

## Allgemeine Hinweise:

- Die **Deadline** zur Abgabe der Hausaufgaben ist am **Donnerstag, den 27.11.2025, um 14 Uhr**.
- Der **Workflow** sieht wie folgt aus. Die Abgabe der Hausaufgaben erfolgt **im Moodle-Lernraum** und kann nur in **Zweiergruppen** stattfinden. Dabei müssen die Abgabepartner\*innen **dasselbe Tutorium** besuchen. Nutzen Sie ggf. das entsprechende **Forum** im Moodle-Lernraum, um eine\*n Abgabepartner\*in zu finden. Es darf **nur ein\*e** Abgabepartner\*in die Abgabe hochladen. Diese\*r muss sowohl die **Lösung** als auch den **Quellcode** der Programmieraufgaben hochladen. Die Bepunktung wird dann von uns für **beide** Abgabepartner\*innen **separat** im Lernraum eingetragen. Die Feedbackdatei ist jedoch nur dort sichtbar, wo die Abgabe hochgeladen wurde und muss innerhalb des Abgabepaars **weitergeleitet** werden.
- Die **Lösung** muss als PDF-Datei hochgeladen werden. Damit die Punkte beiden Abgabepartner\*innen zugeordnet werden können, müssen **oben** auf der **ersten Seite** Ihrer Lösung die **Namen**, die **Matrikelnummern** sowie die **Nummer des Tutoriums** von **beiden** Abgabepartner\*innen angegeben sein.
- Der **Quellcode** der Programmieraufgaben muss als **.zip**-Datei hochgeladen werden und **zusätzlich** in der PDF-Datei mit Ihrer Lösung enthalten sein, sodass unsere Hiwis ihn mit Feedback versehen können. Auf diesem Blatt muss Ihre Codeabgabe Ihren vollständigen Java-Code in Form von **.java**-Dateien enthalten. Aus dem Lernraum heruntergeladene Klassen dürfen nicht mit abgegeben werden. Stellen Sie sicher, dass Ihr Programm von **javac** in der **Version 25** akzeptiert wird. Generell sollten alle Programme für alle Eingaben terminieren, solange in der Spezifikation (bzw. der Aufgabenstellung) nicht explizit etwas anderes verlangt wird!
- Einige Hausaufgaben müssen im Spiel **Codescape** gelöst werden. Klicken Sie dazu im Lernraum rechts im Block "Codescape" auf den angegebenen Link. Diese Aufgaben werden getrennt von den anderen Hausaufgaben gewertet.
- Aufgaben, die mit einem \* markiert sind, sind Sonderaufgaben mit erhöhtem Schwierigkeitsgrad. Die Bearbeitung dieser Sonderaufgaben ist nicht erforderlich, um die Klausurzulassung zu erreichen. Ihnen werden jedoch die in solchen Aufgaben erreichten Punkte ganz normal gutgeschrieben.

**Übungsaufgabe 1 (Überblickswissen):**

- a) Was ist ein StackOverflowError in Java und was ist der häufigste Grund dafür?
- b) Was ist der Vorteil von Schleifen gegenüber Rekursion? Wann ist Rekursion sinnvoller?
- c) Was versteht man unter einem Rekursionsanker/Basisfall? Warum wird ein solcher benötigt?

## Übungsaufgabe 2 (Programmieren mit Rekursion):

- a) Die Fibonacci-Zahlen sind wie folgt definiert:  $F_0 = 0$ ,  $F_1 = 1$  und  $F_n = F_{n-1} + F_{n-2}$  für  $n > 1$ . Schreiben Sie eine Klasse `Fibonacci`, welche die zwei statischen Methoden `calculateIterative` und `calculateRecursive` enthält. Diese Methoden erhalten jeweils einen `int`-Parameter  $n$  und geben die  $n$ -te Fibonacci-Zahl zurück. Dabei soll die erstgenannte Methode keine Rekursion und die zweitgenannte Methode keine Schleifen (aber Rekursion) benutzen. Die rekursive Methode soll dabei nicht mehr als  $n$  rekursive Aufrufe brauchen, um die  $n$ -te Fibonacci-Zahl zu berechnen.

- b) Schreiben Sie eine Klasse `MonoArray`, welche eine statische Methode

```
findKey(int[] sorted_array, int key)
```

enthält. Die Methode soll die Position des `int`-Wertes `key` im Array `sorted_array` als `int`-Wert zurückgeben. Sie dürfen davon ausgehen, dass das Array `sorted_array` aufsteigend sortiert ist und den `int`-Wert `key` genau einmal enthält. Die rekursive Methode soll dabei nicht mehr als  $n + 1$  rekursive Aufrufe brauchen, um ein Array der Länge  $2^n$  zu durchsuchen.

