - - - / B B B
- W W / W - - / B B B
- W W / B - - / B - B
- W W / W B - / B - B
- W W / W - B / B B -
- - W / W - - / B - B
0
- - W / B W - / B - B
1
- W - / B - W / B - B
1
- W - / W W - / B - B
-1
- W - / W B W / B - B
-10
- W W / - - B / B W -
-10
- - W / W W B / B B -
- - W / W - W / B B -

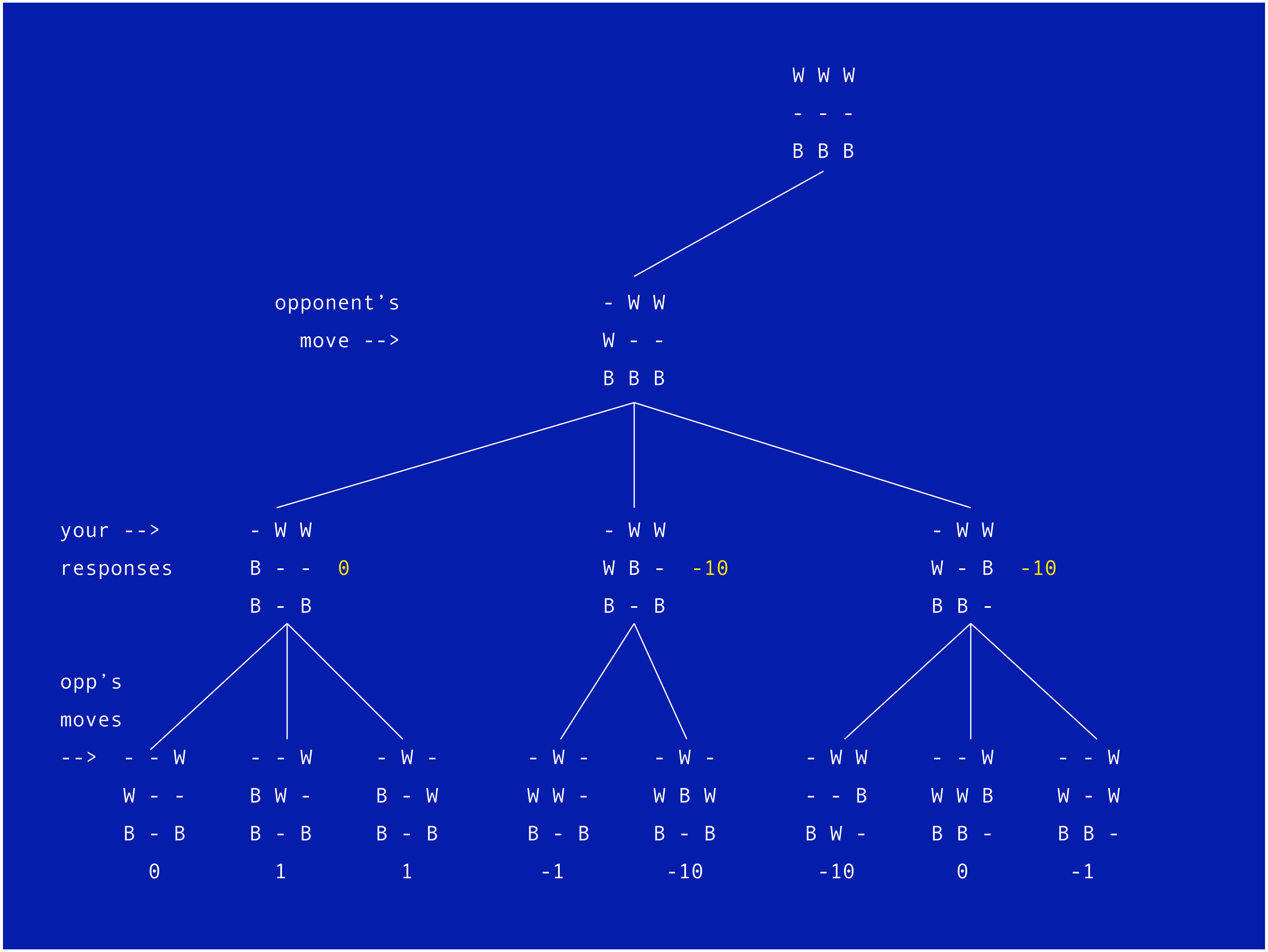

W W W
- - -
B B B
- W W
W - -
B B B
- W W
B - -
B - B
- W W
W B -
B - B
- W W
W - B
B B -
- - W
W - -
B - B
0
- - W
B W -
B - B
1
- W -
B - W
B - B
1
- W -
W W -
B - B
-1
- W -
W B W
B - B
-10
- W W
- - B
B W -
-10
- - W
W W B
B B -
0
- - W
W - W
B B -

