Objektorientierte Programmierung

Studiengang Medieninformatik

Hans-Werner Lang

Hochschule Flensburg

Vorlesung 11
31.05.2017

Hochschule Flensburg
University of Applied Sciences

Heute:

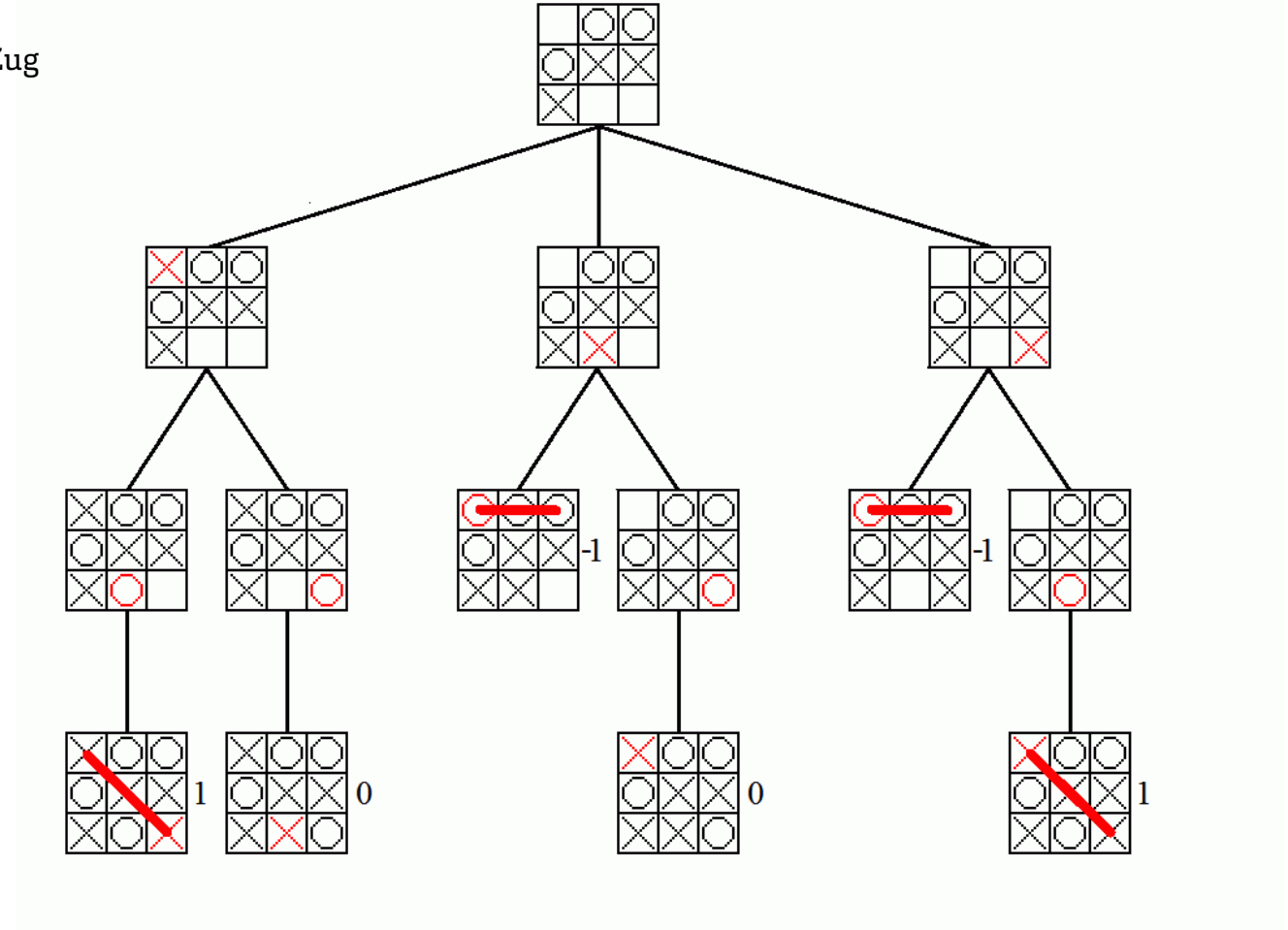
- Spielbaum-Auswertung
- Computerspieler *SuperPlayer*

Spielbaum-Auswertung - Minimax-Strategie (1)

Blätter des Baums bewerten (aus Sicht des Spielers, der in der Ausgangsstellung am Zug ist)

