Aufgabenblatt 6

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Mittwoch, 03.01.2024 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 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 soll das Spiel *Sokoban*[1] implementiert werden. Einige Programmteile und Methoden sind bereits vorgegeben, andere Programmteile müssen Sie fertigstellen, damit Sie am Ende ein funktionierendes Spiel erhalten. Bei diesem Spiel muss in einem Labyrinth versucht werden, mit einer Spielfigur herumliegende Kisten an ihren finalen Bestimmungsort zu verschieben. Die Figur kann dabei aber nur immer eine Kiste gleichzeitig verschieben und kann Kisten, die sich in einer Ecke befinden, nicht mehr bewegen. In Abbildung 1 sehen Sie ein komplettes Sokoban-Level mit allen Spielelementen.

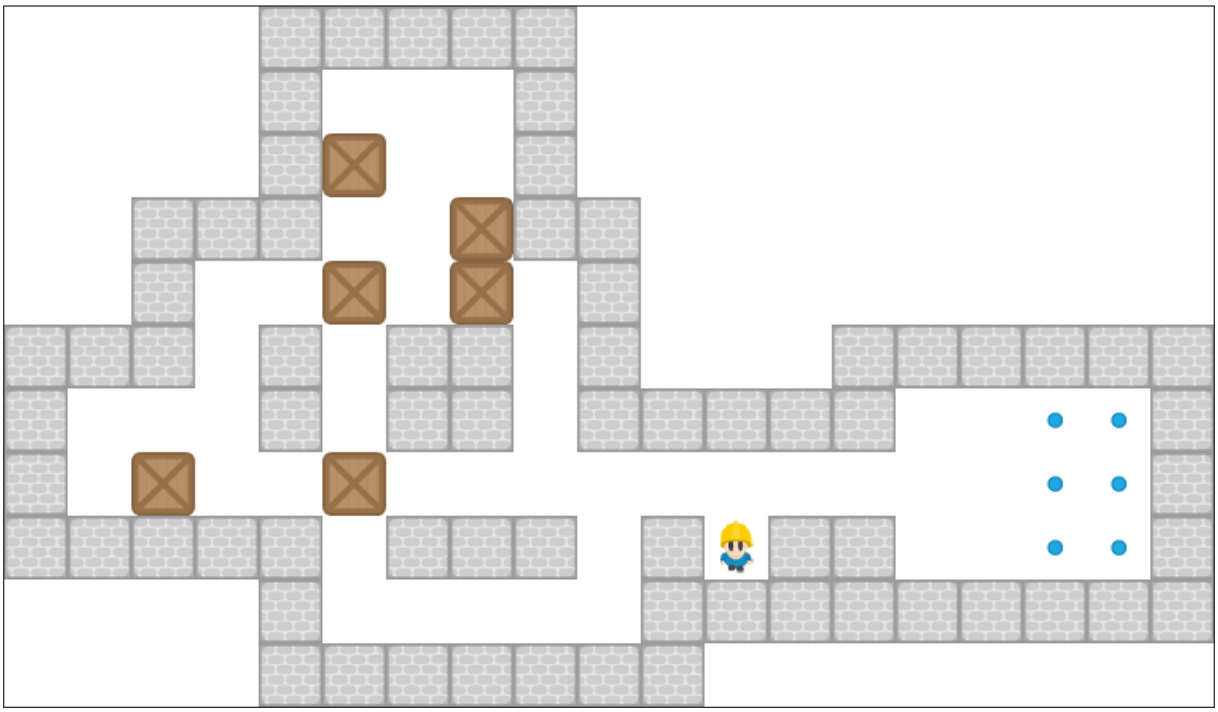


Abbildung 1: Ein *Sokoban*-Level mit allen Spielelementen.

- Sie haben die Methode `main` bereits vorgegeben. In der Methode `main` sind bereits Datenstrukturen vorhanden, die für das Spiel verwendet werden.

