Universität des Saarlandes
FR 6.2 Informatik

Prof. Dr. Kurt Mehlhorn, Dr. Konstantinos Panagiotou **WiSe 2011/2012**

Übungen zu Computational Thinking

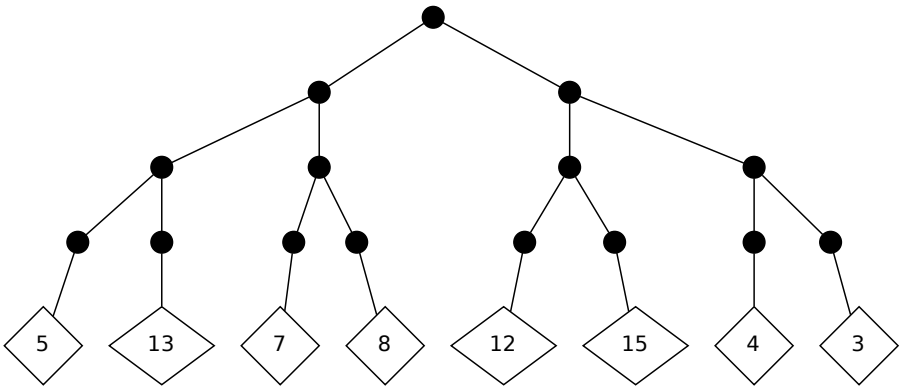
http://www.mpi-inf.mpg.de/departments/d1/teaching/ws11/ct/

Bonus-Blatt 4

Regeln: Bis zum Semesterende müssen mindestens 42% der maximal erreichbaren Punkte aller Übungszettel erworben werden.
Programmcode ist elektronisch per E-Mail abzugeben. Zusätzlich müssen die Ausgaben einer exemplarischen Programmausführung mitgeliefert werden.

Punkte können Sie erhalten, wenn Sie die Aufgaben bis zum 10.2.12 bearbeiten.

Aufgabe 1 (*20 Punkte*) Führen Sie den Min-Max Algorithmus und den α-β Algorithmus auf dem folgenden Spielbaum aus:



Aufgabe 2 (*10 Punkte*) Um wieviel schneller als der Min-Max Algorithmus ist die α-β Suche im besten und im schlechtesten Fall? Begründen Sie ihre Antwort.

Aufgabe 3 (*10 Punkte*) Schreiben Sie ein Programm, um das 15-Puzzle zu lösen. Es handelt sich um ein 4×4 Feld, auf dem sich 15 nummerierte Plättchen und eine Lücke befinden. Ziel ist es, die Plättchen durch verschieben zu sortieren.

Machen Sie ihr Programm effizienter, indem sie bereits untersuchte Stellungen kein zweites Mal untersuchen. Für jede lösbare Ausgangsstellung soll ihr Programm die kürzeste Schiebesequenz ausgeben, die zur Lösung führt.

Aufgabe 4 (*20 Punkte*) Schreiben Sie einen optimal spielenden Computergegner für TicTacToe.

Aufgabe 5 (*30 Punkte*) Die meisten Spiele haben zu viele Stellungen um optimal zu spielen. Statt den gesamten Spielbaum zu durchsuchen, muss man die Suche bei einer gewissen Tiefe abbrechen und die Stellung heuristisch bewerten. Eine sehr einfache Heuristik erhält man, wenn man von der zu bewertenden Stellung ausgehend das Spiel mit zufälligen Zügen zuende spielt und überprüft, wer gewinnt. Nachdem man ausreichend viele solcher zufälligen Spiele gespielt hat, kann man die Gewinnwahrscheinlichkeit für den Knoten abschätzen. Wenn man mehrere Heuristiken einsetzt, muss man sie geeignet verbinden, zum Beispiel durch eine gewichtete Summe mit geschickt gewählten Gewichten.
Schreiben Sie einen möglichst guten Computergegner für Vier Gewinnt auf einem 7×6 Feld.
Ihr Programm muss eine Klasse mit Namen `Spiel` enthalten. Im Konstruktor nimmt die Klasse eine Zahl, die entweder 0 oder 1 ist und bestimmt für welchen Spieler die Klasse spielen soll. Außerdem muss die Klasse zwei Funktionen zur Verfügung stellen,

- `machZug(spalte, spieler)` und
- `getZug()`.

Die Funktion `machZug` nimmt die Spalte und den Spieler und verändert die derzeitige Stellung. Die Funktion `getZug()` liefert den von ihrer KI bevorzugten Zug, ohne ihn auszuführen. Die Funktion, die ihre KI gegen einen Mitspieler spielen lässt könnte also etwa so aussehen:

```
def spiel():
 spieler = [Spiel(0), gegner.Spiel(1)]
 stellung = Stellung()
 amZug = 0
 while not stellung.gewonnen():
   x = spieler[amZug].getZug()
   spieler[0].machZug(x,amZug)
   spieler[1].machZug(x,amZug)
   stellung.machZug(x,amZug)
```

```
   amZug = (amZug + 1) % 2
 print stellung.gewinner()
```

und die Klasse, die einen menschlichen Spieler teilnehmen lässt etwa so:

```
class Spiel(object):
  def __init__(self,i):
    print "Du bist Spieler ", i

  def machZug(x,s):
    print "Spieler ",s,"setzt in",x

  def getZug():
    return int(raw_input("In welche Spalte setzt du?"))
```

Die Klasse zu erstellen und den nächsten Zug zu berechnen sollte nicht länger als 10 Sekunden dauern. Wenn diese Aufgabe von nicht weniger als 4 Personen brauchbar bearbeitet wird, erhält der Autor der stärksten KI 40 zusätzliche Punkte.