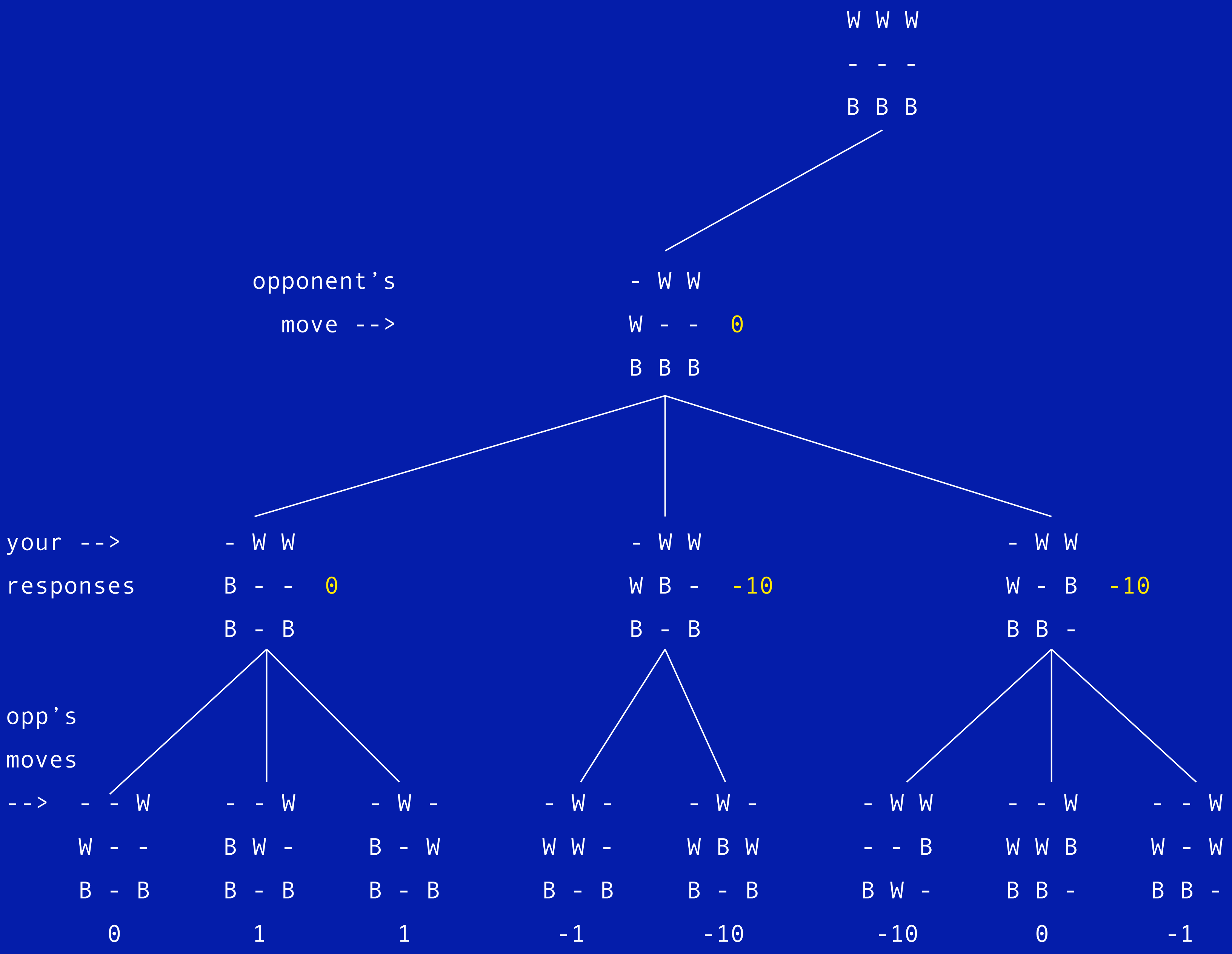

W W W / - - - / B B B
- W W / W - - / B B B
- W W / B - - / B - B
- W W / W B - / B - B
- W W / W - B / B B -
- - W / W - - / B - B
0
- - W / B W - / B - B
1
- W - / B - W / B - B
1
- W - / W W - / B - B
-1
- W - / W B W / B - B
-10
- W W / - - B / B W -
-10
- - W / W W B / B B -
0
- - W / W - W / B B -
-1