  Beschreibung der Datenstrukturen in `main`:

  - `allLevels`: Dieses Array beinhaltet alle Levels, nachdem diese von der Datei `sokoban_levels.csv` eingelesen wurden. Jedes Level wird hier als String abgelegt. Das heißt, dass das zweidimensionale Level hier als lange Stringzeichenkette abgelegt ist und ein Arrayeintrag einem ganzen Level entspricht.
  - `level`: Dieses zweidimensionale `char`-Array beinhaltet das aktuelle Level. Jedes Level ist immer von Wänden (ohne Löcher und ohne Ausgänge) umgeben. Die unterschiedlichen Zeichen eines Levels sind:

[1] `https://en.wikipedia.org/wiki/Sokoban`

  - `' '` ... repräsentiert ein leeres Feld.
  - `'#'` ... repräsentiert eine Wand.
  - `'.'` ... repräsentiert einen Zielpunkt.
  - `'$'` ... repräsentiert eine Kiste.
  - `'@'` ... repräsentiert die Spielfigur.

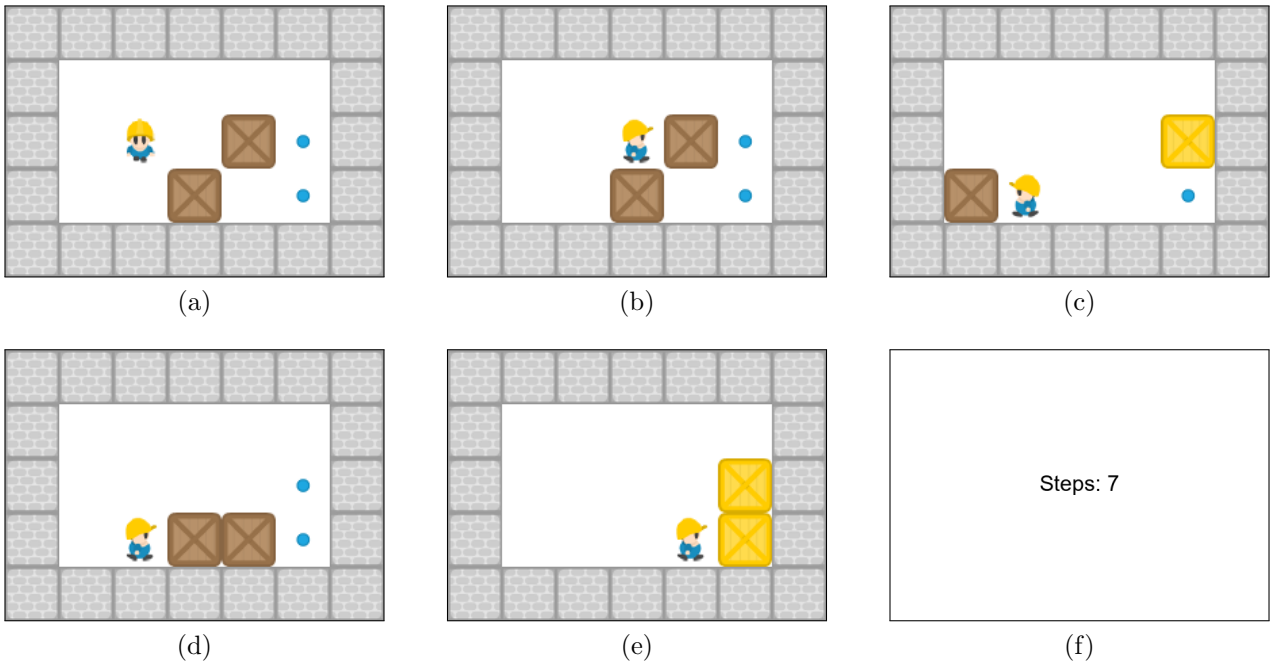
  Das Array `level` ist so zu verwenden, sodass die erste Dimension der y-Koordinate und die zweite Dimension der x-Koordinate entspricht. Zum Beispiel wird der Punkt $P(x, y)$ (z.B. P(2,5)) im Array in der Form `A[y][x]` (z.B. `A[5][2]`) angesprochen.