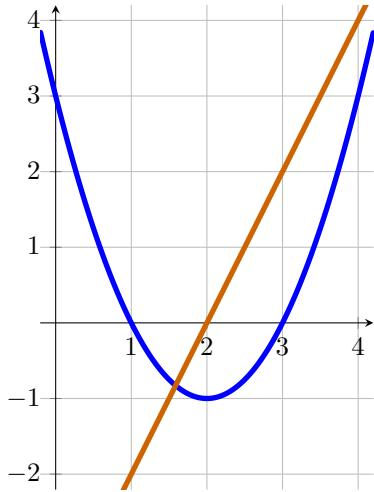


Abbildung 1: Plot der quadratischen Funktion  $x^2 - 4x + 3$  in Blau und deren Ableitung  $2x - 4$  in Orange. Die blaue Funktion hat Nullstellen bei  $x = 1$  und  $x = 3$ . Sie hat eine Extremstelle bei  $x = 2$ . Dort hat die orangene Funktion eine Nullstelle.

### Hausaufgabe 3 (Programmieren mit Rekursion): $(8 + 4 + 10 + 8 + 8^* = 30 + 8^*$ Punkte)

In dieser Aufgabe sollen Sie ein Programm schreiben, welches rekursiv die  $x$ -Koordinate von Nullstellen (engl. root) und Extremwerten (engl. extremum) quadratischer Funktionen der Form  $ax^2 + bx + c$  bestimmt. Ein Beispiel für solch eine Funktion und deren Ableitung sehen Sie in Abbildung 1. Die Idee ist, dass diese Werte nicht direkt bestimmt werden sollen (etwa via  $pq$ -Formel). Stattdessen soll binäre Suche verwendet werden, um die Werte auf eine Abweichung von  $\epsilon$  genau zu bestimmen. Hat die Funktion beispielsweise eine Nullstelle bei  $x = 1$  und es ist  $\epsilon = 0.5$ , dann sind auch 0.5, 0.73 und 1.5 gültige Rückgabewerte des Programms (alle Werte im Bereich  $1 \pm 0.5$ ).

Ein Beispiel für solch ein Vorgehen mit binärer Suche sehen Sie auch in Vorlesung 12 vom 17. November 2025, Folie 6 aus Abschnitt II.3.1. Dort wird mithilfe des folgenden Algorithmus die Wurzel einer Zahl  $x$  auf ein  $\epsilon$  genau angenähert:

```
public static float sqrt (float uG, float oG, float x) {
    float m, epsilon = 1e-3f;
    m = (uG + oG) / 2;
    if (oG - uG <= epsilon) return m;
    else if (m * m > x)      return sqrt (uG, m, x);
    else                      return sqrt (m, oG, x);
}
```

Dazu wird eine untere Grenze  $uG$  und eine obere Grenze  $oG$  solange angepasst, bis die Differenz  $oG - uG$  kleiner gleich  $\epsilon$  ist. Ist beispielsweise der momentane Rückgabewert  $m$  größer als die Wurzel ( $m*m > x$ ), so wird die obere Grenze im rekursiven Aufruf auf  $m$  gesetzt, da nun klar ist, dass sich die richtige Wurzel zwischen  $uG$  und  $m$  befinden muss. Auf diese Weise wird der Abstand  $oG - uG$  bei jedem rekursiven Aufruf halbiert.

Für die nachfolgenden Programmieraufgaben dürfen Sie stets eigene Hilfsmethoden implementieren und Programmteile aus vorherigen Teilaufgaben nutzen, selbst wenn Sie diese nicht implementiert haben. Sie dürfen jedoch keine vordefinierten Methoden verwenden, sofern dies nicht explizit erlaubt ist. Berücksichtigen Sie die Prinzipien der Datenkapselung.

- Geben Sie für den Aufruf `sqrt(0, 4, 4)` der obigen Methode `sqrt` aus der Vorlesung die Werte von  $m$ ,  $uG$ ,  $oG$  bei Aufruf der Methode und in den ersten drei rekursiven Aufrufen an (z.B. als Tabelle mit einer Zeile pro Aufruf (4 Zeilen) und einer Spalte für jede Variable). Erklären Sie was hier passiert; welches Problem tritt auf? Lässt sich das Problem beheben ohne das Programm zu sehr abzuändern?
- Erstellen Sie eine Klasse `SquareFun` mit Attributen `a`, `b`, `c` vom Typ `double`, welche eine quadratische

Funktion  $ax^2 + bx + c$  repräsentieren soll. Implementieren Sie in der Klasse einen Konstruktor, der die Attribute auf die Werte der formalen Parameter setzt.

