Aufgabenblatt 6

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Freitag, 10.01.2020 13:00 Uhr** in TUWEL hoch.
- Zusätzlich müssen Sie in TUWEL ankreuzen, ob Sie die Aufgabe gelöst haben und während der Übung präsentieren können.
- Ihr Programm muss kompilierbar und ausführbar sein.
- Ändern Sie bitte **nicht** die **Dateinamen** und die **vorhandene Ordnerstruktur**.

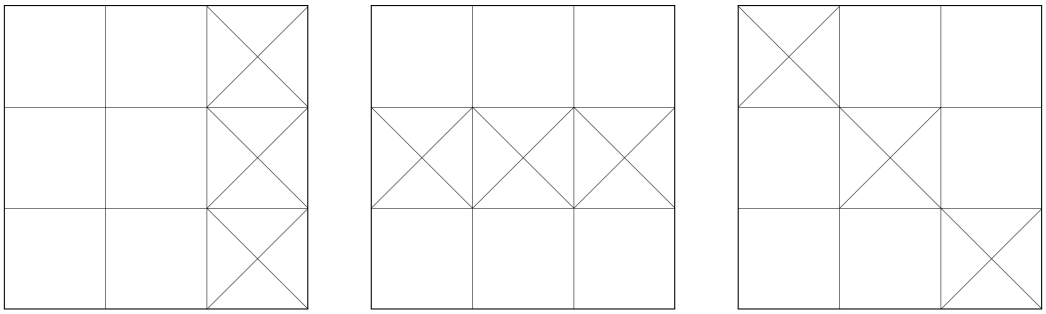
In diesem Aufgabenblatt werden folgende Themen behandelt:

- Ein- und zweidimensionale Arrays
- Methoden
- Rekursion
- Grafische Darstellung
- `minimax`-Algorithmus

Aufgabe 1

Implementieren Sie folgende Aufgabenstellung:

- Bei dieser Aufgabe sollen Sie das Spiel *Tic Tac Toe* implementieren. Bei diesem Spiel spielen zwei Personen gegeneinander und versuchen auf einem 3×3 Spielfeld drei Zeichen entweder vertikal, horizontal oder diagonal (siehe Abbildung 1) anzuordnen. Gewonnen hat jene Person, die als erstes drei Zeichen in Position gebracht hat. Bei Ihrer Implementierung soll es aber auch die Möglichkeit geben, gegen den Computer zu spielen. Dazu müssen Sie einen Algorithmus implementieren, der es dem Computer ermöglicht, einen Spielzug zu finden und auszuführen.



(a) (b) (c)

Abbildung 1: *Tic Tac Toe* Spielfeld mit drei `'X'` in einer a) vertikalen, b) horizontalen und c) diagonalen Anordnung. Die Abbildungen dienen zur Veranschaulichung, repräsentieren aber keinen Spielzustand.

- Sie haben in `main` bereits die Einstellungen für das `StdDraw`-Ausgabefenster definiert. Zusätzlich ist ein zweidimensionales `char`-Array `gameBoard` gegeben, das als Spielfeld verwendet werden soll. Die Variable `twoPlayer` wird dazu verwendet, um zu steuern, ob zwei Personen gegeneinander spielen (`true`), oder ob eine Person gegen den Computer spielt (`false`). Mit der Variable `player` wird gesteuert, wer an der Reihe ist. Hier steht `true` für die spielende Person mit dem `'X'` und `false` für die spielende Person/Computergegner mit dem `'O'`. Es gibt noch eine Variable `maxDepth`, mit der ausgewählt wird, wie viele Spielzüge vorausgeschaut werden (Tiefe), falls gegen den Computer gespielt werden soll. Die maximale Tiefe für das Spiel *Tic Tac Toe* ist 8, da nach dem ersten Spielzug 8 weitere Spielzüge auf dem 3×3 Spielfeld möglich sind, wenn der aktuelle Spielzug des Computers zur Tiefenzählung hinzugezählt wird.

  Spielablauf: In `main` wird in einer Schleife solange das Spiel gespielt, bis das Spiel *Gewonnen* wurde, oder es ein *Unentschieden* gibt. Ist die Variable `twoPlayer == true`, dann spielen zwei Personen gegeneinander und es muss immer auf die Mausklicks innerhalb des Spielfeldes reagiert werden. Dazu können Sie die Methode `StdDraw.isMousePressed()` verwenden. Danach können Sie die Koordinaten des Klicks mit `StdDraw.mouseX()` und `StdDraw.mouseY()` auslesen. Die Koordinaten müssen umgerechnet werden, damit der Klick richtig dem Spielfeldeintrag in `gameBoard` zugeordnet werden kann. Nach jedem Spielzug wird das Spielfeld mit allen bis dahin gesetzten Zeichen gezeichnet (siehe Methode `drawGameBoard`). Ist die Variable `twoPlayer == false`, dann spielt eine Person gegen den Computer und es wird

abwechselnd ein Klick ausgeführt (menschlicher Spielzug) und dann der Algorithmus (*minimax*[1]) für die Spielzugsuche des Computers aufgerufen (`minimax`-Methode).