Hochschule Flensburg
University of Applied Sciences

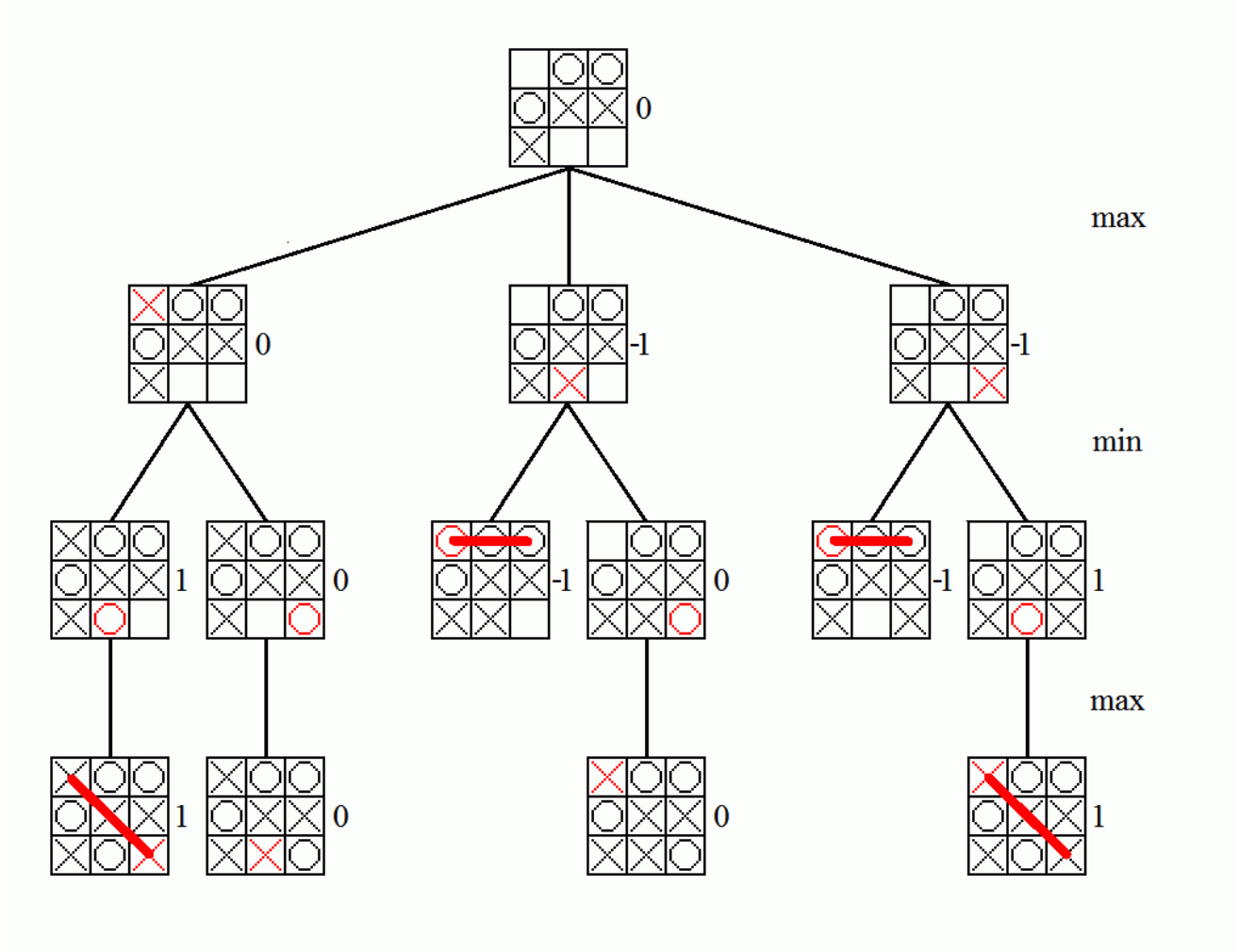
X am Zug



Spielbaum-Auswertung - Minimax-Strategie (2)