- c) Implementieren Sie in der Klasse **SquareFun** eine nicht-statische Methode **findExtremum** mit zwei Argumenten **l** und **r** des Typs **double**. Diese Methode soll die  $x$ -Koordinate des Extremwerts von  $ax^2 + bx + c$  annähern und als Ergebnis des Typs **double** zurückgeben. Nutzen Sie dazu die Genauigkeit **epsilon = 1e-15**, wie oben beschrieben. Verwenden Sie binäre Suche. Wir gehen davon aus, dass  $l \leq r$  gilt, dass die Funktion  $ax^2 + bx + c$  eine Extremstelle hat, und dass die Extremstelle im Intervall  $[l, r]$  zu finden ist. Unter diesen Voraussetzungen lässt sich Folgendes beobachten: Sei  $m = \frac{l+r}{2}$  und  $d_m = 2am + b$  der Wert der Ableitung von  $ax^2 + bx + c$  bei  $x = m$ . Falls  $d_m = 0$  ist, befindet sich die Extremstelle bei  $m$ . Sonst ist die Extremstelle links von  $m$ , wenn die Vorzeichen von  $a$  und  $d_m$  übereinstimmen, und rechts von  $m$ , falls die Vorzeichen nicht übereinstimmen.

Ist beispielsweise  $a > 0$  wie in Abbildung 1 (dort gilt  $a = 1$ ), dann ist die Extremstelle

- bei  $m$ , wenn  $d_m = 0$  ist (z.B. im Fall  $m = 2$  in Abbildung 1),
- links von  $m$ , wenn  $d_m > 0$  ist (z.B. im Fall  $m = 2.5$  in Abbildung 1) und
- rechts von  $m$ , wenn  $d_m < 0$  ist (z.B. im Fall  $m = 1.5$  in Abbildung 1).

- d) Implementieren Sie in der Klasse **SquareFun** eine nicht-statische Methode **getRoots** mit zwei Argumenten **l** und **r** des Typs **double**. Diese Methode soll die  $x$ -Koordinaten beider Nullstellen von  $ax^2 + bx + c$  annähern und als **double[]** zurückgeben. Nutzen Sie dazu die Genauigkeit **epsilon = 1e-15**, wie oben beschrieben. Verwenden Sie binäre Suche. Bestimmen Sie die Nullstellen nicht auf andere Weise, etwa via *pq*-Formel. Sie dürfen annehmen, dass  $l \leq r$  gilt, dass die Funktion  $ax^2 + bx + c$  zwei verschiedene reelle Nullstellen besitzt, und dass diese im Intervall  $[l, r]$  zu finden sind. Nutzen Sie folgende Beobachtungen, welche in diesem Fall gelten:

- Sei  $x_{\text{EXTREM}}$  die  $x$ -Koordinate der Extremstelle von  $ax^2 + bx + c$  (welche in diesem Fall auch existieren muss) und  $x_1, x_2$  die  $x$ -Koordinaten der beiden Nullstellen mit  $x_1 < x_2$ . Dann ist  $x_1 < x_{\text{EXTREM}} < x_2$ .
- Befindet sich in  $[u, o]$  mit  $u < o$  genau eine Nullstelle von  $ax^2 + bx + c$ , ist  $m = \frac{u+o}{2}$  die Position in der Mitte von  $[u, o]$ , ist  $d_m = 2am + b$  der Wert der Ableitung von  $ax^2 + bx + c$  bei  $x = m$ , und  $f(m) = am^2 + bm + c$  der Wert von  $ax^2 + bx + c$  bei  $x = m$ , dann ist diese Nullstelle
  - bei  $m$ , wenn  $f(m) = 0$  ist,
  - links von  $m$ , falls die Vorzeichen von  $d_m$  und  $f(m)$  übereinstimmen, und
  - rechts von  $m$ , falls die Vorzeichen nicht übereinstimmen.

