

Aufgabenblatt 6

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Mittwoch, 07.01.2026 23:55 Uhr** in TUWEL hoch.
- Zusätzlich müssen Sie in TUWEL ankreuzen, ob Sie die Aufgabe gelöst haben.
- Ihr Programm muss kompilierbar und ausführbar sein.
- Ändern Sie bitte **nicht** die **Dateinamen** und die **vorhandene Ordnerstruktur**.
- Bitte achten Sie auch darauf, dass Sie eine eigenständige Lösung erstellen. **Wir werden** bei dieser Aufgabe auf Plagiate überprüfen und **offensichtliche Plagiate nicht bewerten**.
- Es werden nur vollständig spielbare Lösungen bewertet!

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Arrays
- Methoden
- Grafische Darstellung
- Spiellogik

Aufgabe 1 (6 Punkte)

Implementieren Sie folgende Aufgabenstellung:

- Bei dieser Aufgabe soll das Spiel *Black Box Chess* implementiert werden. Bei diesem Spiel geht es darum, die Platzierung von fünf Schachfiguren mit möglichst wenig Hinweisen und Fehlversuchen zu erraten. Ein neues Spiel beginnt mit 100 Punkten. Ein Hinweis kostet 3 Punkte, ein Fehlversuch 7. Sobald der abgegebene Versuch korrekt ist, endet das Spiel mit dem aktuellen Punktestand. Sobald 0 Punkte erreicht sind, hat man verloren. Ein Hinweis wird gekauft, indem man eines der 59 nicht belegten Felder anklickt. Dabei erscheint eine Zahl von 0 bis 5. Diese Zahl sagt aus, wie viele Figuren auf dieses Feld ziehen, ausgehend von der gesuchten Platzierung. Dabei gelten für die Zugmöglichkeiten die Standard-Regeln des Schachspiels¹ mit der zusätzlichen Regel, dass nicht nur der Springer, sondern **alle** Figuren andere Figuren beim Zug überspringen können. Abbildung 1 zeigt ein neu generiertes Spiel.