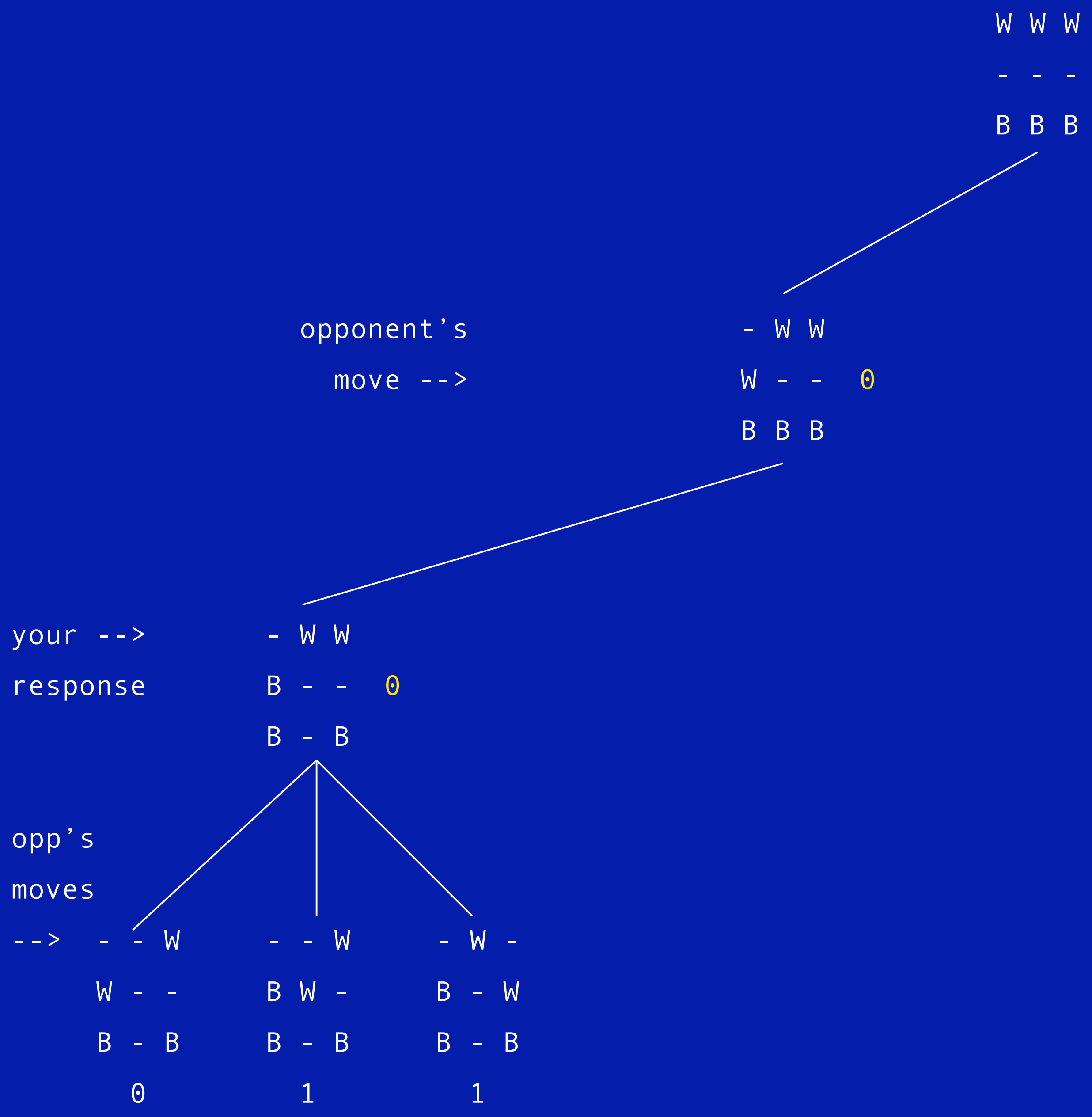

W W W
- - -
B B B
opponent's move -->
- W W
W - -
B B B
your --> responses
- W W
B - -
B - B
- W W
W B -
B - B
- W W
W - B
B B -
opp's moves -->
- - W
W - -
B - B
0
- - W
B W -
B - B
1
- W -
B - W
B - B
1
- W -
W W -
B - B
-1
- W -
W B W
B - B
-10
- W W
- - B
B W -
-10
- - W
W W B
B B -
0
- - W
W - W
B B -
-1


W W W
- - -
B B B
opponent's move -->
- W W
W - -
B B B
your --> responses
- W W
B - -
B - B
0
- W W
W B -
B - B
- W W
W - B
B B -
opp's moves -->
- - W
W - -
B - B
0
- - W
B W -
B - B
1
- W -
B - W
B - B
1
- W -
W W -
B - B
-1
- W -
W B W
B - B
-10
- W W
- - B
B W -
-10
- - W
W W B
B B -
0
- - W
W - W
B B -
-1

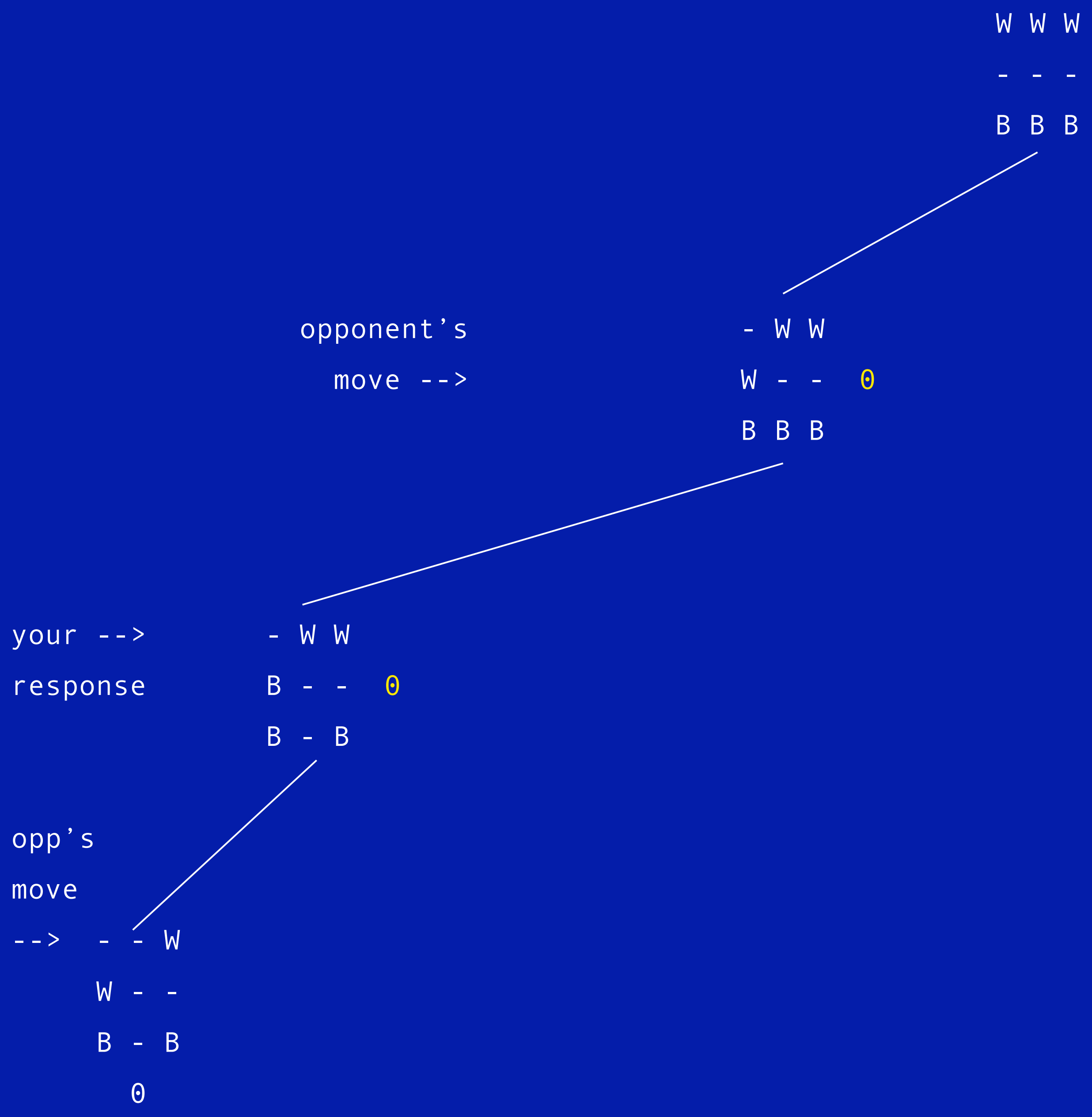

W W W
- - -
B B B
opponent's move -->
- W W
W - -
B B B
your --> responses
- W W
B - - 0
B - B
- W W
W B - -10
B - B
- W W
W - B
B B -
opp's moves -->
- - W
W - -
B - B
0
- - W
B W -
B - B
1
- W -
B - W
B - B
1
- W -
W W -
B - B
-1
- W -
W B W
B - B
-10
- W W
- - B
B W -
-10
- - W
W W B
B B -
0
- - W
W - W
B B -
-1


W W W
- - -
B B B
opponent's move -->
- W W
W - -
B B B
your --> responses
- W W
B - - 0
B - B
- W W
W B - -10
B - B
- W W
W - B -10
B B -
opp's moves -->
- - W
W - -
B - B
0
- - W
B W -
B - B
1
- W -
B - W
B - B
1
- W -
W W -
B - B
-1
- W -
W B W
B - B
-10
- W W
- - B
B W -
-10
- - W
W W B
B B -
0
- - W
W - W
B B -
-1