- e) Bonusaufgabe: Erstellen Sie eine Klasse **PolyFun**, die ein Polynom  $c_n \cdot x^n + c_{n-1} \cdot x^{n-1} + \dots + c_1 \cdot x + c_0$  vom Grad  $n$  repräsentiert. Implementieren Sie in dieser Klasse eine nicht-statische Methode **getRoots(double l, double r)**, welche die  $x$ -Koordinaten aller Nullstellen annähern und als **double[]** zurückgeben soll. Verallgemeinern Sie dazu Ihre Erkenntnisse aus den vorherigen Teilaufgaben. Sie dürfen annehmen, dass  $l \leq r$  gilt, dass die Funktion  $c_n \cdot x^n + c_{n-1} \cdot x^{n-1} + \dots + c_1 \cdot x + c_0$  insgesamt  $n$  paarweise verschiedene reelle Nullstellen besitzt, und dass alle davon im Intervall  $[l, r]$  zu finden sind. Die vordefinierte Methode **Math.pow(a, b)** darf verwendet werden. Diese gibt den Wert von  $a^b$  zurück.

## Übungsaufgabe 4 (Programmieren mit rekursiven Datenstrukturen):

In dieser Aufgabe sollen eine Datenstruktur für boolesche Binärbäume sowie einige rekursive Algorithmen darauf implementiert werden.

Ein boolescher Binärbaum repräsentiert in dieser Aufgabe eine boolesche Formel, die aus Variablen, den konstanten Wahrheitswerten `true` und `false`, dem binären Konjunktionsoperator  $\wedge$  und dem unären Negationsoperator  $\neg$  aufgebaut ist. Der Disjunktionsoperator  $\vee$  soll durch die anzulegende Datenstruktur explizit *nicht* modelliert werden. Dies stellt aber keine echte Einschränkung dar, da alle booleschen Funktionen bereits über der Operatorenmenge  $\{\wedge, \neg\}$  ausgedrückt werden können; diese ist *funktional vollständig*.

Jeder Knoten kann null, ein oder zwei Kinder haben. Diese Kinder sind wiederum boolesche Binärbäume. Außerdem kann er mit einer Variable vom Typ `String` markiert sein. Was ein bestimmter Baumknoten repräsentiert, wird implizit über die Anzahl der Kinder ausgedrückt, die nicht den Wert `null` haben:

- Hat ein Knoten keine solchen Kinder, so steht er für seine Variable. Die beiden Kinder sind dann `null`. Die Variable darf nicht der leere String `" "` sein.
- Hat ein Knoten genau ein nicht-`null`-Kind, so steht er für die Negation der Formel, die durch sein Kind repräsentiert wird. Hierbei soll durch Ihre Implementierung sichergestellt werden, dass ein Negationsknoten stets sein erstes Kind nutzt. Das erste Kind ist dann nicht `null`, das zweite Kind ist `null` und die Variable soll auf den leeren String `" "` gesetzt sein.
- Hat ein Knoten zwei nicht-`null`-Kinder, so repräsentiert er die Konjunktion der beiden Formeln, die durch seine beiden Kinder repräsentiert werden. Wiederum soll die Variable der leere String `" "` sein.
- Es gibt weiterhin Knoten mit den konstanten Wahrheitswerten `true` und `false`. Diese gelten als Variablenknoten, tragen als Attribut Variable aber `"true"` bzw. `"false"`. Das heißt auch, dass echte Variablen diese beiden Strings als Namen nicht annehmen dürfen.

In Abbildung 2 finden Sie drei Beispiele für solche booleschen Binärbäume.

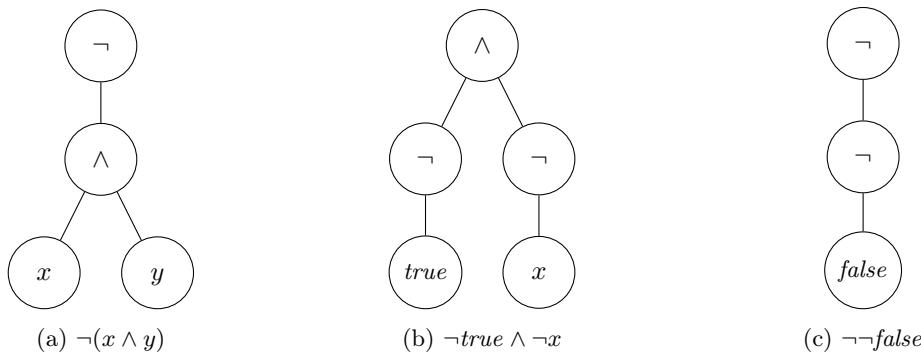


Abbildung 2: Beispiele für boolesche Binärbäume mit den jeweils repräsentierten Formeln

Ihre Implementierung sollte mindestens die folgenden Methoden beinhalten. Sie sollten dabei die Konzepte der Datenkapselung berücksichtigen. Hilfsmethoden müssen als `private` deklariert werden. In dieser Aufgabe dürfen Sie die in der Klasse `Utils` zur Verfügung gestellten Hilfsfunktionen, aber keine Bibliotheksfunktionen verwenden. Sie finden die Klasse im Lernraum.