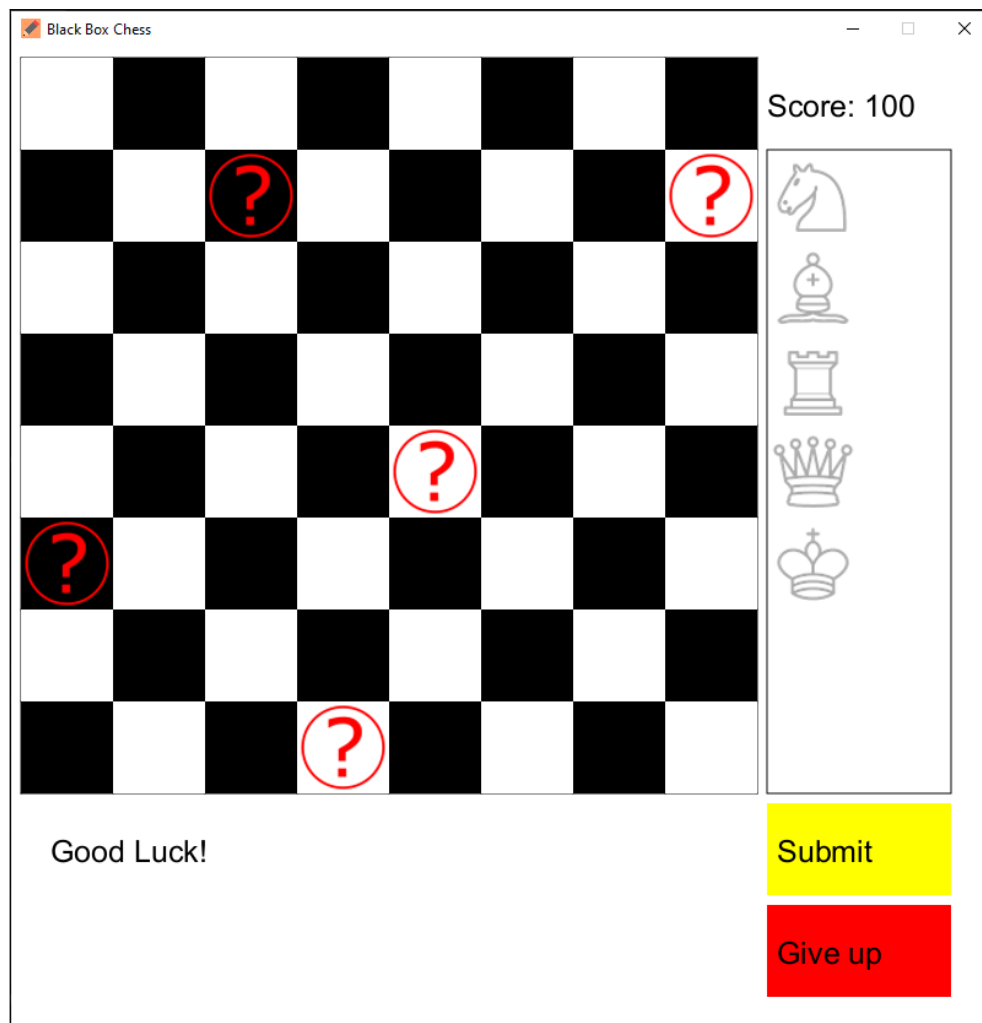


Abbildung 1: Ein neu generiertes Spiel.

¹<https://de.wikipedia.org/wiki/Schach#Zugregeln>

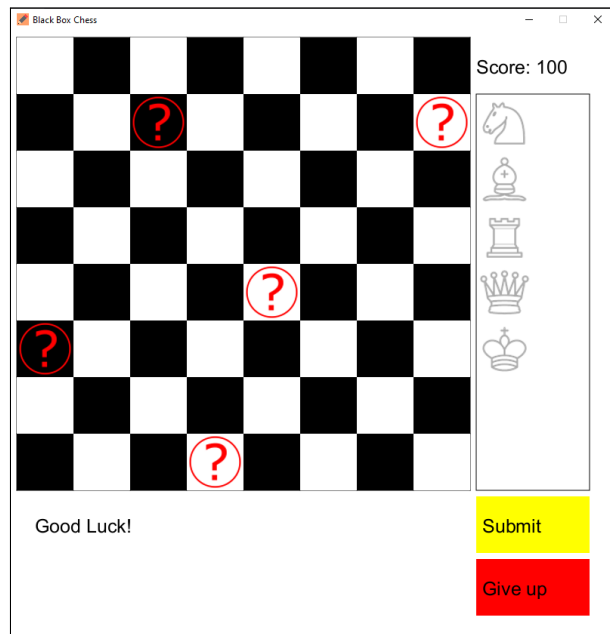
- **Spiefeldaufbau:** Das Spielfeld ist in fünf Zonen aufgeteilt. Oben links befindet sich das Herzstück: das 8×8-Schachbrett mit der aktuellen Platzierung und eventuell aufgedeckten Hinweisen. Auf der rechten Seite befindet sich oben der aktuelle Punktestand, und darunter die noch nicht platzierten Figuren. Unter der Sidebar mit den Figuren sind zwei Schaltflächen - jeweils zum Einreichen des Versuches und zur Aufgabe des Spieles. Die Fläche unter dem Schachbrett ist für Informationstext reserviert. Interaktion via Mausklick ist mit allen Bereichen, abgesehen von Textbereichen, möglich, wobei die Koordinaten des Mausklicks ausgewertet und die entsprechende Aktion durchgeführt wird.

Die Informationen für die Hinweise sind in dem 8×8-**int**-Array **board** hinterlegt. Der Index des äußeren Arrays stellt die Zeile, und der des inneren Arrays die Spalte dar - so können die Koordinaten des Schachbretts simuliert werden. Bei den Koordinaten an denen die Figuren platziert sind, wird der Wert -1 gespeichert. Alle anderen Felder enthalten Werte von 0 bis 5, je nach dem, wie viele Figuren das entsprechende Feld erreichen können. Zusätzlich gibt es noch ein 8×8-**boolean**-Array **hints**, in dem hinterlegt wird, welche Hinweise bereits aufgedeckt wurden.