- `goals`: Dieses Array dient zur Speicherung aller Zielpunkte, wo die Kisten hingeschoben werden müssen. In der Methode `newLevel(...)` wird das Array `goals` mit den Zielpositionen befüllt. Jede Zielposition wird mit ihrer x- und y-Koordinate abgespeichert. Das heißt, dass jede Zeile im Array `goals` die Länge zwei hat. In der ersten Spalte sind die x-Werte und in der zweiten Spalte sind die y-Werte abgelegt. Diese separate Speicherung ist nötig, da während des Spielverlaufes Kisten die Zielposition überdecken können und somit diese Information verloren gehen würde. Das heißt, dass nach dem Einlesen des Levels die Zielpunkte `'.'` im Array `level` nicht weiter vorhanden sind.

Spielablauf: In der Methode `main` wird in einer Schleife das Spiel solange gespielt, bis das Spiel gewonnen (alle Levels gelöst) wurde. Das Spiel beginnt mit einem Level in der Ausgangsposition (siehe Abbildung 2a) und mit dem Wert 0 für die Variable `moveDirection` als neutrale Richtung und keine Bewegung. Danach wartet das Programm in der Schleife, bis eine Taste gedrückt wird. Dafür wird mit der Methode `hasKeyDownEvent()` überprüft, ob eine Taste gedrückt wurde. Danach wird mit `nextKeyDownEvent()` die gedrückte Taste ausgelesen und auf diverse Tastenkonstanten abgefragt, die in der Enumeration `Key` hinterlegt sind. Ist die jeweilige Taste gedrückt worden, dann wird `true` zurückgegeben. Das Spiel wird über die Pfeiltasten Oben, Unten, Links und Rechts gesteuert. Erfolgt ein Tastendruck auf eine der Pfeiltasten, dann wird überprüft, ob es möglich ist, die Spielfigur in diese Richtung zu bewegen. Ist dies möglich, bewegt sich die Spielfigur auf das entsprechende Feld und das aktualisierte Spielfeld wird gezeichnet (siehe Abbildung 2b). Abhängig von der Richtung wird die Figur anders gezeichnet. Ist dies nicht möglich, bleibt das Spielfeld unverändert und die Spielfigur ändert ihre Blickrichtung. Danach wird überprüft, ob sich alle Kisten auf den Zielpositionen (Kiste auf Zielpostion wird anders eingefärbt) befinden. Sollte dieser Zustand eintreffen (siehe Abbildung 2e), dann ist das aktuelle Level gelöst und es wird noch die Anzahl der benötigten Schritte angezeigt (Anzeige bereits implementiert), die zum Lösen dieses Levels verwendet wurden (siehe Abbildung 2f). Danach wird das nächste Level geladen bzw. das Spiel beendet, falls es das letzte Level gewesen ist. War es das letzte Level, dann werden alle benötigten Schritte des gesamten Spiels angezeigt (Anzeige bereits implementiert).

Während des Spiels können Spielzustände eintreten, die es unmöglich machen, das Level fortzusetzen und einen Neustart erfordern. Dies passiert beispielsweise, wenn eine Kiste in eine Ecke geschoben wird (siehe Abbildung 2c), oder zwei Kisten hintereinander an einer Wand liegen (siehe Abbildung 2d) und somit nicht weiter geschoben werden können.

Im Spiel wurden drei zusätzliche Tasten eingebaut, um hier bei der Steuerung zu helfen. Mit der Taste `r` kann ein Level zurückgesetzt und nochmals gestartet werden. Mit der Taste `t` haben Sie die Möglichkeit, Levels zu überspringen. Diese Taste dient zu Testzwecken, um Levels schneller probieren zu können. Die Taste `q` wird noch angeboten, um das Spiel zu beenden.



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

Abbildung 2: Verschiedene Spielzustände des Spiels Sokoban.