**In dieser Aufgabe dürfen Sie *keine* Schleifen verwenden. Die Verwendung von Rekursion ist hingegen erlaubt.**

- a) Erstellen Sie eine Java-Klasse `BoolTreeNode` mit den Attributen `variable` (vom Typ `String`), `child1` und `child2` (vom Typ `BoolTreeNode`).
- b) Erstellen Sie die folgenden drei Konstruktoren, um Objekte vom Typ `BoolTreeNode` zu erzeugen.

```
BoolTreeNode(String variableInput)
BoolTreeNode(BoolTreeNode negated)
BoolTreeNode(BoolTreeNode conjunct1, BoolTreeNode conjunct2)
```

Der erste Konstruktor soll einen Knoten ohne Kinder erstellen, dessen Attribut `variable` den übergebenen Wert hat. Der zweite Konstruktor soll einen Negationsknoten erstellen, dessen (erstes) Kind der übergebene Knoten ist. Der dritte Konstruktor soll einen Konjunktionsknoten erstellen, dessen beiden Kinder die übergebenen Knoten sind.

Alle drei Konstruktoren sollen für den Benutzer der Klasse nicht sichtbar sein. Sorgen Sie dafür, dass die Attribute wie in der Einleitung beschrieben gesetzt werden. Sie müssen sich auch hier noch nicht um die Zulässigkeit der Eingaben kümmern. Dies wird in der nächsten Teilaufgabe behandelt.

- c) Jetzt kommen wir zu den Methoden, mit denen der Benutzer der Klasse neue Knoten erzeugen kann:

```
BoolTreeNode boolTreeTrueNode()
BoolTreeNode boolTreeFalseNode()
BoolTreeNode boolTreeVariableNode(String variableInput)
BoolTreeNode boolTreeNotNode(BoolTreeNode negated)
BoolTreeNode boolTreeAndNode(BoolTreeNode conjunct1,
                           BoolTreeNode conjunct2)
```

Greifen Sie dafür auf die soeben erstellten Konstruktoren zurück. Sie müssen jetzt auch überprüfen, ob die übergebenen Parameter den Anforderungen entsprechen. Kann ein Knoten wegen einer Anforderungsverletzung nicht erstellt werden, so ist eine aussagekräftige Fehlermeldung auszugeben und `null` zurückzugeben. Nutzen Sie für Fehlermeldungen die statische Methode `error` aus der Klasse `Utils`. Entscheiden Sie begründet, ob Sie diese Methoden statisch implementieren. Setzen Sie jeweils auch einen sinnvollen Zugriffsmodifikator (z.B. `public` oder `private`) .

- d) Erstellen Sie eine Methode, um die Tiefe eines `BoolTreeNode` zu bestimmen.

```
public int depth()
```

Ein Variablenknoten habe dabei Tiefe 0, ein Negationsknoten eine um eins höhere Tiefe als sein Kind, und ein Konjunktionsknoten habe eine um eins höhere Tiefe als das Tiefste der beiden Kinder. Setzen Sie diese rekursive Definition der Tiefe einer Formel in Ihrer Implementierung auch rekursiv um.

*Hinweis:* Sie finden in der Klasse `Utils` eine Methode `max`, um das Maximum von zwei `int`-Werten zu bestimmen.

- e) Erstellen Sie folgende Methoden, um zu bestimmen, von welcher Art ein Knoten ist.

```
public boolean isLeaf()
public boolean isTrueLeaf()
public boolean isFalseLeaf()
public boolean isNegation()
public boolean isConjunction()
```

Dabei soll `isLeaf()` genau dann `true` zurückgeben, wenn der Knoten ein Variablenknoten ist (d.h. wenn beide Kinder des Knotens den Wert `null` haben). Die Methoden `isTrueLeaf()` und `isFalseLeaf()` sollen genau dann `true` zurückgeben, wenn der Knoten den konstanten Wahrheitswert `true` bzw. `false` repräsentiert. Durch `isNegation()` soll genau dann `true` zurückgegeben werden, wenn der Knoten genau ein nicht-`null`-Kind hat (und zwar das erste). Die Methode `isConjunction()` soll genau dann `true` zurückgeben, wenn der Knoten zwei nicht-`null`-Kinder hat.

