Aufgabenblatt 6

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Donnerstag, 12.01.2023 20:00 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 beachten Sie die Vorbedingungen! Sie dürfen sich darauf verlassen, dass alle Aufrufe die genannten Vorbedingungen erfüllen. Sie müssen diese nicht in den Methoden überprüfen.
- Bitte achten Sie auch darauf, dass Sie eine eigenständige Lösung erstellen. Wir werden bei dieser Aufgabe wieder auf Plagiate überprüfen und offensichtliche Plagiate nicht bewerten.

In diesem Aufgabenblatt werden folgende Themen behandelt:

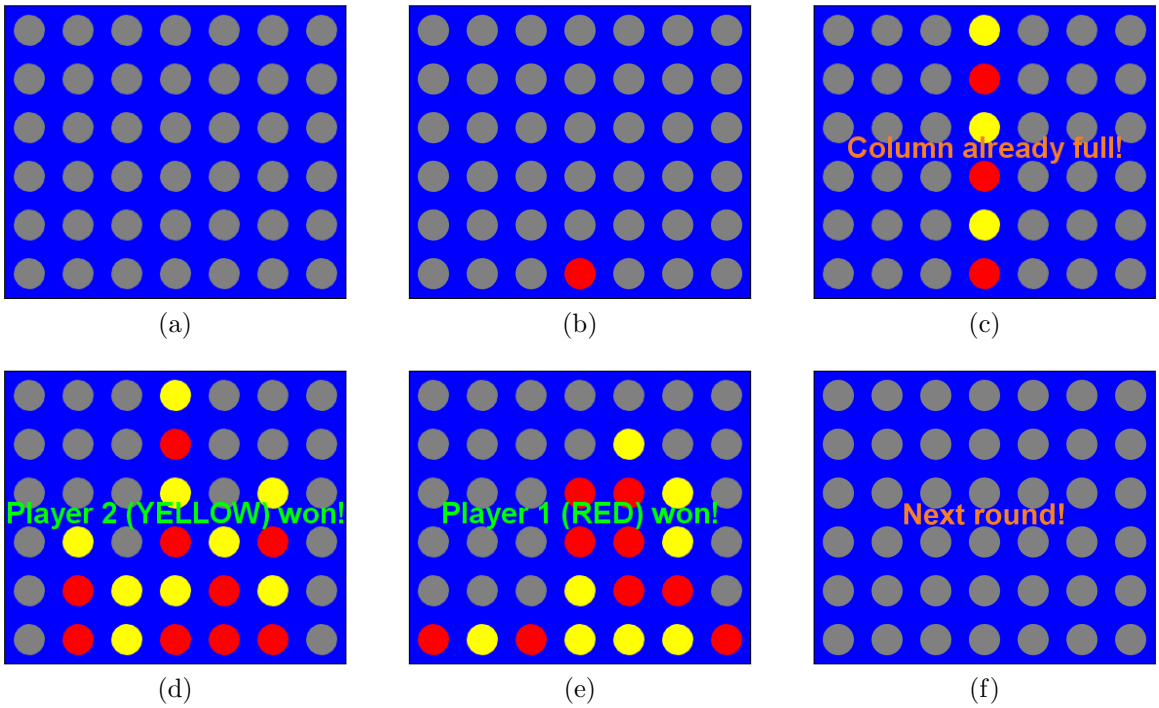
- Ein- und zweidimensionale Arrays
- Methoden
- Grafische Darstellung
- Spiellogik

Aufgabe 1 (7 Punkte)

Implementieren Sie folgende Aufgabenstellung:

- Bei dieser Aufgabe geht es um die Implementierung des Spiels *Vier Gewinnt*[1]. Dazu ist ein Teil der Spiellogik in `main` bereits vorgegeben. Erweitern Sie `main` so, dass Sie nach Implementierung der unten stehenden Methoden ein lauffähiges Spiel erhalten.

  Spielablauf:
  In `main` wird die Spiellogik implementiert. Die Größe des Spielfeldes ist angegeben und das Ausgabefenster wird angelegt. Danach wird mittels der Methode `genGameBoard` das Array für das Spielfeld erstellt, in `myGameBoard` abgespeichert und ausgegeben (Abbildung 1a). Dann beginnt Spieler_in 1 (rot) das Spiel und das Programm beginnt in einer while-Schleife zu laufen und wird erst dann beendet, sobald das CodeDraw-Fenster geschlossen wird, oder durch das Drücken der Taste `q` die Variable `gameActive` auf `false` gesetzt wird.