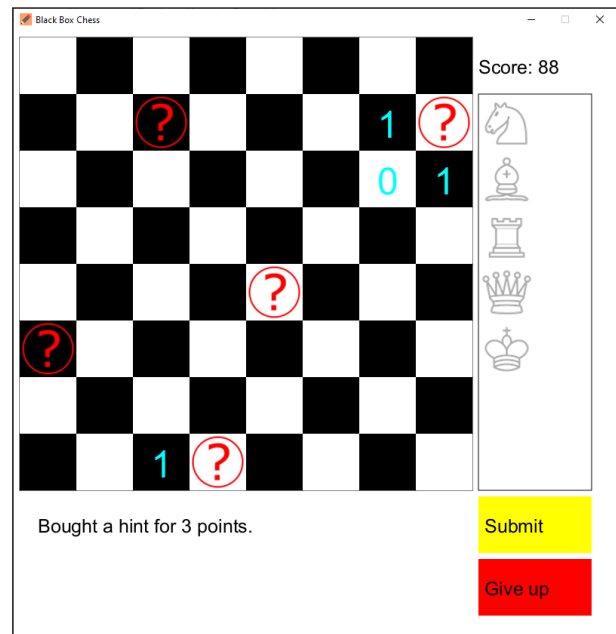
Die zu erratende Platzierung ist im 6×2-**int**-Array **placements** gespeichert. Der Index des äußeren Array steht für eine Figur (0 = keine, 1 = Springer, 2 = Läufer, 3 = Turm, 4 = Dame, 5 = König). Das zugehörige innere Array symbolisiert die Koordinaten der jeweiligen Figur in der gesuchten Platzierung, in der Reihenfolge [Reihe, Spalte]. Eine nicht platzierte Figur hat die Koordinaten [-1,-1]. Die aktuelle Platzierung ist im Array **attempt** gespeichert, welches die Information der aktuellen Platzierung mit derselben Logik wie **placements** speichert. Ein Klick auf „Submit“ löst einen Vergleich der Arrays **placements** und **attempt** aus. Die Arrays sind genau dann gleich, wenn die aktuelle Platzierung der gesuchten entspricht.

- **Spielablauf:** Zu Beginn werden bereits außerhalb der **main**-Methode einige Konstanten auf Basis der Fenstergröße (standardmäßig 800×800 Pixel) berechnet, die Punkte auf 100 gesetzt und die Bilder für die Figuren und Markierungen geladen. In **main** wird dann die Lösung mit Hilfe der Methode **generatePlacement** generiert und im Array **placements** gespeichert. Dann wird die game-loop gestartet.

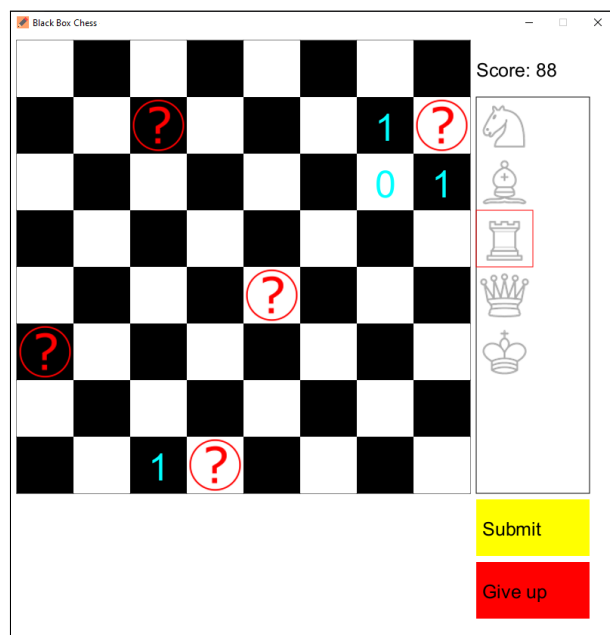
Für einen möglichen Spielablauf betrachten wir zunächst erneut die Ausgangsposition des generierten Spiels 2a. Um nicht blind eine aus 120 verschiedenen Positionierungen zu raten, kaufen wir Hinweise. Die 4 per Mausklick gekauften Hinweise 2b lassen bereits den Schluss zu, dass oben rechts nur der Turm positioniert werden kann. Daher wählen wir den Turm 2c per Mausklick, der nun rot umrandet erscheint. Durch den Klick auf die entsprechende Platzierungsmarkierung wird der Turm auf dieses Feld platziert 2d. Ein Klick auf Koordinaten, die nicht zu einem der fünf gesuchten Feldern gehören, hätte den Turm „abgewählt“, und der rote Rahmen wäre dementsprechend verschwunden. Der bereits positionierte Turm kann auch wieder vom Feld entfernt werden, indem man entweder eine andere Figur auf dieses Feld positioniert, oder den Turm erneut auswählt und dann ein weiteres Mal auf ihn klickt. Wir kaufen nun weitere Hinweise und setzen die entsprechenden Figuren. Dann reichen wir den Versuch durch Klick auf „Submit“ ein 3a - doch leider ist uns da ein Fehler unterlaufen. Mit wenigen Klicks tauschen wir König und Dame aus und versuchen es erneut 3b - mit Erfolg! Das Spiel endet mit einer Niederlage, wenn man den „Give Up“-Button anklickt, das **CodeDraw**-Fenster schließt, oder 0 Punkte erreicht.