W W W
- - -
B B B
opponent's move -->
- W W
W - - 0
B B B
your --> responses
- W W
B - - 0
B - B
- W W
W B - -10
B - B
- W W
W - B -10
B B -
opp's moves -->
- - W
W - -
B - B
0
- - W
B W -
B - B
1
- W -
B - W
B - B
1
- W -
W W -
B - B
-1
- W -
W B W
B - B
-10
- W W
- - B
B W -
-10
- - W
W W B
B B -
0
- - W
W - W
B B -
-1



W W W
- - -
B B B
opponent's
move -->
- W W
W - - 0
B B B
your -->
response
- W W
B - - 0
B - B
opp's
moves
-->
- - W
W - -
B - B
0
- - W
B W -
B - B
1
- W -
B - W
B - B
1


W W W
- - -
B B B
opponent's move -->
- W W
W - - 0
B B B
your --> response
- W W
B - - 0
B - B
opp's move -->
- - W
W - -
B - B
0

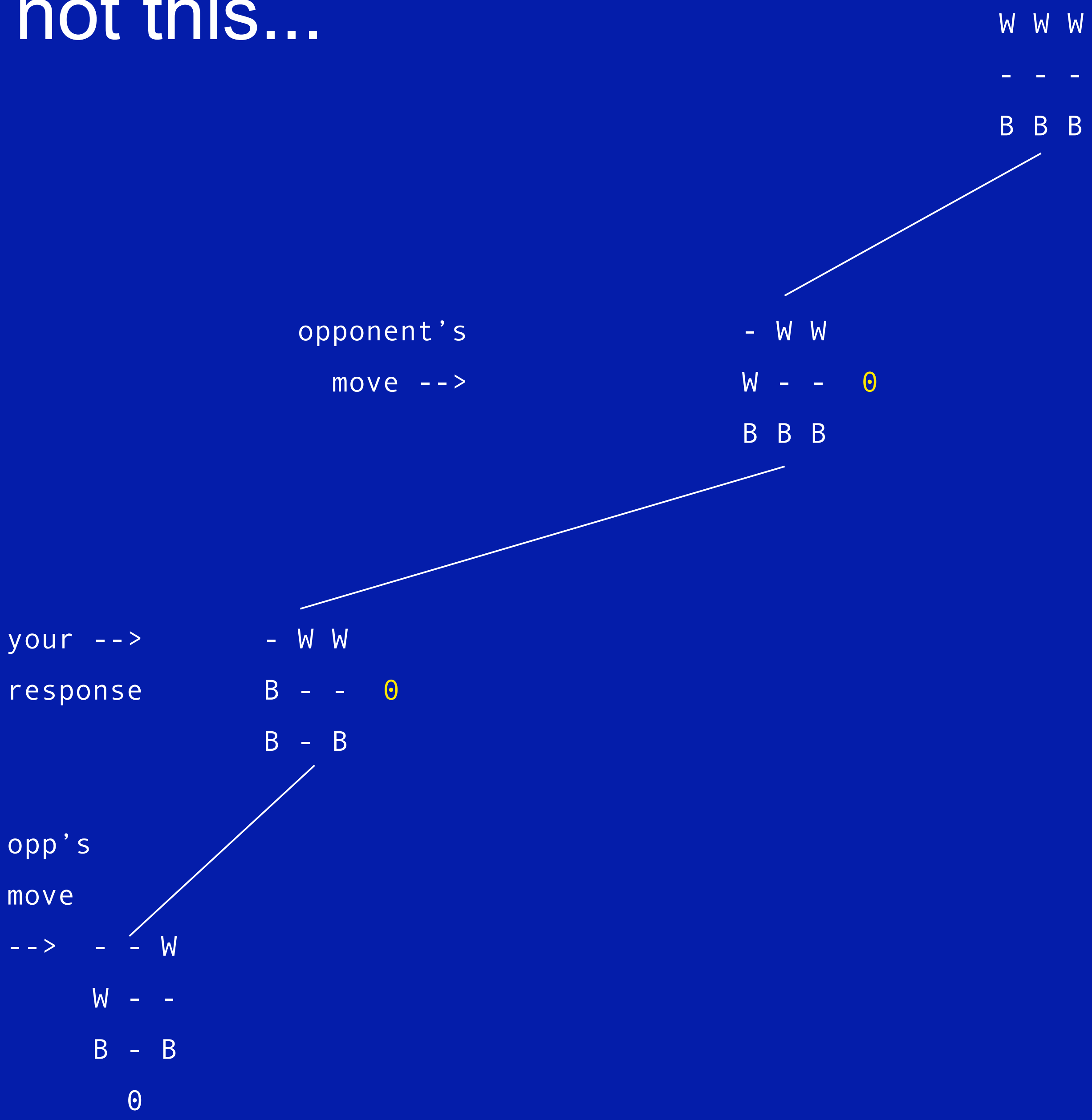
What happens next?

```
    your                 - W W
    move -->             B - -
                         B - B
                           |
                           |
                           |
opponent's               - - W
    move -->             W - -
                         B - B
```