- Implementieren Sie eine rekursive Methode `minimax`:

```
int[] minimax(char[][] gameBoard, boolean player, int depth)
```

Die Methode sucht den bestmöglichen Spielzug für den Computer nach jedem Spielzug der spielenden Person. Dazu bewertet der Algorithmus die aktuelle Spielsituation für den Computer bzw. der spielenden Person und gibt zurück, welche Position die beste Bewertung hat. Dafür hat die Methode als Rückgabewert ein eindimensionales Array der Länge 3. Bei Index 0 im Array wird die y-Position (Zeile im Array) und bei Index 1 die x-Position (Spalte im Array) für die Rückgabe eines geeigneten Spielzuges abgespeichert. Bei Index 2 des Arrays wird der Bewertungswert für einen Spielzug hinterlegt und zurückgegeben. Auch wenn nur der Computer die Spielzugsuche benötigt, muss dieser die Möglichkeit haben, einen Spielzug für die spielende Person auszuführen, um verschiedene Spielstellungen durchprobieren zu können. Dieses abwechselnde *Spielzug durchführen* kann mit der Variablen `player` realisiert werden. Ist `player == false` wird ein Spielzug für den Computer und bei `player == true` ein Spielzug für die spielende Person durchgeführt. Die Variable `depth` steuert die Anzahl der Spielzüge, die vorausgeschaut werden sollen und wird bei jedem rekursiven Aufruf um 1 reduziert.

Die Methode wird in `main` das erste Mal mit `player = false` aufgerufen, da der Computer nach dem menschlichen Spielzug an der Reihe ist. Dazu probiert der Computer auf dem nächsten freien Spielfeld einen Spielzug und evaluiert diesen. Dazu wird dieses Spielfeld mit `'O'` markiert. Der Algorithmus prüft, bevor die nächste Rekursion aufgerufen wird, ob dieses `'O'` zu einem Gewinn des Computers führt, das Spielfeld voll ist oder `depth-1` bereits 0 ist. Wenn keines dieser Kriterien zutrifft, dann wird die `minimax`-Methode erneut mit `player = true` und `depth-1` aufgerufen. Nun wird für die spielende Person ein Spielzug durchgeführt und ein freies Feld mit einem `'X'` belegt. Der Algorithmus prüft, bevor die nächste Rekursion aufgerufen wird, ob dieses `'X'` zu einem Gewinn der spielenden Person führt, das Spielfeld voll ist oder `depth-1` bereits 0 ist. Wenn keines dieser Kriterien zutrifft, dann wird die `minimax`-Methode erneut mit `player = false` und `depth-1` aufgerufen. Mit diesem Schema führt der Algorithmus für alle freien Felder einer Spielstufe abwechselnd einen Spielzug für den Computer bzw. die spielende Person aus, bis eines der Kriterien zutrifft und die Rekursion abgebrochen wird.

Wenn eine Spielstellung zu einem Gewinn für den Computer oder die spielende Person führt, wird wie zuvor erwähnt die Rekursion abgebrochen. Für diese Situation muss ein Rückgabewert definiert werden. Wir legen fest, dass ein Gewinn des Computers mit +1 und ein Gewinn der spielenden Person mit -1 bewertet wird. Wenn eine Gewinnstellung für den Computer gefunden wurde, dann wird +1 bei Index 2 des Rückgabearrays hinterlegt und die Koordinaten des Feldes, die zu diesem Gewinn führen. Wenn eine Gewinnstellung zu Gunsten der spielenden Person gefunden wurde, dann wird der Wert -1 bei Index 2 des Rückgabearrays hinterlegt und die Koordinaten des Feldes, das verwendet werden muss, um einen Gewinn der spielenden Person zu verhindern. Für den Fall, dass das Spielfeld voll oder `depth-1 == 0` mit keinem Gewinner ist, wird bei Index 2 eine 0 hinterlegt. Die Bewertung inklusive der

[1] `https://de.wikipedia.org/wiki/Minimax-Algorithmus`

Koordinaten werden durch die rekursiven Aufrufe bis zum Aufruf in `main` zurückgegeben, wo letztendlich der Spielzug für den Computer durchgeführt wird.

