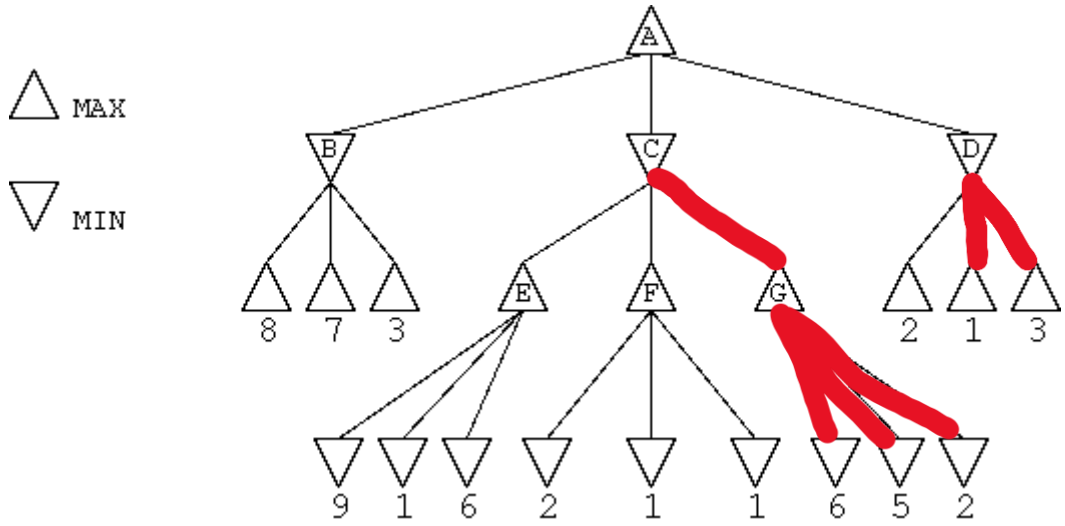
Games.01:

1.

A	B	C	D	E	F	G
3	3	2	1	9	2	6

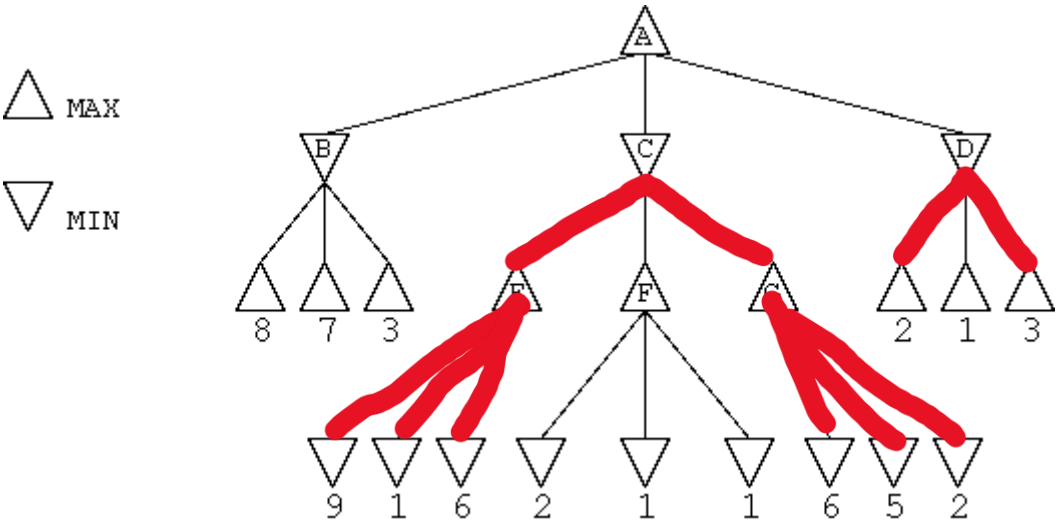
2.

Die rotmarkierten Pfade werden nicht weiter betrachtet.



A	B	C	D	E	F	G
v: 3, α: 3, β: $+inf$	v: 3, α: $-inf$, β: 3	v: 2, α: 3, β: $+inf$	v: 1, α: 3, β: 2	v: 9, α: 9, β: +inf	v: 2, α: 2, β: $+inf$	v: 6, α: $-inf$, β: +inf

3. Der Knoten kann geordnet werden, sodass mehr Zweige abgeschnitten werden.



Ordnung:

A -> B,

B -> 8 : B=8, B -> 7 : B=7, B -> 3 : B=3,

A -> C : A = 3,

C -> F,

F -> 2 : F=2, F -> 1 : F=2, F -> 1 : F=2,

C=2,

E und G fallen weg, da A=3 und C=2 und C somit nicht mehr kleiner werden kann.

A -> D: A=2,

D -> 2, D=2

Alle anderen pfade von D fallen weg da A=2 und D=2 und D somit nicht mehr größer wird.