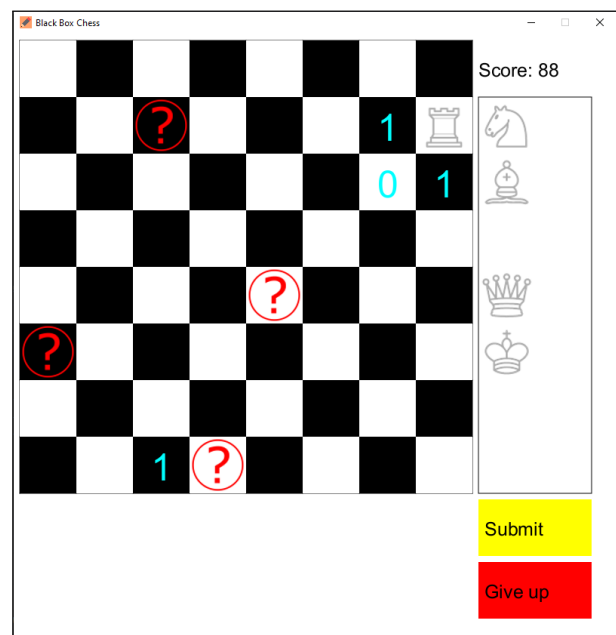
(a)



(b)



(c)



(d)

Abbildung 2: Verschiedene Zustände des Spiels.

Es folgen im Anschluss detailliertere Beschreibungen der Methoden (inklusive des Enum `Piece`), die als Grundlage für die Aufgabenstellung verwendet werden. Fünf Methoden davon müssen Sie selbst implementieren, um alle Punkte für dieses Aufgabenblatt zu bekommen. Diese fünf Methoden sind extra gekennzeichnet. Sie dürfen noch zusätzliche Hilfsmethoden schreiben, wenn Sie klar und schlüssig argumentieren können, warum diese implementiert wurden.