Die folgenden vier Methoden sind bereits vorgegeben und werden hier nochmals kurz beschrieben:

- `readLevels`: Diese Methode liest alle Levels von der Datei `sokoban_levels.csv` ein. Der erste Wert in der Datei beschreibt, wie viele Levels in der Datei zu finden sind. Danach wird jedes Level als einzelner String in ein Array abgelegt. Da die Levels aus mehreren Zeilen bestehen, wird der Umbruch einer Zeile mit `'\n'` innerhalb des Strings codiert.
- `newLevel`: Diese Hilfsmethode erzeugt aus einem Level, das in einem String codiert ist, ein zweidimensionales `char`-Array.
- `adjacentPosition`: Diese Hilfsmethode gibt aufgrund der aktuellen Position und der Richtung die so neu entstandene Position in Form von Koordinaten (x,y) zurück. Wenn z.B. die aktuelle Position (4,3) ist und die Richtung 4 (rechts), dann ist die neue Position (5,3).
- `showText`: Diese Hilfsmethode dient dazu, einen Text im CodeDraw-Fenster auszugeben.

Zusätzlich werden noch sechs weitere Methoden benötigt, die Sie für die Komplettierung des Spiels implementieren müssen.

- Implementieren Sie eine Methode `numberOfGoals`:

```
int numberOfGoals(String levelString)
```

Diese Methode zählt die Anzahl der Zielpositionen innerhalb eines Levels `levelString`. Das ganze Level steht hier in einem String und es werden alle Vorkommen von `'.'` gezählt und zurückgegeben.

Vorbedingungen: `levelString != null` und `levelString.length > 0`.

- Implementieren Sie eine Methode `figurePosition`:

  ```
  int[] figurePosition(char[][] level)
  ```

  Diese Methode sucht innerhalb des zweidimensionalen Arrays `level` die aktuelle Position der Spielfigur. Die Koordinaten, bei der die Spielfigur `'@'` gefunden wurde, werden in der Form von `new int[]{x,y}` (x (Spalte) und y (Zeile) stehen für die Koordinaten im Array `level`) zurückgegeben.

  Vorbedingung: `level != null` und `level[i] != null` für alle gültigen Indizes `i`. Das Symbol kommt in `level` genau einmal vor.

- Implementieren Sie eine Methode `move`:

  ```
  boolean move(char[][] level, int direction)
  ```

  Diese Methode bewegt die Spielfigur `'@'` innerhalb des Spielfeldes `level` in die angegebene Richtung `direction` ($1 \mathrel{\widehat{=}}$ oben, $2 \mathrel{\widehat{=}}$ unten, $3 \mathrel{\widehat{=}}$ links, $4 \mathrel{\widehat{=}}$ rechts). Dazu muss überprüft werden, ob die Figur bei ihrer Bewegung auf ein Hindernis stößt. Wenn das Feld leer ist (`' '`), dann wird die Figur auf die neue Position gesetzt. Befindet sich auf der neuen Position eine Wand (`'#'`), dann wird die Spielfigur auf der aktuellen Position belassen. Sollte sich auf der neuen Position eine Kiste (`'$'`) befinden, dann muss überprüft werden, ob die Kiste verschoben werden kann. Dazu wird überprüft, ob sich hinter der Kiste eine weitere Kiste bzw. eine Wand befindet. Nur wenn hinter der Kiste ein leeres Feld oder ein Zielpunkt ist, dann werden die Kiste und die Spielfigur verschoben. Die Spielfigur kann auch ein Feld mit einem Zielpunkt betreten und auch die Kiste auf einem Zielfeld kann entsprechend der Regeln weiter verschoben werden. Ist die Spielfigur verschoben worden, dann wird `true` zurückgegeben, ansonsten `false`. Hinweis: Sie können für die Implementierung die Methode `adjacentPosition(...)` verwenden.

  Vorbedingungen: `level != null`, `level[i] != null` für alle gültigen Indizes `i`, `direction > 0` und `direction < 5`.