Games.02:

Vergleich zwischen Minimax ohne Pruning und mit:

- Ohne alpha beta pruning:

```
Start Bord:
 [['_' '_' '_']
 ['_' '_' '_']
 ['_' '_' '_']]
Bester Zug für Max: (2, 2)
Anzahl besuchter Knoten: 549946

Process finished with exit code 0
```

- Mit alpha beta pruning:

```
Start Bord:
 [['_' '_' '_']
 ['_' '_' '_']
 ['_' '_' '_']]
Bester Zug für Max: (2, 2)
Anzahl besuchter Knoten: 30710

Process finished with exit code 0
```

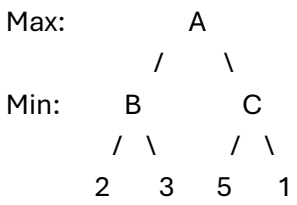
Games.03:

Vereinfachter Minimax Code:

```python
def minimax(state):  1 usage
    if terminate_test(state): return utility(state)

    val = float('-inf')
    action = None
    for (a, s) in successor(state):
        v = -minimax(s)
        if val <= v:
            val = v
            action = a
    return action
```

Beispiel Minimax:



A	B	C
2	2	1

Mit Vereinfachter Code:

Startzustand: A : val = -inf
A -> B : val = inf
B -> 2 : v = -2, B -> 3 : v = -3
B : max(-2, -3) = -2
B : val = -(-2)
A -> C : val = 2
C -> 5 : val = -5, C -> 1 : val = -1
C : max(-5, -1) = -1
C : val = -(-1)
A : max(2, 1) = 2
A : val = 2

Games.04:

Spielzustand 1:

_ | _ | _

_ | _ | _

_ | _ | _

Eval(s)	$3X_2(s)$	$X_1(s)$	$3O_2(s)$	$O_1(s)$
0-0=0	3*0=0	0	3*0=0	0

Spielzustand 2:

_ | _ | _

_ | X | _

_ | _ | _

Eval(s)	$3X_2(s)$	$X_1(s)$	$3O_2(s)$	$O_1(s)$
4-0=4	3*0=0	4	3*0=0	0

Spielzustand 3:

X | _ | _

_ | O | _

_ | _ | _

Eval(s)	$3X_2(s)$	$X_1(s)$	$3O_2(s)$	$O_1(s)$
2-3=-1	3*0=0	2	3*0=0	3

Spielzustand 4:

X | X | _

O | _ | _

_ | _ | O

Eval(s)	$3X_2(s)$	$X_1(s)$	$3O_2(s)$	$O_1(s)$
4-3=1	3*1=3	1	3*0=0	3

Spielzustand 5:

X | O | _

_ | O | X

_ | _ | _

Eval(s)	$3X_2(s)$	$X_1(s)$	$3O_2(s)$	$O_1(s)$
2-4=-2	3*0=0	2	3*1=3	1

Spielzustand 6:

X | X | X

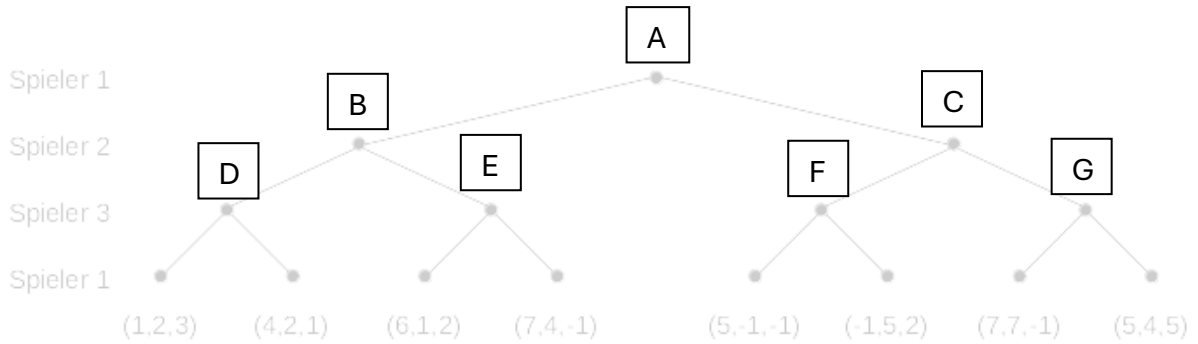
O | O | _

_ | _ | _

Eval(s)	$3X_2(s)$	$X_1(s)$	$3O_2(s)$	$O_1(s)$
1-3=-2	3*0=0	1	3*1	0

Diese Eval Funktion könnte sinnvoll sein, da er erkennt, wenn ein Spieler gute oder schlechte Gewinnchancen hat bzw. wer wen in einem Zustand dominiert. Dazu ist diese Funktion einfach zu berechne.

Games.05:



A	B	C	D	E	F	G
(1, 2, 3)	(1, 2, 3)	(-1, 5 ,2)	(1, 2, 3)	(6, 1, 2)	(-1, 5, 2)	(5, 4, 5)