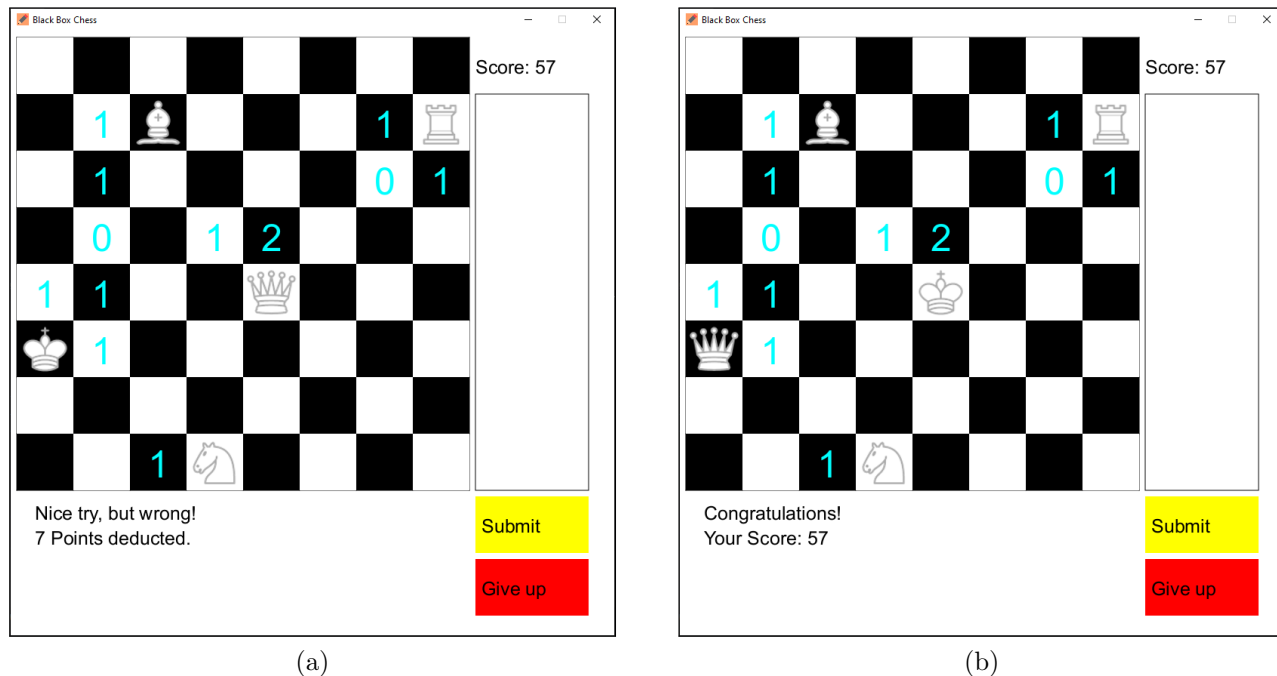


Abbildung 3: Ein Fehlversuch und die Verbesserung.

- Das Enum `Piece` ist eine sprechende Repräsentation der Figuren als Zahlen. Mit der Methode `ordinal()` kann ein Enum-Wert in seinen korrespondierenden Wert als `int` umgewandelt werden. Beispiel: Der Aufruf von `Piece.ROOK.ordinal()` gibt den `int` 3 zurück. Umgekehrt kann eine Zahl auch auf den entsprechenden `Piece` umgewandelt werden, indem auf das Array zugegriffen wird, das von `Piece.values()` zurückgegeben wird. So ist `Piece.values()[3]` äquivalent zu `Piece.ROOK`.
- Die Methode `void main(String[] args)` ist vorgegeben und konfiguriert die Parameter des `CodeDraw`-Objekts `game`. Durch Änderung des Wertes von `CANVAS_HEIGHT` werden alle anderen Parameter abhängig von diesem Wert gesetzt. Anschließend wird per Zufallsgenerator eine Positionierung erstellt und die Bilder der Schachfiguren geladen. Das Spiel wird dann in einer Schleife ausgeführt, wobei in jeder Iteration ein `Event` verarbeitet wird. Mittels `EventScanner` kann das Event verarbeitet werden - uns interessieren dabei `MouseEvent` und eventuell `KeyPressEvent`. Mausklicks werden in der Methode `handleMouseEvent` behandelt. Die Schleife terminiert, sobald man das Spiel gewonnen oder verloren hat.
- Es ist eine Methode `drawGame` gegeben:

```
void drawGame(CodeDraw game, int[][] board, boolean[][] hints,
              int[][] attempt, int[][] placements)
```

Die Methode zeichnet das `CodeDraw`-Fenster, welches den aktuellen Spielzustand repräsentiert. Begonnen wird mit dem Schachbrett inklusive aufgedeckter Hinweise. Anschließend werden die Figuren und eventuell Markierungen für Felder, auf denen keine Figur platziert ist, gezeichnet. Anschließend wird, für den Fall, dass eine Figur gerade ausgewählt ist, ein roter Rahmen um die Figur gezeichnet. Zuletzt werden die Texte für den Punktestand

und den Informationstext generiert.