(a) (b) (c) (d) (e) (f)

Abbildung 1: Spielbrett von *Vier Gewinnt* mit verschiedenen Spielzuständen.

Wird nun von Spieler_in 1 mit der Maus in das Spielfeld geklickt, dann wird die Methode `isMovePossible` aufgerufen und überprüft, ob in der Spalte ein Spielzug möglich ist. Ist der Spielzug möglich (Abbildung 1b), dann retourniert die Methode `isMovePossible true`. Danach wird noch die Methode `makeMove` aufgerufen, um den möglichen Spielzug durchzuführen. Nach dem Spielzug ist die andere Spieler_in (Spieler_in 2) am Zug.

Wenn die Methode `isMovePossible false` zurückliefert, dann war die Spalte bereits voll und es wird die Nachricht `"Column already full!"` angezeigt (Abbildung 1c). Dann darf dieselbe Spieler_in nochmals einen Spielzug durchführen.

[1] `https://de.wikipedia.org/wiki/Vier_gewinnt`

Hat die Methode `isMovePossible true` zurückgegeben, muss die Methode `existsWinner` aufgerufen werden, um zu überprüfen, ob der Spielzug einer Spieler_in zu einem Sieg geführt hat. Gibt die Methode `true` zurück, dann hat die aktuell spielende Spieler_in gewonnen und es wird eine entsprechende Nachricht (Abbildung 1d und 1e) angezeigt. Nachdem die Meldung mit der Spieler_in die gewonnen hat für 3 Sekunden angezeigt wurde, wird das Spielfeld neu generiert und der Schriftzug `"Next round!"` eingeblendet (Abbildung 1f). Danach beginnt die Spieler_in, die verloren hat, mit dem ersten Spielzug ein neues Spiel.

Falls `existsWinner false` zurückliefert und das Spielfeld noch nicht voll ist, dann darf die andere Spieler_in einen Spielzug durchführen. Sollte das Spielfeld voll sein und niemand gewonnen haben, dann wird `"Board full!"` eingeblendet (Abbildung 2a). Für die Berechnung, ob das Spielfeld voll ist, können Sie die Variable `fieldsUsed` verwenden. Wenn das Spielfeld voll ist, dann wird das Spiel nicht beendet, sondern das Spielfeld Zeile für Zeile geleert. Es sollen die Inhalte des Spielfeldes immer um eine Zeile nach unten verschoben und neu gezeichnet werden (Abbildung 2b bis 2f).

Zwischen den Ausgaben verwenden Sie eine Pause von 500ms, um die Veränderungen im Spielfeld zu sehen. Nach 6 Verschiebungen ist das komplette Spielfeld leer. Danach wird der Schriftzug `"Try again!"` eingeblendet (Abbildung 2g) und das Spiel beginnt wieder von vorne mit jener Spieler_in die als nächstes an der Reihe gewesen wäre. Hat Spieler_in 1 den letzten Spielzug gemacht, dann ist jetzt Spieler_in 2 an der Reihe und umgekehrt.