- f) In dieser Teilaufgabe soll die boolesche Formel, die durch den Baum repräsentiert wird, ausgewertet werden.

```
public boolean evaluate(String... trueVars)
```

Dabei soll der Baum unverändert bestehen bleiben. Die konstanten Wahrheitswerte, die Konjunktion und die Negation haben die übliche Semantik. Der Methode wird weiterhin eine Menge von Strings übergeben. Dies sind die Namen der Variablen, die zu `true` evaluiert werden. Alle anderen Variablen werden zu `false` evaluiert.

Hinweise:

- Sie finden die statische Methode `evaluateVariable(String elem, String... strings)` in der Klasse `Utils`. Diese führt die Evaluation einer Variable `elem` bei gegebenen (mit `true` belegten) Variablen `strings` durch. Hier werden auch die Fälle "`true`" und "`false`" behandelt.

## Hausaufgabe 5 (Programmieren mit rekursiven Datenstrukturen): (5 + 6 + 5 + 4 = 20 Punkte)

Das Spiel TicTacToe funktioniert wie folgt: Auf einem 3x3 Raster setzen zwei Spielende (hier Min und Max genannt) abwechselnd die Symbole X (Min) und O (Max). Die erste Person, welche eine ganze Reihe, eine ganze Spalte, oder eine der beiden Diagonalen vollständig mit ihrem Symbol füllen kann, gewinnt. Ist das Raster komplett mit Symbolen gefüllt und niemand hat gewonnen, so ist das Spielergebnis unentschieden. Ein Ausschnitt aus einem solchen Spiel ist in Abbildung 3 zu sehen.

In dieser Aufgabe vervollständigen Sie den Code für ein TicTacToe Programm. Der Spieler Min ist hierbei der Computer und Max ist der menschliche Spieler. Im jetzigen Stand fehlt dem Programm noch die Möglichkeit, den Computer Spielzüge machen zu lassen. Um das zu ermöglichen, soll die Klasse `MinMaxTree` vervollständigt werden, welche aktuell so aussieht:

```
public class MinMaxTree {
    private GameState state;
    private final boolean min;

    private MinMaxTree[] successors;

    public MinMaxTree(boolean min, GameState state){
        this.min = min;
        this.state = state;

        this.successors = null;
    }

    public MinMaxTree[] getSuccessors() {
        return successors;
    }

    public GameState getState() {
        return state;
    }

    public boolean isMin() {
        return min;
    }
}
```

Jedes `MinMaxTree`-Objekt repräsentiert einen Baum wie in Abbildung 3. Jeder Knoten eines solchen Baums repräsentiert einen Spielzustand mithilfe des Attributs `state` vom vorgegebenen Typ `GameState`. Das Attribut `min` des Typs `boolean` wird genutzt, um zu speichern, ob im aktuellen Spielzustand Min oder Max am Zug ist. Lässt sich das Spiel im repräsentierten Zustand fortsetzen, dann kann der Knoten für jeden möglichen Spielzug des aktiven Spielers einen Nachfolger-Knoten haben, welcher den resultierenden Spielzustand nach diesem Spielzug repräsentiert. Diese Nachfolger werden mithilfe des Attributs `successors` verwaltet. Da bei Erzeugung eines `MinMaxTree`-Objekts die möglichen Spielzüge noch nicht untersucht werden, wird `successors` zunächst auf `null` gesetzt.

Die Idee ist, dass ein solcher Baum alle möglichen Spielverläufe ausgehend vom aktuellen Spielzustand darstellen kann. Ausgehend davon kann der aktive Spieler sich für jeden möglichen Zug die resultierenden Spielverläufe anschauen und entscheiden, welcher Zug die besten Gewinnaussichten bietet.

Ein Konstruktor sowie Getter sind bereits gegeben. Achten Sie bei der Implementierung der nachfolgenden Methoden darauf, die Namen und Argumente genau aus der Aufgabenstellung zu übernehmen. Ansonsten wird das gesamte Spiel vermutlich nicht funktionieren.