Das Schachbrett ist jeweils `OFFSET` Pixel vom linken bzw. oberen Rand des Fensters entfernt. Die Sidebar mit den Figuren ist rechtsseitig `OFFSET` Pixel vom Schachbrett entfernt und weist eine Höhe von 7 und eine Breite von 2 Quadraten aus. Die Figuren sind untereinander in der Reihenfolge, wie das Enum `Piece` definiert ist, aufgestellt. Die Figuren befinden sich ohne Abstand direkt untereinander. Die Schaltflächen „Submit“ und „Give Up“ sind untereinander mit `OFFSET` Pixel Abstand, sowie `OFFSET` Pixel Abstand unterhalb der Sidebar entfernt platziert. Der kleine Bereich über der Sidebar ist für den Punktestand reserviert. Der Bereich unterhalb des Schachbretts ist für den Informationstext reserviert.

- Es ist eine Methode `generatePlacement` gegeben:

```
int[] [] generatePlacement(int[] [] board, boolean[] [] hints)
```

Die Methode erstellt eine zufallsgesteuerte Platzierung, indem das Array `placements` befüllt wird. Dabei ist zu beachten, dass jede Figur auf legalen Koordinaten platziert ist. Zudem darf nicht mehr als eine Figur auf denselben Koordinaten platziert werden. Sobald die Platzierung für eine Figur festgelegt ist, müssen auch die Werte in den Arrays `board` und `hints` für die entsprechenden Koordinaten angepasst werden. Die Änderungen in `board`, welche die möglichen Züge betreffen, können in die Methode `simulateMovements` ausgelagert werden. Der Rückgabewert ist das 6×2 -int-Array `placements`.

- **Implementieren Sie** eine Methode `simulateMovements`:

```
void simulateMovements(Piece piece, int rank, int file, int[] [] board)
```

Die Methode ist eine Hilfsmethode für `generatePlacement`. Nachdem die Koordinaten `[rank, file]` der Figur `piece` festgelegt sind, muss nun ermittelt werden, welche Koordinaten für diese Figur von den gesetzten Koordinaten aus erreichbar sind. Es bietet sich an, mittels `switch`-Statement die Fälle für die fünf möglichen `piece` Werte abzudecken. Bei jedem erreichbaren Feld, auf dem keine Figur platziert ist, muss also der Wert um 1 in `board` erhöht werden. Die Prüfung, ob die erreichbaren Koordinaten legal sind, sowie die eventuelle Erhöhung um 1, können in die Methode `reachPosition` ausgelagert werden.

- **Implementieren Sie** eine Methode `reachPosition`:

```
void reachPosition(int rank, int file, int[] [] board)
```

Die Methode ist eine Hilfsmethode für `simulateMovements`. Anstatt vieler if-Abfragen in `simulateMovements` zur Ermittlung, ob die Koordinaten gültig sind, zu schreiben, wird diese Prüfung auf diese Methode ausgelagert. Falls die Koordinaten sich auf dem Spielfeld befinden und kein Figur darauf platziert ist, dann wird in `board` an entsprechender Stelle der Inhalt um eins erhöht. Durch die Methode `reachPosition` können die Abfragen und Bewegungen der einzelnen Figuren in `simulateMovements` stark vereinfacht werden, indem man z.B. den Turm mittels Schleifen 7 Schritte in alle vier Himmelsrichtungen gehen lässt.

- Es ist eine Methode `handleMouseClicked` gegeben:

```
boolean handleMouseClicked(int mouseX, int mouseY, int[] [] board,  
    boolean[] [] hints, int[] [] attempt, int[] [] placements)
```