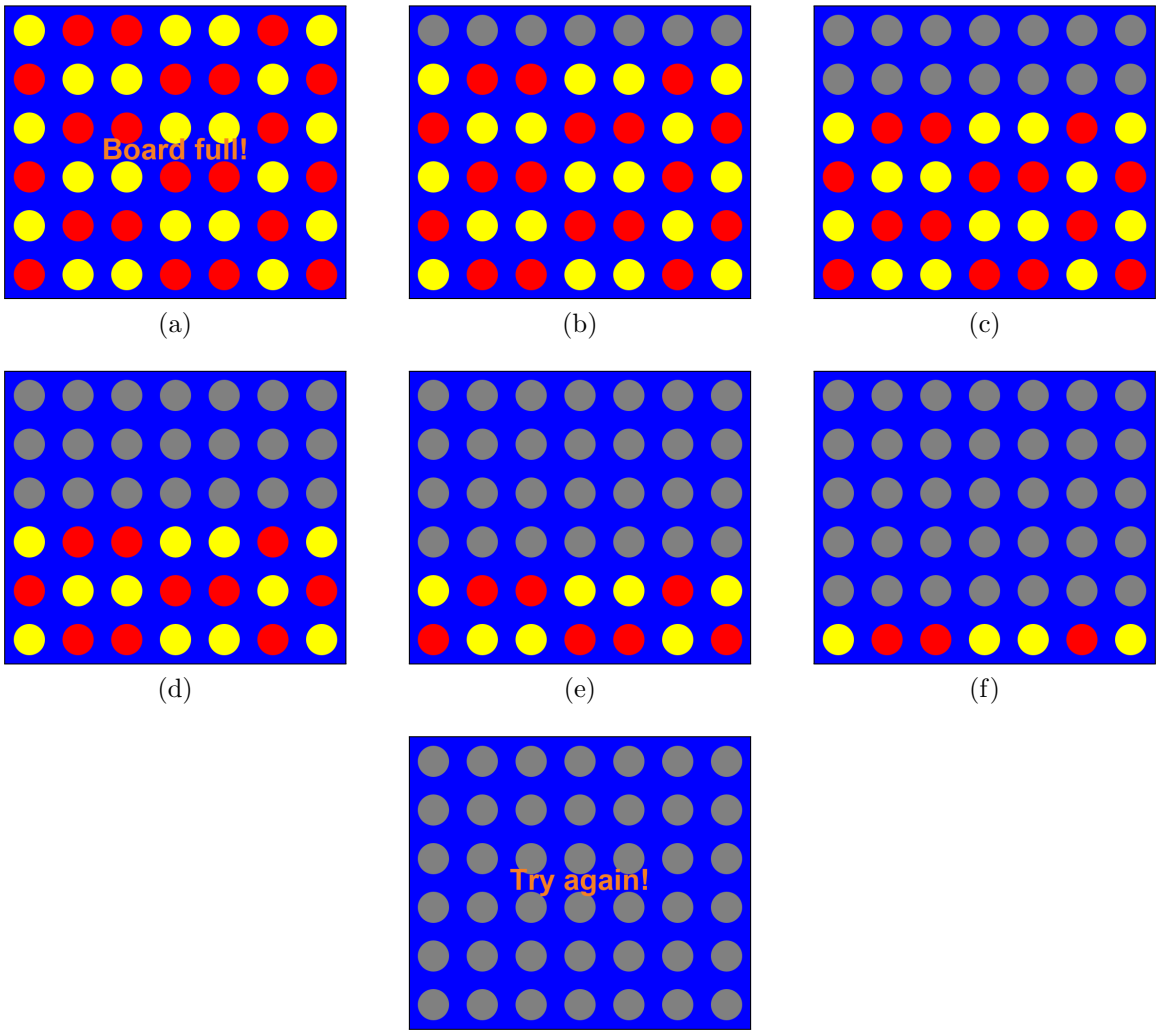
Folgende Hinweise sollen Ihnen bei der Implementierung von `main` helfen:

- Geben Sie die Spieler_in, die den Spielzug durchführen soll, mittels `System.out.println(...)` auf der Konsole aus.
- Für Textausgaben wurde der Font wie folgt formatiert: `FontSize = 28`, `FontName = "Arial"`, `TextOrigin = TextOrigin.CENTER` und `Bold = true`.
- Die gezeigten Textfarben sind `Palette.GREEN` und `Palette.ORANGE`.
- Die Schriftzüge `"Column already full!"`, `"Board full!"`, `"Try again!"` und `"Next round!"` sollen für eine Sekunde angezeigt werden, bevor diese wieder verschwinden. Verwenden Sie dafür den Befehl `myDrawObj.show(1000)`, um eine Pause von einer Sekunde zu machen.
- In den Variablen `mouseX` und `mouseY` werden die Koordinaten des letzten Mausklicks abgespeichert.
- Spieler_in 1 wird im Array `myGameBoard` mit 1 gespeichert und wird mit der Farbe *rot* gezeichnet. Spieler_in 2 wird als 2 im Array `myGameBoard` abgelegt und durch die Farbe *gelb* repräsentiert.

(!) Sie dürfen neben den unten stehenden Methoden noch zusätzliche Hilfsmethoden implementieren, wenn Sie diese als hilfreich und nötig erachten. Die Dokumentation unter dem Link `https://krassnig.github.io/CodeDrawJavaDoc/v3.0.x/codedraw/package-summary.html` kann Ihnen beim Verständnis der verschiedenen Methoden der Klasse `CodeDraw` helfen.