Nachfolgend finden Sie eine Beschreibung des zu implementierenden `minimax`-Algorithmus in Form von Pseudo[2]-Code:

```
function minimax(gameBoard, player, depth)
  declare an int array retArray of length 3
  if player is true
    set retArray[2] to integer max value
  else
    set retArray[2] to integer min value
  for all rows in gameBoard
    for all columns in gameBoard
      if field in gameBoard is empty
        if player is true
          set current field to 'X'
        else
          set current field to 'O'
        if winner is player 'X'
          fill retArray with row, column, -1
        else if winner is player 'O'
          fill retArray with row, column, 1
        else if gameBoard is full or depth-1 is 0
          fill retArray with row, column, 0
        else
          call minimax and save return value in an int array tempArray
          if player is true
            if tempArray[2] is smaller than retArray[2]
              save row, column and tempArray[2] in retArray
          if player is false
            if tempArray[2] is greater than retArray[2]
              save row, column and tempArray[2] in retArray
        set current field in gameBoard to ' ' (empty)
  return retArray
```

Zusätzlich wird in Abbildung 2 ein kompletter Spieldurchlauf mit `maxDepth = 3` gezeigt. Spielzüge der spielenden Person werden durch das Kreuz gekennzeichnet. Zur Visualisierung der Vorausschau des Computers wurden in den Abbildungen 2b, 2d, 2f und 2h die jeweiligen Züge und Kombinationen, die der `minimax`-Algorithmus analysiert hat, eingezeichnet (dient zur Veranschaulichung und muss nicht implementiert werden). Diese vier Abbildungen veranschaulichen alle Entscheidungen, die getroffen werden können. Jedes der möglichen Felder für einen Spielzug des Computers beinhaltet alle möglichen Folgezüge. Beispielsweise enthält das linke obere Feld in Abbildung 2b alle Spielverläufe, wenn nach dem Start (`'X'` in der Mitte) dort als nächstes ein Spielzug (`'O'`) ausgeführt wird. Für den nächsten Zug wird das Feld wieder in neun Bereiche geteilt, die wieder alle möglichen Folgezüge enthalten. Diese Prozedur wird fortgesetzt, solange noch Züge möglich sind. Nach dieser Analyse

[2] `https://de.wikipedia.org/wiki/Pseudocode`

wird ein Spielzug des Computers durchgeführt, welcher in den Abbildungen 2c, 2e, 2g und 2i als Kreis (rot) gekennzeichnet ist. In Abbildung 2j ist das Spielfeld voll und es wurde ein *Unentschieden* erreicht.

Vorbedingungen: `gameBoard != null`, `gameBoard.length == gameBoard[i].length` (gilt für alle gültigen `i`), `gameBoard.length == 3` und `depth > 0`.

- Implementieren Sie eine Methode `checkIfFull`:

```
boolean checkIfFull(char[][] gameBoard)
```

Diese Hilfsmethode überprüft, ob das Spielfeld voll ist. Das heißt, dass überprüft wird, ob bei jedem Arrayeintrag ein `'X'` oder `'O'` vorhanden ist. Wenn es noch ein Leerzeichen (`' '`) geben sollte, dann wird `false` zurückgegeben, ansonsten `true`.

Vorbedingungen: `gameBoard != null`, `gameBoard.length == gameBoard[i].length` (gilt für alle gültigen `i`) und `gameBoard.length == 3`.

- Implementieren Sie eine Methode `checkIfWinner`:

```
boolean checkIfWinner(char[][] gameBoard, boolean player)
```

Diese Hilfsmethode überprüft, ob drei gleiche Zeichen in einer Spalte, Zeile oder Diagonale vorhanden sind. Wenn die Variable `player == true` ist, dann wird überprüft, ob drei `'X'` in erwähnter Konstellation vorhanden sind und bei `player == false` wird überprüft, ob drei `'O'` in entsprechenden Feldern gefunden werden können. Wird eine Gewinnposition detektiert, dann wird `true` zurückgegeben, ansonsten `false`.