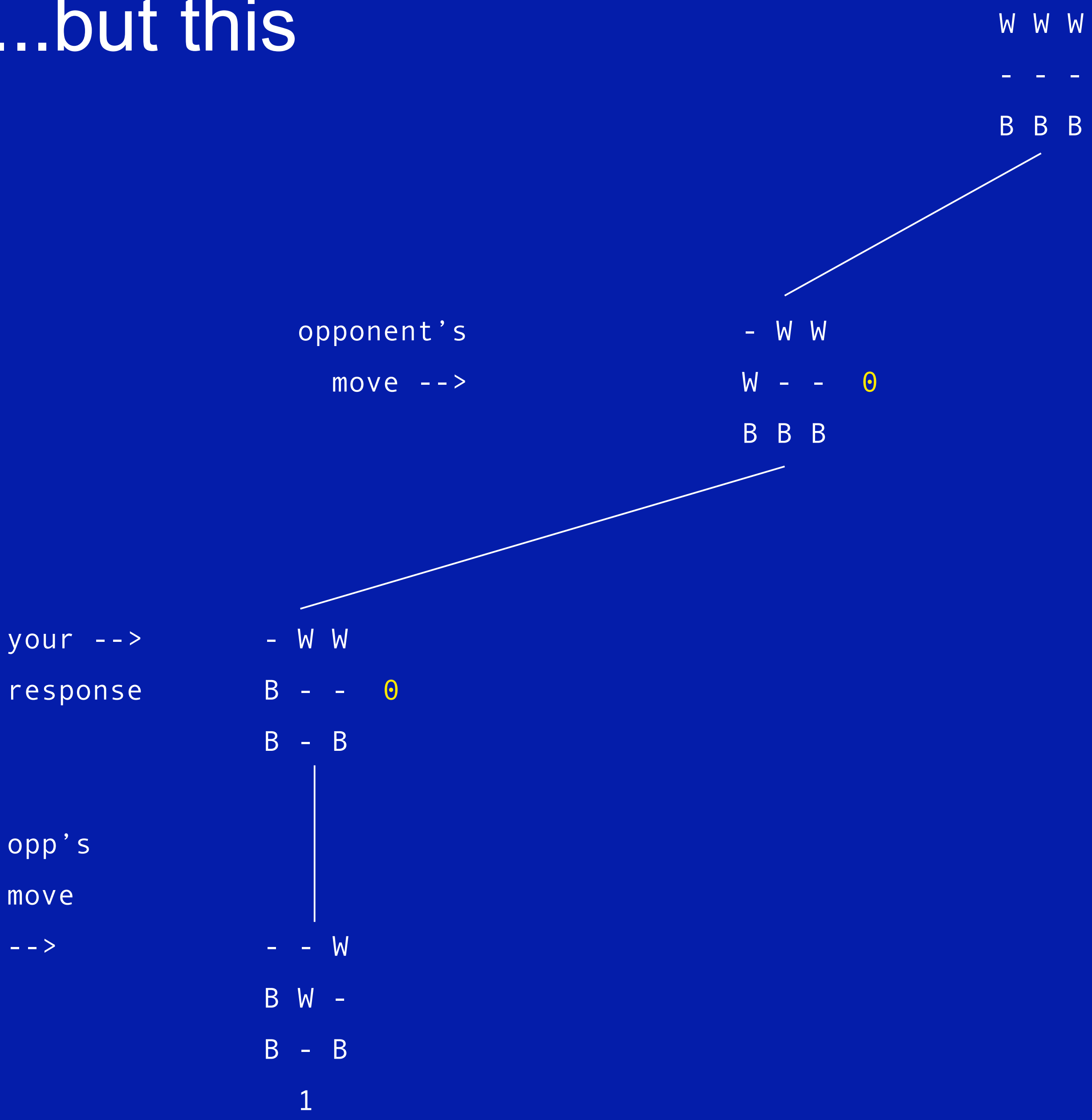
```
  your              - W W
  move -->          B - -
                    B - B
                      |
                      |
                      |
opponent's          - - W
  move -->          W - -
                    B - B
                      |
                      |
                      |
  your              - - W
responses -->       W - B
                    B - -
  your                  - W W
  move -->              B - -
                        B - B
                          |
                          |
opponent's              - - W
  move -->              W - -
                        B - B
                          |
                          |
  your                  - - W
responses -->           W - B
                        B - -
                          |
                          |
opponent's                |
  moves -->             none (we win! woohoo!)
```

What if white makes a different move?

not this...


W W W
- - -
B B B
opponent's
move -->
- W W
W - - 0
B B B
your -->
response
- W W
B - - 0
B - B
opp's
move
--> - - W
W - -
B - B
0