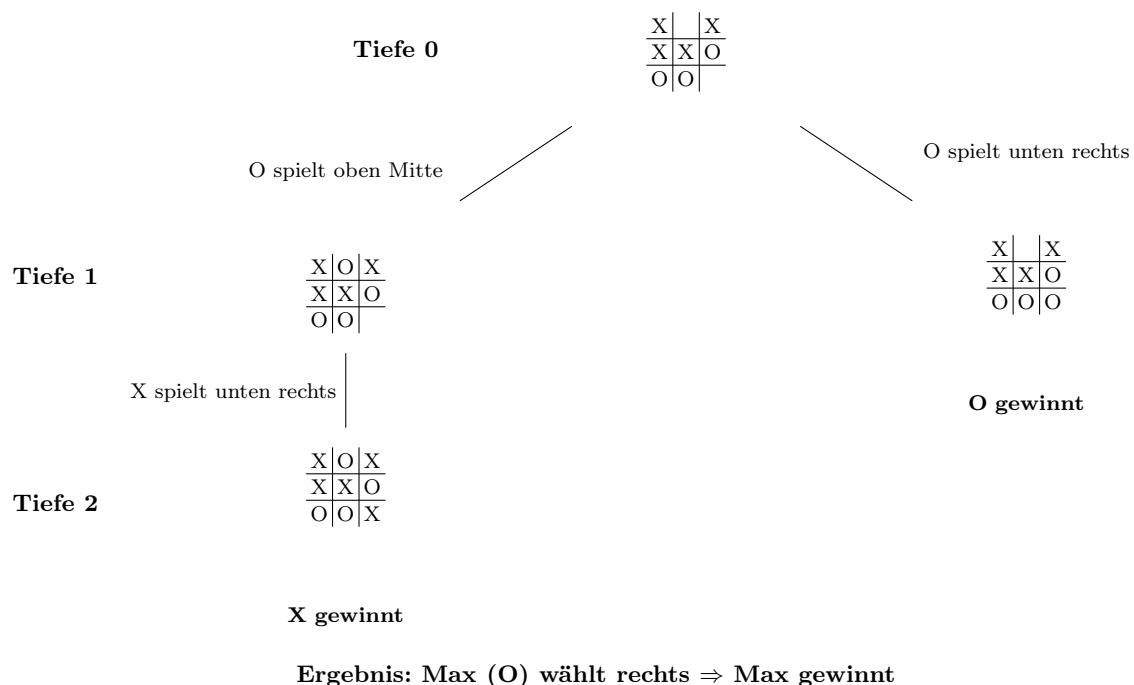


Abbildung 3: Ein `MinMaxTree` Objekt. Der Spieler Min spielt X, während Max die Symbole O spielt.

Ihre Implementierung soll genau die folgenden Methoden beinhalten. Berücksichtigen Sie dabei die Konzepte der Datenkapselung. In dieser Aufgabe dürfen Sie keine Hilfsmethoden schreiben und keine Bibliotheksfunktionen verwenden, sofern dies nicht explizit erlaubt ist. Sie dürfen aber die Methoden vorangehender Teilaufgaben nutzen, auch wenn Sie diese nicht selbst implementiert haben.

- a) Ergänzen Sie die Klasse `MinMaxTree` um eine öffentliche Methode `unfold()`. Diese Methode soll ein neues `MinMaxTree`-Objekt als Ergebnis zurückgeben, welches in der Wurzel den gleichen `state` und den gleichen Spieler (`min`) wie das aktuelle `MinMaxTree`-Objekt hat. Darüber hinaus sollen im Ergebnis-Objekt die direkten Nachfolge-Knoten wie oben beschrieben gesetzt werden. Das aktuelle Objekt, auf welchem die Methode aufgerufen wird, soll dabei nicht verändert werden. Nutzen Sie die Methode `getSuccessors(boolean min)` der Klasse `GameState`, welche ein Array des Typs `GameState[]` zurückgibt. Dieses enthält alle möglichen direkt nachfolgenden Spielzustände unter der Annahme, dass der durch das Argument `min` gekennzeichnete Spieler am Zug ist. Achten Sie darauf, bei den Nachfolgern das Attribut `min` korrekt zu setzen. TicTacToe wird abwechselnd gespielt!

Beispiel:

Initialisiert man ein neues `MinMaxTree`-Objekt mit dem in Abbildung 3 bei Tiefe 0 gezeigten Spielzustand und wendet `unfold()` an, so soll ein `MinMaxTree`-Objekt zurückgegeben werden, welches den in Abbildung 3 gezeigten Baum bis zur Tiefe 1 repräsentiert. Der Knoten, bei welchem **X** gewinnt, ist also noch nicht enthalten.