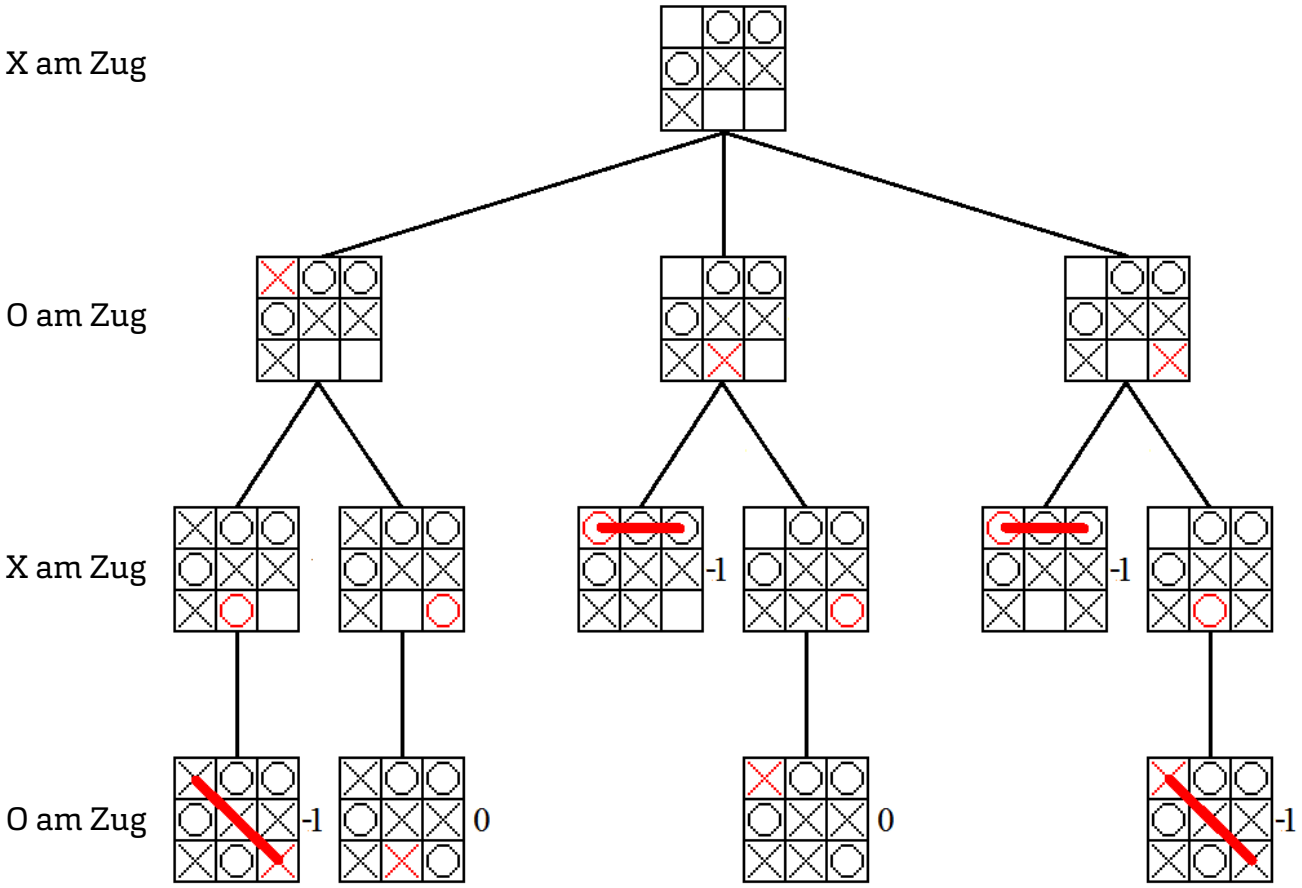
Innere Knoten bewerten (bottom-up)

Hochschule Flensburg
University of Applied Sciences



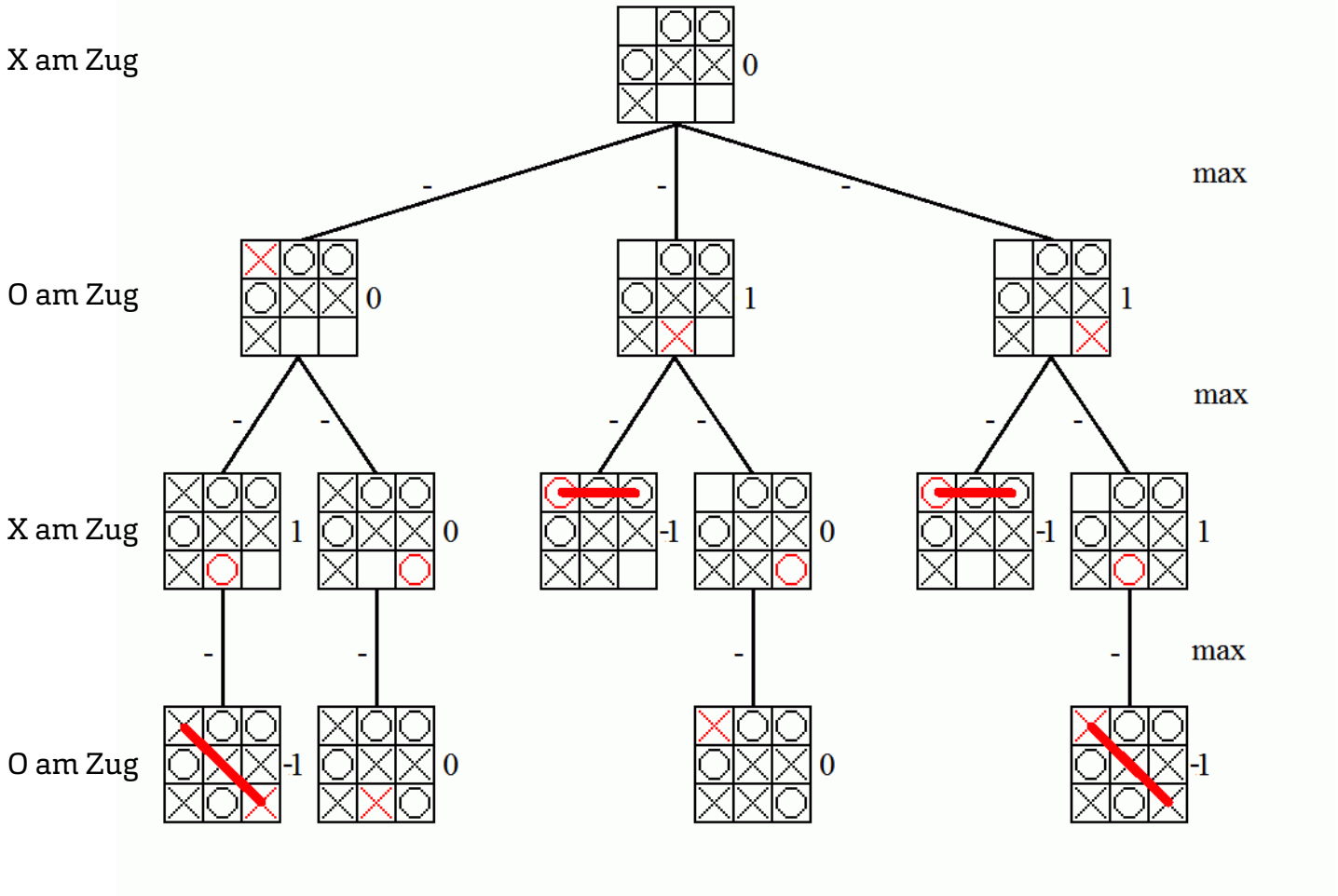
Spielbaum-Auswertung - Negamax-Strategie (1)