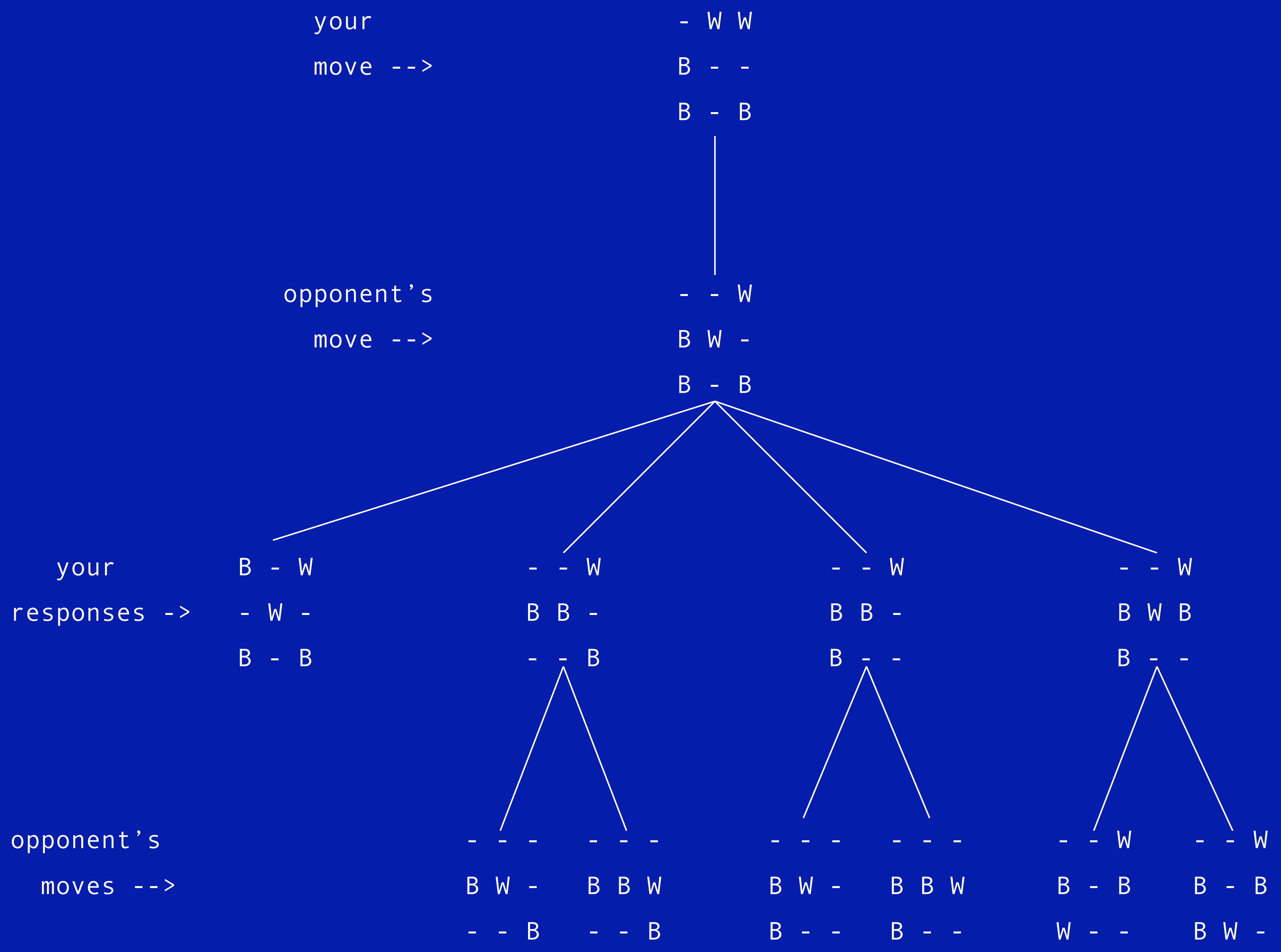
...but this

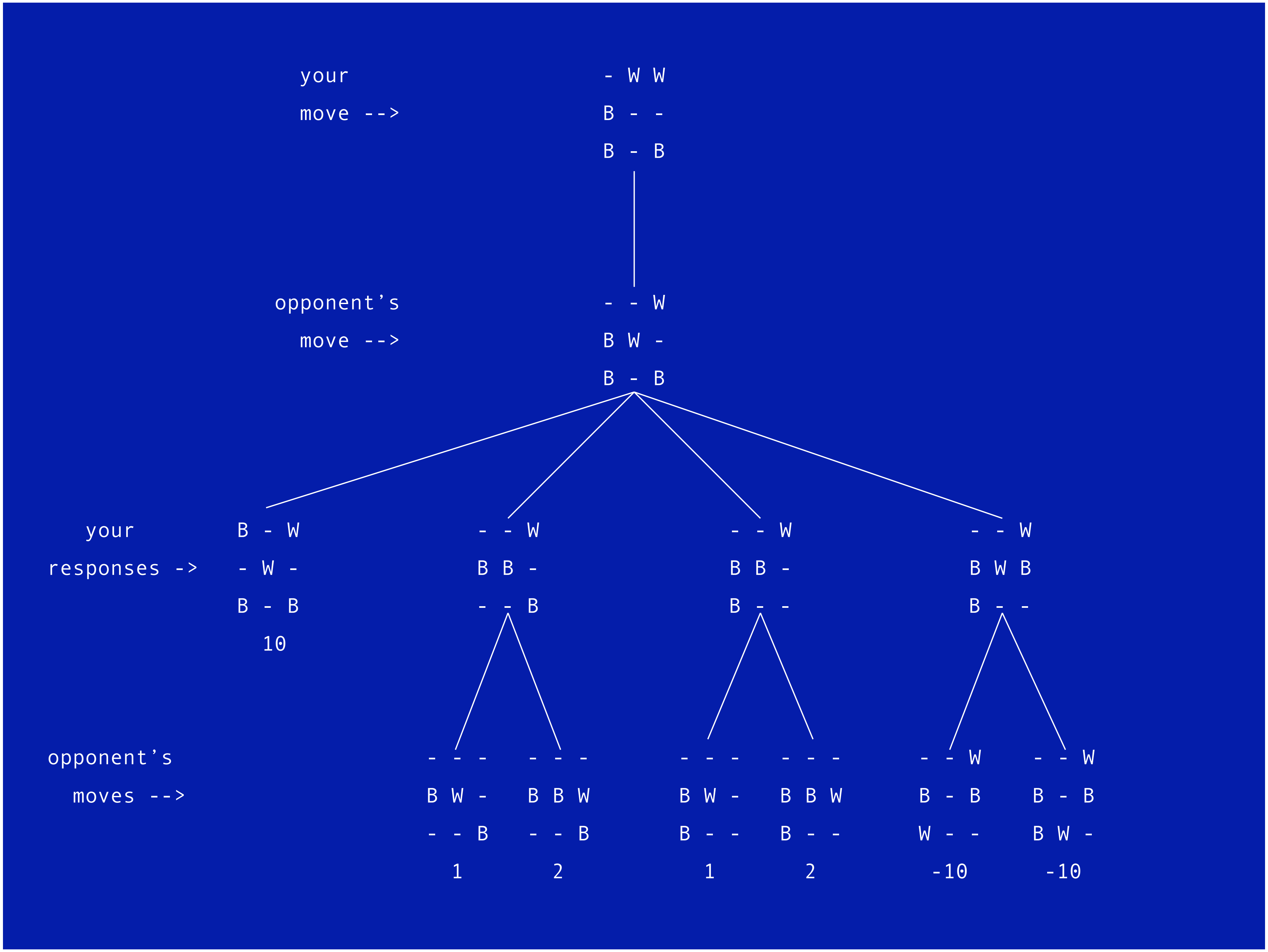

W W W
- - -
B B B
opponent's move -->
- W W
W - - 0
B B B
your --> response
- W W
B - - 0
B - B
opp's move -->
- - W
B W -
B - B
1