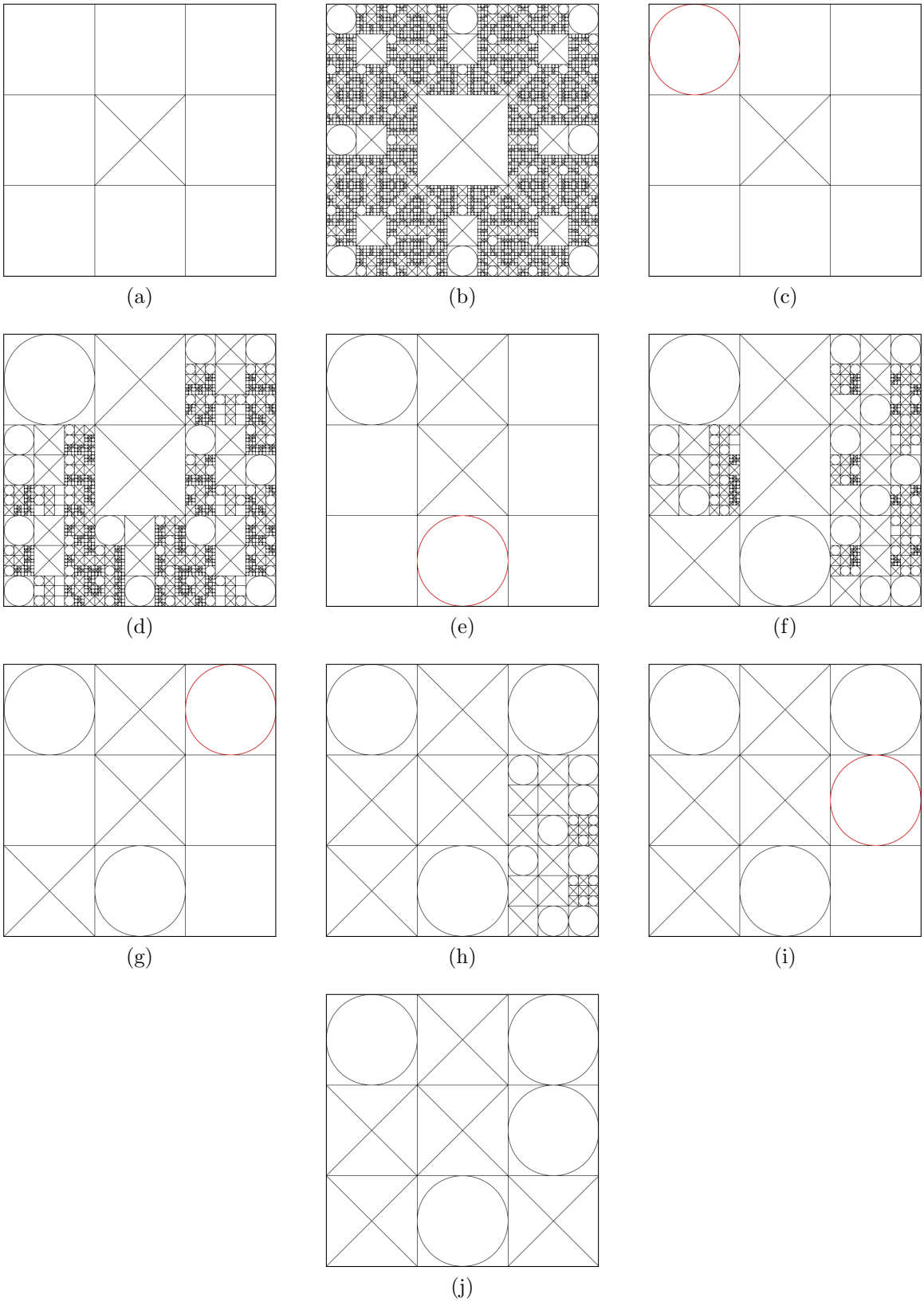
Vorbedingungen: `gameBoard != null`, `gameBoard.length == gameBoard[i].length` (gilt für alle gültigen `i`) und `gameBoard.length == 3`.

- Implementieren Sie eine Methode `drawGameBoard`:

```
void drawGameBoard(char[][] gameBoard, int size)
```

Die Methode zeichnet das Spielfeld `gameBoard` mit den vier Trennlinien, um die neun Felder zu kennzeichnen. Für jeden Eintrag `'X'` in `gameBoard` zeichnet die Methode zwei Linien, die sich zu einem Kreuz kombinieren. Für jeden Eintrag `'O'` in `gameBoard` wird ein Kreis in das entsprechende Feld gezeichnet. Mit dem Parameter `size` wird der Methode die Größe des StdDraw-Fensters mitgegeben, damit das Spielfeld fensterfüllend gezeichnet werden kann.

Vorbedingungen: `gameBoard != null`, `gameBoard.length == gameBoard[i].length` (gilt für alle gültigen `i`), `gameBoard.length == 3` und `size >= 200`.



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j)

Abbildung 2: Ein kompletter Spielverlauf Mensch (`'X'`) vs. Computer (`'O'`) mit einer Vorausschau der Tiefe 3 des Computers.