- Implementieren Sie eine Methode `genGameBoard`:

```
int[][] genGameBoard(int row, int col)
```



(a) (b) (c) (d) (e) (f) (g)

Abbildung 2: Sequenz zum Leeren des Spielfeldes nachdem dieses voll ist.

Diese Methode erzeugt und retourniert ein zweidimensionales int-Array mit `row` Zeilen und `col` Spalten.

Vorbedingungen: `row > 0` und `col > 0`.

- Implementieren Sie eine Methode `drawGameBoard`:

```
void drawGameBoard(CodeDraw myDrawObj, int[][] currentGameBoard, int oneSquareSize)
```

Diese Methode zeichnet den aktuellen Zustand des Spielfeldes, der im zweidimensionalen Array `currentGameBoard` (siehe Abbildung 1a) gespeichert ist. Dafür stellen Sie sich das Spielfeld als zusammengesetzte Quadrate vor. Jedes dieser Quadrate hat eine Seitenlänge von `oneSquareSize` und wird blau eingefärbt. In der Mitte dieser Quadrate werden die Kreise gezeichnet, die als Radius ein Drittel von `oneSquareSize` haben.

Hinweis: Sie können dazu das komplette Spielfeld zunächst blau färben und die Größe `oneSquareSize` als Information zum Zeichnen der Kreise verwenden. Die Kreise werden

grau gezeichnet, falls die Position im Spielfeld leer ist. Für Spieler_in 1 werden rote Kreise und für Spieler_in 2 werden gelbe Kreise gezeichnet (siehe Abbildung 1b).

Vorbedingungen: `currentGameBoard != null` und `currentGameBoard.length > 0`, dann gilt auch für alle gültigen `i`, dass `currentGameBoard[i].length > 0`. `oneSquareSize > 0`.

- Implementieren Sie eine Methode `isMovePossible`:

```
boolean isMovePossible(int[][] currentGameBoard, int col)
```

Diese Methode wird nach einem Klick einer Spieler_in in `main` aufgerufen und überprüft, ob in der gewünschten (geklickten) Spalte im Spielfeld ein Spielzug möglich ist. Ein weiterer Parameter `col` gibt die Spalte an, für die der Spielzug durchgeführt werden soll. Die Methode prüft, ob in der entsprechenden Spalte noch ein freier Platz vorhanden ist. Wenn ja, dann wird `true` zurückgegeben, ansonsten wird `false` zurückgegeben.

Vorbedingungen: `currentGameBoard != null` und `currentGameBoard.length > 0`, dann gilt auch für alle gültigen `i`, dass `currentGameBoard[i].length > 0`. `col >= 0` und `col < currentGameBoard[i].length`.

- Implementieren Sie eine Methode `makeMove`:

```
void makeMove(int[][] currentGameBoard, int player, int col)
```

Diese Methode führt für die Spieler_in `player` in der Spalte `col` einen Spielzug durch. Die Methode prüft die entsprechende Spalte von unten nach oben, wo der nächste freie Platz (Arrayeintrag gleich 0) vorhanden ist. An dieser Stelle wird dann je nach Spieler_in der entsprechende Eintrag im Array `currentGameBoard` geändert.

Vorbedingungen: `currentGameBoard != null` und `currentGameBoard.length > 0`, dann gilt auch für alle gültigen `i`, dass `currentGameBoard[i].length > 0`. `player > 0` und `player < 3`. `col >= 0` und `col < currentGameBoard[i].length`.

- Implementieren Sie eine Methode `existsWinner`:

```
boolean existsWinner(int[][] currentGameBoard, int player)
```

Die Methode überprüft nach einem Spielzug mit `makeMove`, ob der neu hinzugefügte Spielstein zu einem Sieg führt. Dazu bekommt die Methode das aktuelle Spielfeld `currentGameBoard` und die Spieler_in `player`, die den letzten Spielzug durchgeführt hat, übergeben. Die Methode überprüft nun, ob diese Spieler_in vier Spielfelder in einer Zeile, Spalte oder Diagonale nebeneinander belegt hat. Wenn ja, dann gibt die Methode `true` zurück, ansonsten `false`.

Vorbedingungen: `currentGameBoard != null` und `currentGameBoard.length > 0`, dann gilt auch für alle gültigen `i`, dass `currentGameBoard[i].length > 0`. `player > 0` und `player < 3`.