Blätter des Baums bewerten (aus Sicht des Spielers, der in der jeweiligen Spielstellung am Zug ist)



Spielbaum-Auswertung - Negamax-Strategie (2)

Innere Knoten bewerten (bottom-up)

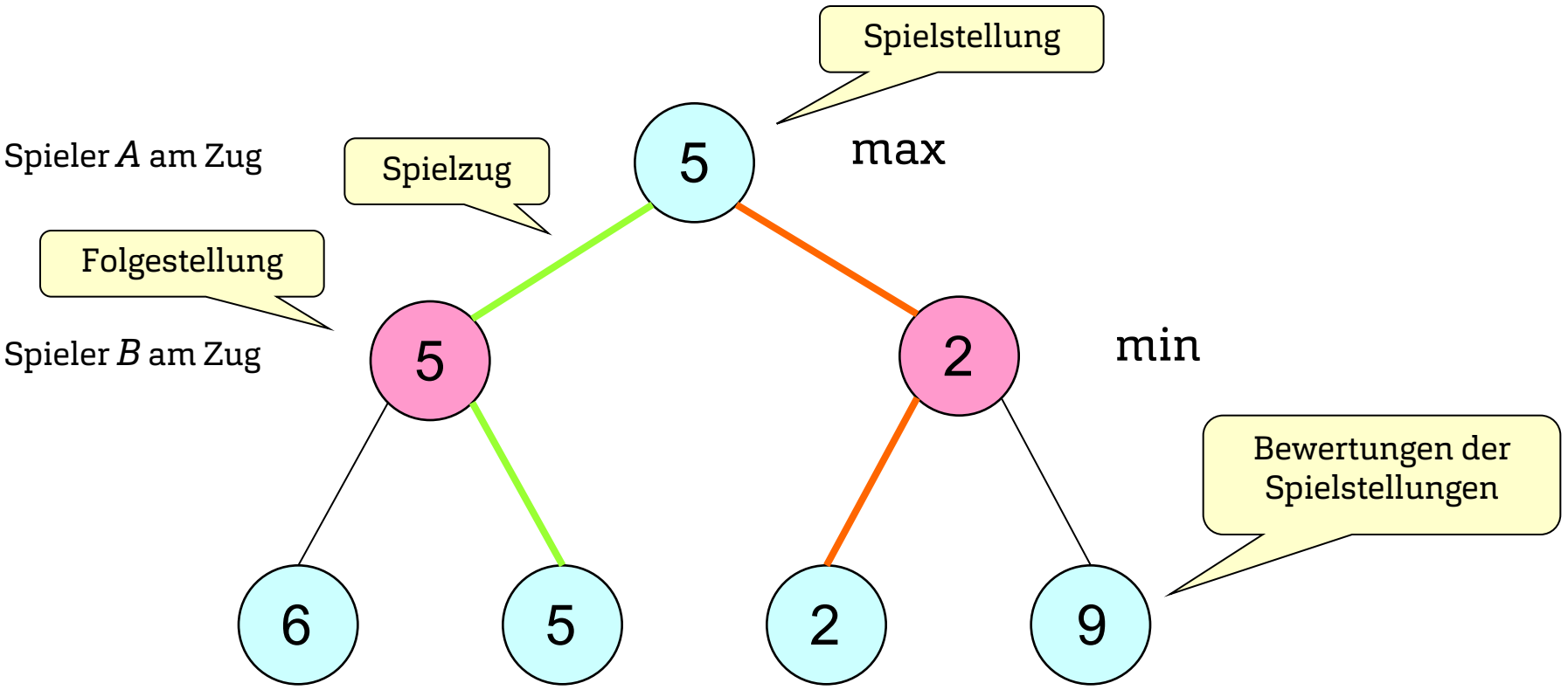


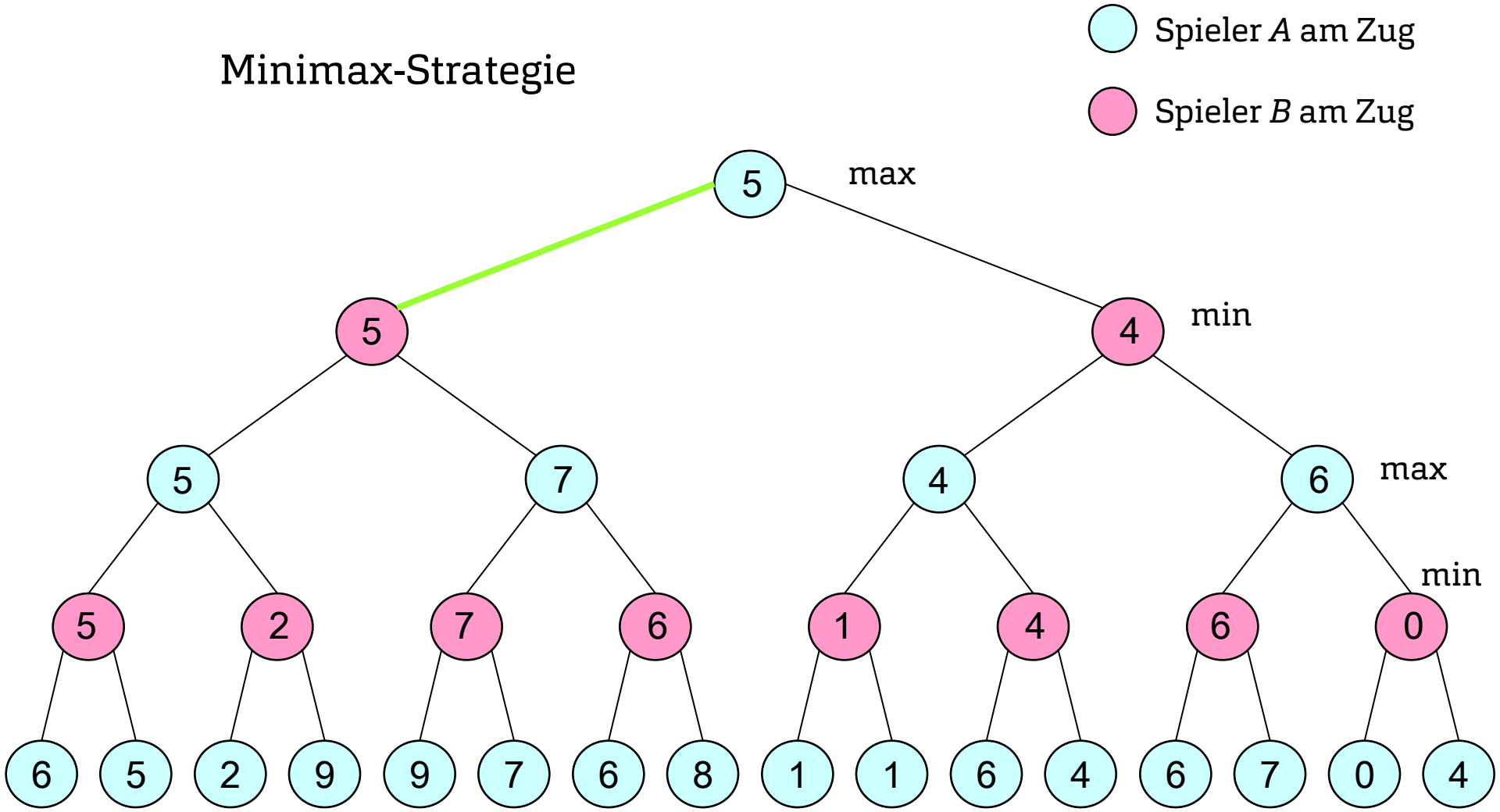
Vorausberechnung von Spielzügen

Berechnung des optimalen Spielzugs

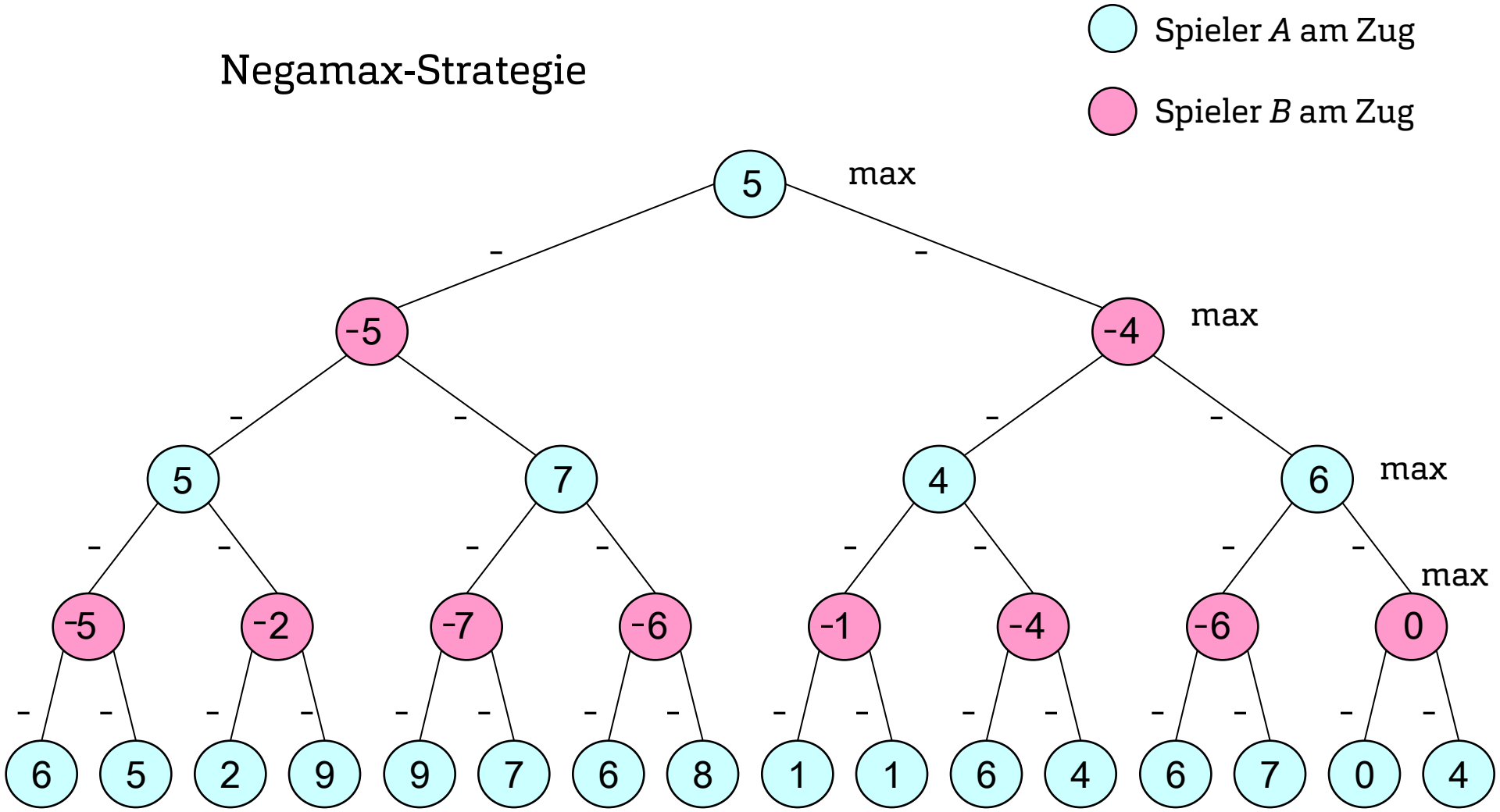
Spielbaum für 2 Züge

Hochschule Flensburg
University of Applied Sciences




Minimax-Strategie
Spieler A am Zug
Spieler B am Zug
max
min
max
min
5
5
4
5
7
4
6
5
2
7
6
1
4
6
0
6
5
2
9
9
7
6
8
1
1
6
4
6
7
0
4

Negamax-Strategie



Methode *eval* in der Klasse *SuperPlayer*

wertet rekursiv den Spielbaum aus (Negamax-Strategie);