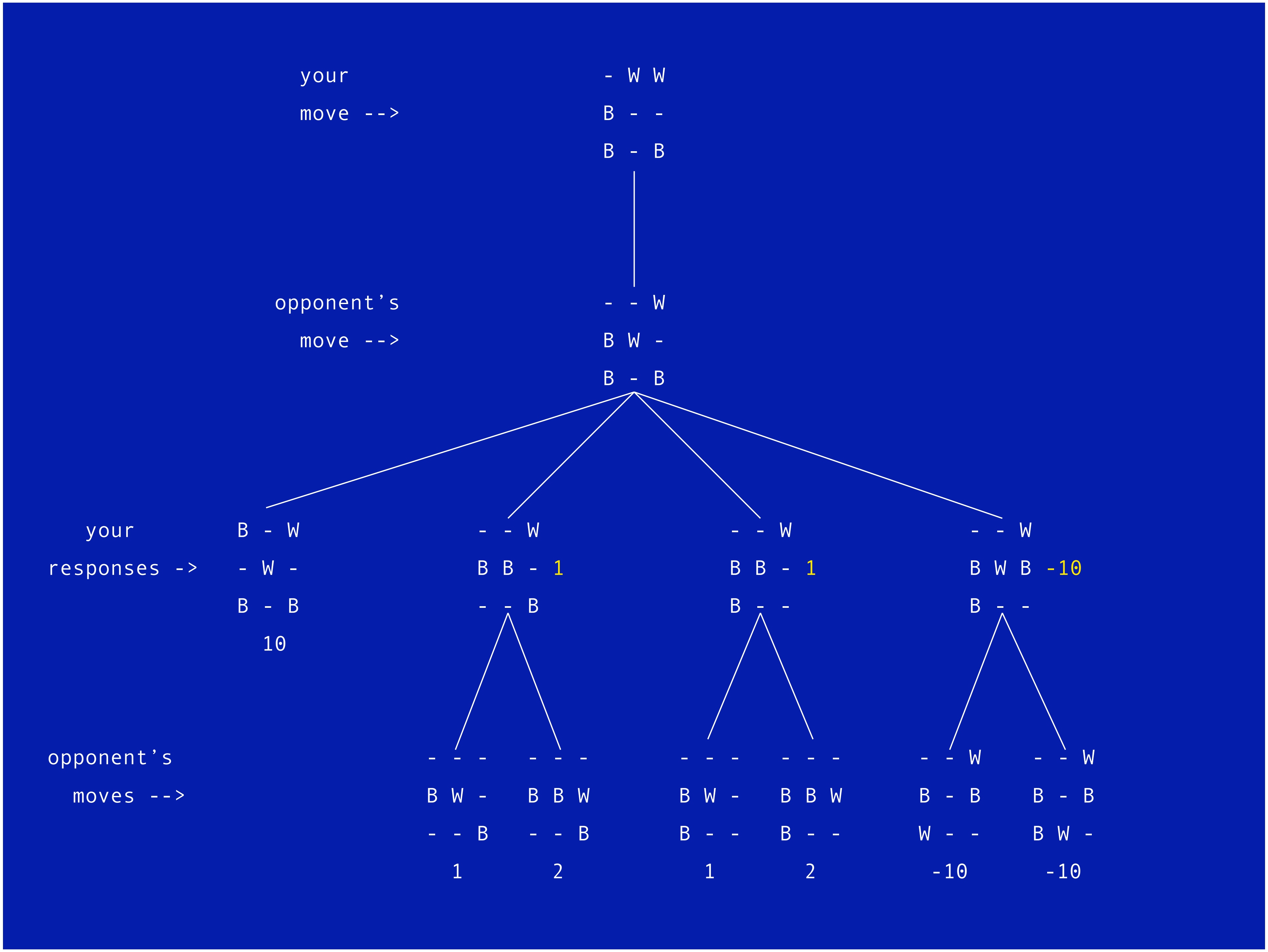
What if the white makes a different move?

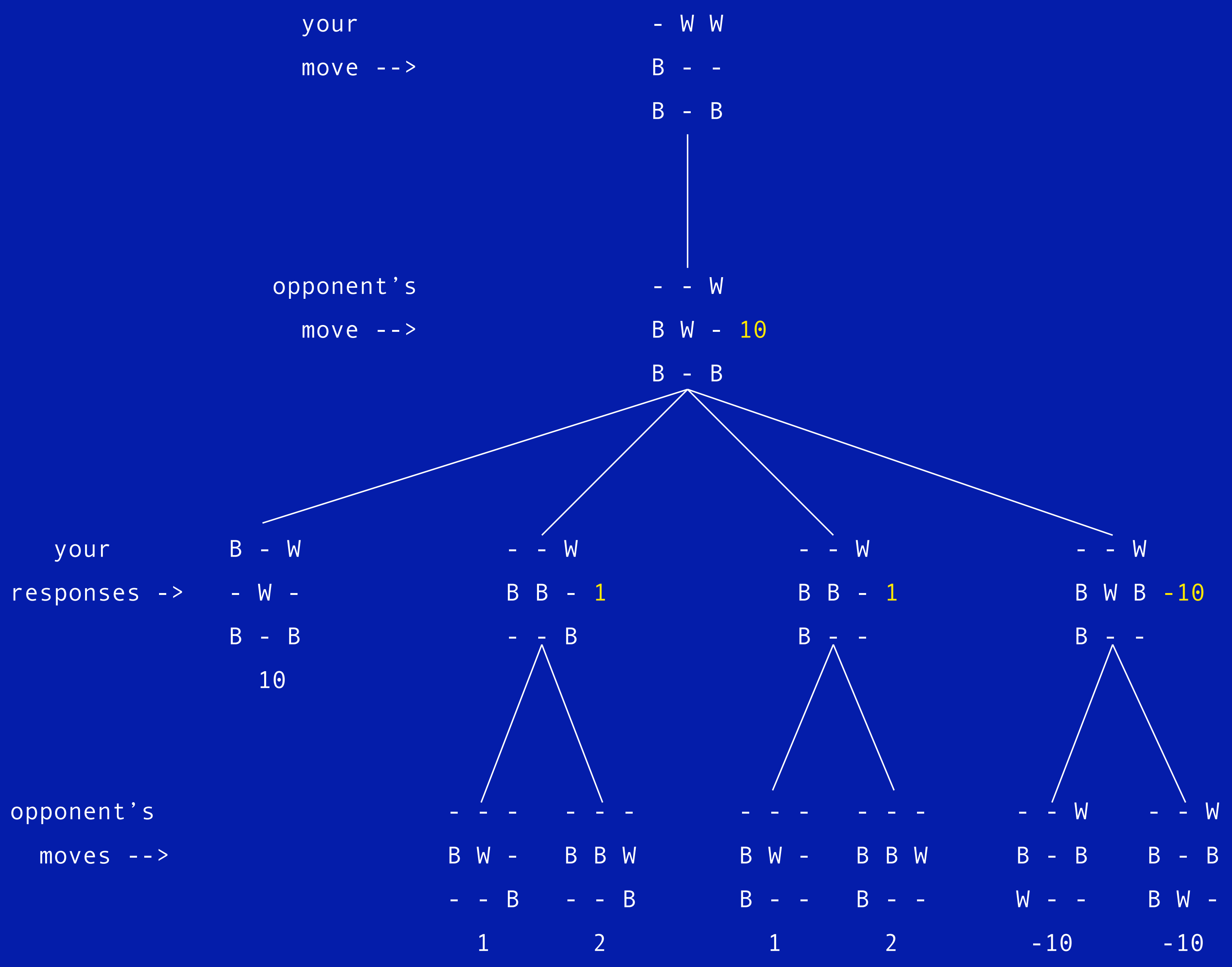
Apply this search technique again to white’s move and make your next move accordingly