Die Methode ermittelt anhand der X- und Y-Koordinaten, welche Aktion im Spiel ausgeführt wird. Wird etwa das Feld angeklickt, muss ermittelt werden, ob ein für die Platzierung vorgesehenes Feld, ein Feld mit einer Figur, ein leeres Feld, oder ein Feld mit einem Hinweis angeklickt wurde. Zudem muss beachtet werden, dass die Aktion für einen Klick auf die gleiche Art von Felder davon abhängig ist, ob gerade eine Figur ausgewählt ist. Ein Klick auf die Sidebar kann eine Figur aus- bzw. abwählen. Ein Klick auf „Submit“ löst die Methoden `isAttemptComplete` und eventuell `isAttemptCorrect` aus. Ein Klick auf „Give up“ beendet das Spiel mit einer Niederlage. Klicks auf alle anderen Bereiche haben keinen Effekt, abgesehen vom Abwählen der eventuell aktuell ausgewählten Figur. Wenn das Spiel nach der Verarbeitung der Klicks beendet ist, wird, unabhängig von Sieg oder Niederlage, `false` zurückgegeben - ansonsten `true`.

- Implementieren Sie eine Methode `isPlacement`:

```
boolean isPlacement(int rank, int file, int[] [] board)
```

Die Methode ist eine Hilfsmethode, die prüft, ob die angegebenen Koordinaten zu einem der fünf Felder gehören, auf denen Figuren zu platzieren sind. Dazu muss nur geprüft werden, ob im Array `board` auf den gegebenen Koordinaten die Zahl -1 hinterlegt ist. Ist das der Fall, dann wird `true` zurückgegeben, ansonsten `false`.

- Es ist eine Methode `placedPiece` gegeben:

```
Piece placedPiece(int rank, int file, int[] [] attempt)
```

Die Methode ist eine Hilfsmethode, welche die Figur zurückgibt, die auf den gegebenen Koordinaten platziert ist. Dazu wird geprüft, ob ein inneres Array von `attempt` mit `[rank, file]` übereinstimmt. Ist das der Fall, dann wird der Index von diesem `Piece` mittels `Piece.values()[index]` zurückgegeben. Ansonsten wird `Piece.NONE` zurückgegeben.

- Es ist eine Methode `removePiece` gegeben:

```
void removePiece(Piece piece, int[] [] attempt)
```

Die Methode ist eine Hilfsmethode, welche eine eventuell platzierte Figur von den gegebenen Koordinaten entfernt. Zu diesem Zweck wird auf dem passenden Index von `attempt`, der leicht mit `piece.ordinal()` bestimmt werden kann, das Array mit den Werten `[-1, -1]` hinterlegt.

- **Implementieren Sie** eine Methode `isAttemptComplete`:

```
boolean isAttemptComplete(int[] [] attempt)
```

Die Methode prüft, ob alle fünf Figuren platziert sind. Dazu wird geprüft, ob eine Zeile von `attempt`, abgesehen von der mit Index 0, mit den Koordinaten [-1, -1], also dem Marker für „nicht platziert“, belegt ist. Index 0 repräsentiert „keine Figur“. Sind Koordinaten nicht belegt, wird `false` zurückgegeben, ansonsten `true`. Zusätzlich informieren Sie die spielende Person über die Klassenvariable `info` mit einem Text, der darauf hinweist, falls nicht alle Figuren gesetzt sind.

- **Implementieren Sie** eine Methode `isAttemptCorrect`:

```
boolean isAttemptCorrect(int[] [] attempt, int[] [] placements)
```

Die Methode prüft, ob der Rateversuch tatsächlich der gesuchten Positionierung entspricht, indem die Zeilen der Arrays `placements` und `attempt` miteinander verglichen werden. Dazu können Sie `Arrays.equals()` verwenden. Diese Methode wird in der Methode `handleMouseClicked` ausgeführt, nachdem mit der Methode `isAttemptComplete` sichergestellt wurde, dass alle Figuren platziert wurden. Sind die Arrays `attempt` und `placements` nicht gleich, dann werden 7 Punkte abgezogen, `false` zurückgegeben und die Klassenvariable `info` mit einem entsprechenden Text belegt. Sind sie gleich, wird `true` zurückgegeben und auf den Sieg über die Variable `info` hingewiesen.

- ⓘ Sie dürfen vorhandene Methodensignaturen nicht verändern. Sie dürfen aber noch zusätzliche Hilfsmethoden schreiben, wenn Sie klar und schlüssig argumentieren können, warum diese implementiert wurden. Bei Fragen kommen Sie ins Programmiertutorium oder verwenden Sie die Diskussionsforen in TUWEL.