beginnend von einer Spielstellung *s* bis zu einer Tiefe von *d*

```
private double eval(Configuration s, int d)
{
    if (d==0 || s.gameOver())
        return s.rate();
    Iterator<Move> it=new NextMoveIterator(s);
    MiniMaximizer<Move> minimax=new MiniMaximizer<Move>();
    Move move;
    Configuration t;
    while (it.hasNext())
    {
        move=it.next();
        t=s.copy();
        t.setMove(move);
        minimax.add(-eval(t, d-1));
    }
    return minimax.getMaxVal();
}
```

Methode *findMove* in der Klasse *SuperPlayer*

Hochschule Flensburg
University of Applied Sciences

wertet rekursiv den Spielbaum aus (Negamax-Strategie), beginnend von der aktuellen Spielstellung *config* bis zu einer Tiefe von *searchdepth*, und gibt den optimalen Spielzug zurück

```
public Move findMove()
{
    Configuration s=config;
    Iterator<Move> it=new NextMoveIterator(s);
    MiniMaximizer<Move> minimax=new MiniMaximizer<Move>();
    Move move;
    Configuration t;
    while (it.hasNext())
    {
        move=it.next();
        t=s.copy();
        t.setMove(move);
        minimax.add(-eval(t, searchdepth-1), move);
    }
    return minimax.getMaxObj();
}
```

Bemerkungen zur Implementierung

Hochschule Flensburg
University of Applied Sciences

- Wir benutzen in der Klasse *Configuration* statt der Methode *applyMove* eine Methode *setMove*, die dasselbe tut wie *applyMove*, jedoch ohne die Observer zu benachrichtigen.