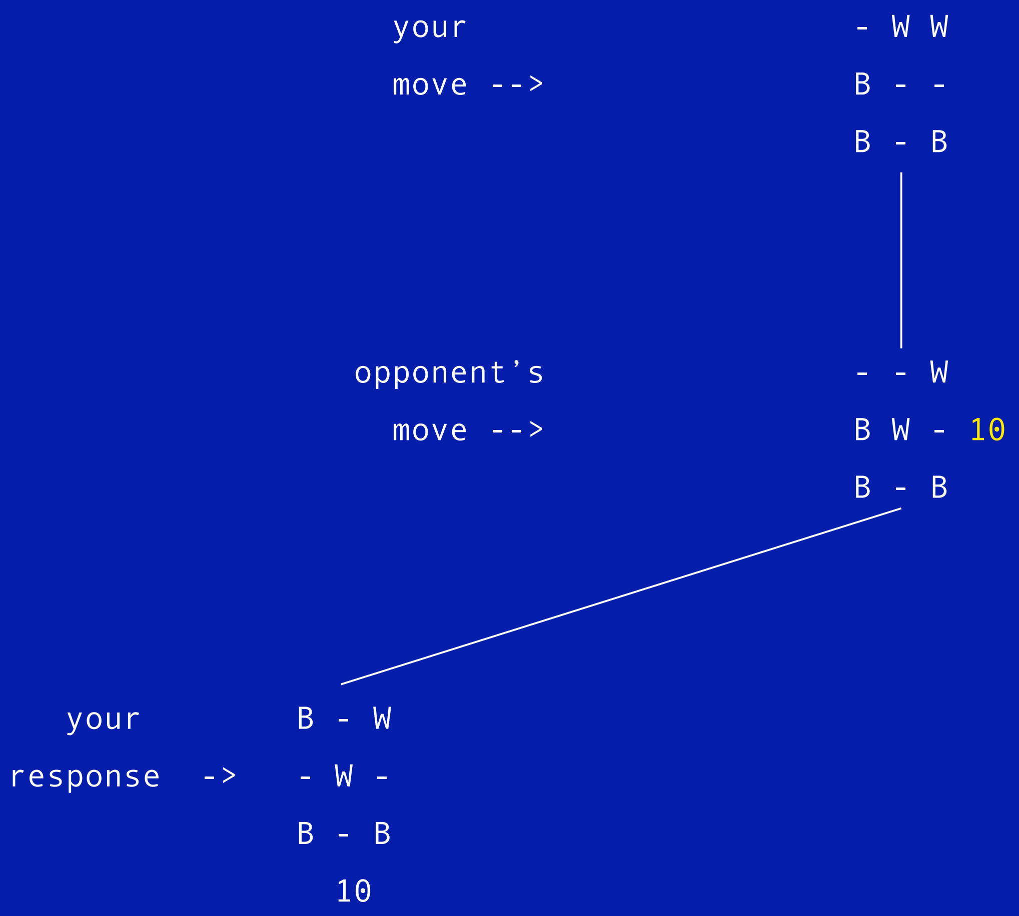
```
  your               - W W
  move -->           B - -
                     B - B
                       |
                       |
                       |
opponent's           - - W
  move -->           B W -
                     B - B
```


your move -->
- W W
B - -
B - B
opponent's move -->
- - W
B W -
B - B
your responses ->
B - W
- W -
B - B
- - W
B B -
- - B
- - W
B B -
B - -
- - W
B W B
B - -
opponent's moves -->
- - -
B W -
- - B
- - -
B B W
- - B
- - -
B W -
B - -
- - -
B B W
B - -
- - W
B - B
W - -
- - W
B - B
B W -


your move -->
- W W
B - -
B - B
opponent's move -->
- - W
B W -
B - B
your responses ->
B - W
- W -
B - B
10
- - W
B B -
- - B
- - W
B B -
B - -
- - W
B W B
B - -
opponent's moves -->
- - -
B W -
- - B
1
- - -
B B W
- - B
2
- - -
B W -
B - -
1
- - -
B B W
B - -
2
- - W
B - B
W - -
-10
- - W
B - B
B W -
-10


your move -->
- W W
B - -
B - B
opponent's move -->
- - W
B W -
B - B
your responses ->
B - W
- W -
B - B
10
- - W
B B - 1
- - B
- - W
B B - 1
B - -
- - W
B W B -10
B - -
opponent's moves -->
- - -
B W -
- - B
1
- - -
B B W
- - B
2
- - -
B W -
B - -
1
- - -
B B W
B - -
2
- - W
B - B
W - -
-10
- - W
B - B
B W -
-10


your move -->
- W W
B - -
B - B
opponent's move -->
- - W
B W - 10
B - B
your responses ->
B - W
- W -
B - B
10
- - W
B B - 1
- - B
- - W
B B - 1
B - -
- - W
B W B -10
B - -
opponent's moves -->
- - -
B W -
- - B
1
- - -
B B W
- - B
2
- - -
B W -
B - -
1
- - -
B B W
B - -
2
- - W
B - B
W - -
-10
- - W
B - B
B W -
-10


your
move -->
- W W
B - -
B - B
opponent's
move -->
- - W
B W - 10
B - B
your
response ->
B - W
- W -
B - B
10