- b) Ergänzen Sie die Klasse `MinMaxTree` um eine öffentliche Methode `unfold(int depth)`. Diese Methode soll ein neues `MinMaxTree`-Objekt als Ergebnis zurückgeben, welches in der Wurzel den gleichen `state` und den gleichen Spieler (`min`) wie das aktuelle `MinMaxTree`-Objekt hat. Darüber hinaus sollen im Ergebnis-Objekt die Nachfolge-Knoten bis zur durch das Argument `depth` angegebenen Tiefe gesetzt werden. Das aktuelle Objekt, auf welchem die Methode aufgerufen wird, soll dabei nicht verändert werden.

Beispiel:

Initialisiert man ein neues `MinMaxTree`-Objekt mit dem in Abbildung 3 bei Tiefe 0 gezeigten Spielzustand und wendet `unfold(2)` an, so soll ein `MinMaxTree`-Objekt zurückgegeben werden, welches den in Abbildung 3 gezeigten Baum repräsentiert. Der Knoten, bei welchem **X** gewinnt, ist jetzt also auch enthalten.

- c) Ziel der Datenstruktur `MinMaxTree` ist es, den besten Spielzug für den aktuellen Spieler ermitteln zu können. Intuitiv ist ein Spielzug gut, wenn der ziehende Spieler dadurch bessere Gewinnaussichten erhält. Um den besten Spielzug ermitteln zu können, nutzen wir einen Algorithmus, welcher als Minimax bekannt ist. Hier wird jedem Knoten im Baum eine Bewertung gegeben, abhängig davon, wie groß die Aussicht ist, dass der im Knoten aktive Spieler das Spiel gewinnen kann. In unserem Fall wird dazu jeder Knoten, in welchem Min bereits gewonnen hat, mit  $-1$  bewertet. Jeder Knoten, in welchem Max bereits gewonnen hat, wird mit  $1$  bewertet. Ein Knoten ohne Nachfolger, in welchem niemand gewonnen hat, wird mit  $0$  bewertet. Ist das Spiel im aktuellen Knoten noch nicht beendet, dann wird die Bewertung folgendermaßen berechnet: Falls Min am Zug ist, wird das Minimum der Bewertungen aller Nachfolger-Zustände gebildet. Umgekehrt wird stattdessen das Maximum gebildet, falls Max am Zug ist.

Implementieren Sie in der Klasse `MinMaxTree` eine private Methode `score()`, welche eine Bewertung wie gerade beschrieben berechnet. Sollte es keine Nachfolger geben, verwenden Sie die Methode `evaluate()` der vorgegebenen Klasse `GameState`, um die Bewertung zu ermitteln. Ansonsten bilden Sie die Bewertung wie gerade beschrieben. Dabei dürfen Sie die statischen Methoden `min(int a, int b)` und `max(int a, int b)` der Klasse `Math` verwenden.

Beispiel:

Der Knoten mit Tiefe 0 in Abbildung 3 erhält Bewertung 1, da der aktive Spieler Max sofort gewinnen kann, indem er unten rechts spielt.

- d) Ergänzen Sie die Klasse `MinMaxTree` um eine öffentliche Methode `makeMove()`. Diese Methode soll unter den Nachfolgern des aktuellen Objekts den besten Nachfolger bestimmen und diesen als Ergebnis zurückgeben. Ist Min am Zug, dann ist der beste Nachfolger derjenige mit der kleinsten Bewertung. Falls Max am Zug ist, dann ist es derjenige mit der größten Bewertung. Falls es mehrere Nachfolger mit optimaler Bewertung gibt, ist es egal, welcher davon zurückgegeben wird. Sollte es keine Nachfolger geben, darf sich die Methode beliebig verhalten.

Beispiel:

Angewendet auf den Knoten mit Tiefe 0 in Abbildung 3 soll ein `MinMaxTree`-Objekt mit dem rechten Nachfolger-Spielzustand zurückgegeben werden, bei welchem Min am Zug ist und welches selbst keine Nachfolger hat.

**Hausaufgabe 6 (Deck 6):****(Codescape)**

Lösen Sie die Missionen von Deck 6 des Codescape Spiels. Ihre Lösung für die Codescape Missionen wird nur dann für die Zulassung gezählt, wenn Sie Ihre Lösung vor der einheitlichen Codescape Deadline am Freitag, den 30.01.2026, um 23:59 Uhr abschicken.