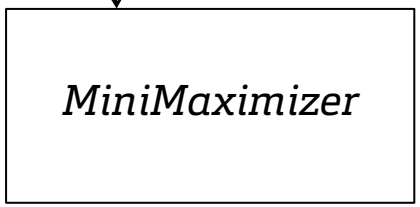
- Statt des Iterators *NextMoveIterator* benutzen wir unseren *ModIntIterator*, um die Spalten von 0 bis 6 beginnend in einer zufälligen Spalte zu durchlaufen, und erzeugen den Spielzug im Schleifenkörper der While-Schleife und prüfen dort, ob er zulässig ist (Spalte ist noch nicht voll).

- Wir benutzen das Software-Muster *MiniMaximizer*, um den maximalen Wert bzw. das zugehörige Objekt einer Folge von Paaren (*Wert, Objekt*) zu bestimmen.

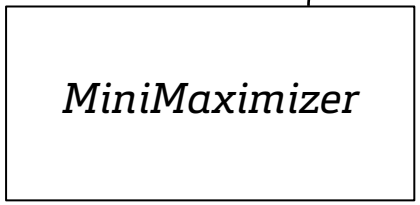
Software-Muster *MiniMaximizer*

Hochschule Flensburg
University of Applied Sciences

```
add(1, "ene");
add(3, "mene");
add(-2,"muh");
```



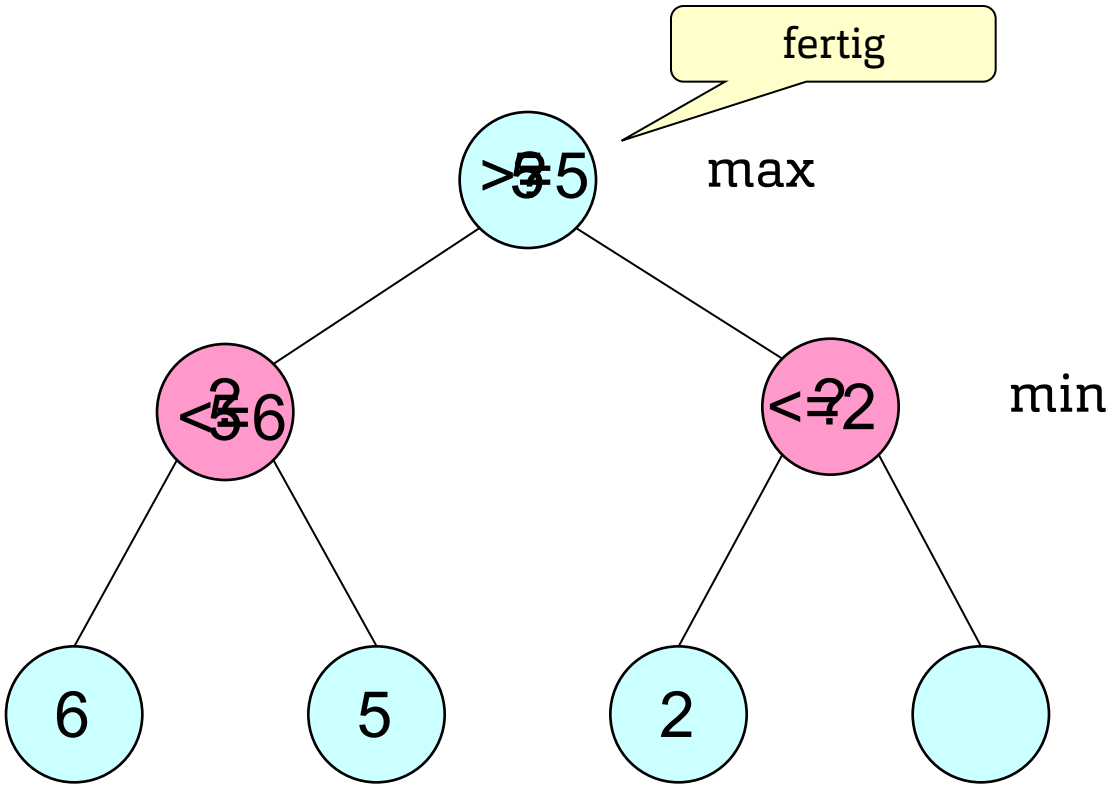

```
getMaxVal(); // 3
getMaxObj(); // "mene"
getMinVal(); // -2
getMinObj(); // "muh"
```



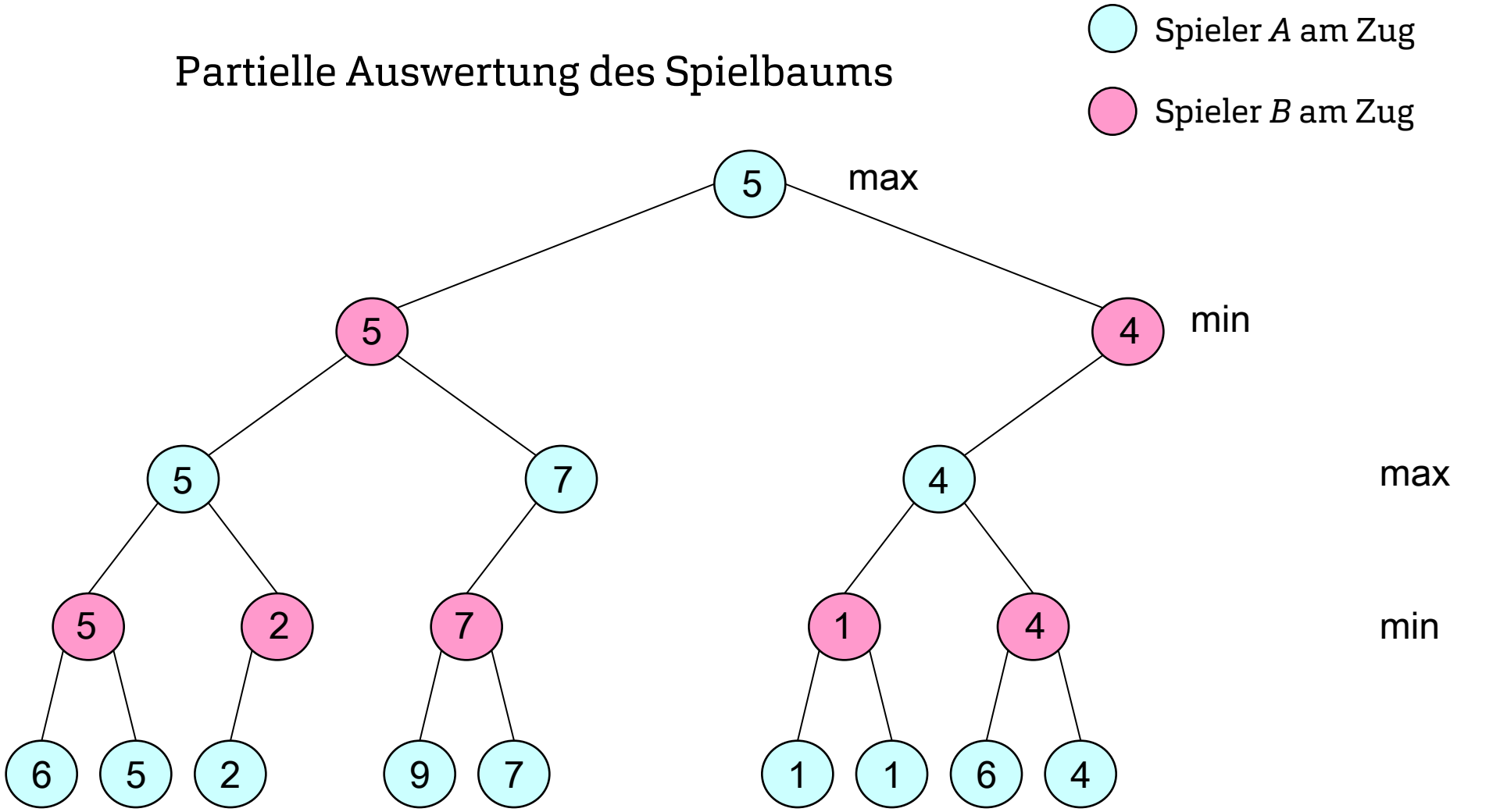
Werden nur die Zahlenwerte benötigt, ist auch z.B. add(3) möglich. Die Methoden *getMaxObj* und *getMinObj* liefern dann *null*.

Partielle Auswertung des Spielbaums

Hochschule Flensburg
University of Applied Sciences



Partielle Auswertung des Spielbaums



Methode *eval* für partielle Auswertung

Hochschule Flensburg
University of Applied Sciences

```
private double eval(Configuration s, int d, double w)
{
    if (d==0 || s.gameOver())
        return s.rate();
    Iterator<Move> it=new NextMoveIterator(s);
    MiniMaximizer<Move> minimax=new MiniMaximizer<Move>();
    Move move;
    Configuration t;
    while (it.hasNext() && minimax.getMaxVal()<w)
    {
        move=it.next();
        t=s.copy();
        t.setMove(move);
        minimax.add(-eval(t, d-1, -minimax.getMaxVal()));
    }
    return minimax.getMaxVal();
}
```

Klasse *SuperPlayer*

Hochschule Flensburg
University of Applied Sciences

```
public class SuperPlayer extends SimplePlayer
{
    private int searchdepth=5;

    @Override
    public Move findMove()
    {
        ...
    }

    private double eval(Configuration s, int d)
    {
        ....
    }

}
```

Beim nächsten Mal:

- Thread-Programmierung (wichtig für Animation von Grafik)