- Implementieren Sie eine Methode `boxPositions`:

  ```
  int[][] boxPositions(char[][] level, int numberOfBoxes)
  ```

  Diese Methode sucht die aktuellen Positionen aller Kisten `'$'` innerhalb des Spielfeldes `level` und gibt deren Koordinaten in einem zweidimensionalen Array zurück. Das Ergebnisarray hat `numberOfBoxes` Zeilen und in jeder Zeile stehen die Koordinaten einer Kiste in der Form `new int[]{x,y}` (x (Spalte) und y (Zeile) stehen für die Koordinaten im Array `level`). Das heißt, dass jede Zeile im Ergebnisarray die Länge zwei hat. In der ersten Spalte sind die x-Werte und in der zweiten Spalte sind die y-Werte abgelegt.

  Vorbedingungen: `level != null`, `level[i] != null` für alle gültigen Indizes `i` und `numberOfBoxes > 0`.

- Implementieren Sie eine Methode `won`:

  ```
  boolean won(char[][] level, int[][] goals)
  ```

  Diese Methode überprüft, ob das aktuelle Level bereits gelöst wurde. Dazu werden alle Kistenpositionen in `level` mit den verfügbaren Zielpositionen in `goals` verglichen. Wenn sich alle Kisten auf einer Zielposition befinden, dann wurde das Level gelöst und es wird `true` zurückgegeben, ansonsten wird `false` retourniert. Es kann angenommen werden, dass es immer eine gleiche Anzahl von Kisten und Zielpositionen gibt.

  Vorbedingungen: `level != null`, `level[i] != null` für alle gültigen Indizes `i`, `goals != null`, `goals[i] != null` für alle gültigen Indizes `i` und `goals[i].length == 2`.

- Implementieren Sie eine Methode `drawGame`:

  ```
  void drawGame(CodeDraw myDrawObj, char[][] level,
          int[][] goals, int direction)
  ```

  Die Methode zeichnet den aktuellen Zustand eines Levels in ein CodeDraw-Ausgabefenster. Das Fenster ist bereits vordefiniert und hat die Größe für das aktuelle Level, sodass jeder Eintrag des Arrays `level` eine Größe von `SQUARE_SIZE`×`SQUARE_SIZE` Pixel hat. Die erste Zeile des Arrays `level` entspricht der obersten Zeile des CodeDraw-Fensters. Im Array `level` finden Sie leere Felder, Wände, Kisten (Kisten werden unterschiedlich gezeichnet, je nachdem ob diese auf einem Zielpunkt liegen oder nicht) oder die Spielfigur (Figur wird noch abhängig von der Richtung `direction` unterschiedlich gezeichnet). Im Array `goals` sind die Zielpunkte der Kisten gespeichert und werden ebenfalls zum Zeichnen des Levels verwendet. Die vorhandenen Elemente in einem Sokoban-Level werden mit Grafiken[2] (Sprites[3]) wie folgt dargestellt:

  1. In `goals[][]` sind die Zielpunkte der Kisten gespeichert und werden mit dem Icon in Abbildung 3b gezeichnet.
  2. In `level[][]` werden leere Felder durch ein `' '` dargestellt und als weißes Quadrat gezeichnet.
  3. In `level[][]` werden Wände durch ein `'#'` dargestellt und mit dem Icon in Abbildung 3a gezeichnet.
  4. In `level[][]` werden Kisten durch ein `'$'` dargestellt und abhängig davon ob diese auf einem Zielpunkt liegt unterschiedlich gezeichnet. In Abbildung 3c wird die Kiste gezeigt, die generell gezeichnet wird. Liegt aber eine Kiste auf einem Zielpunkt, dann wird die in Abbildung 3d gezeigte Darstellung für eine Kiste verwendet.
  5. In `level[][]` wird die Spielfigur durch ein `'@'` dargestellt und abhängig von der Richtung, die zuletzt gedrückt wurde, unterschiedlich gezeichnet. Für die Richtung 0 (Initialrichtung) und 2 (unten) wird die Figur in Abbildung 3e verwendet. Die Richtung 1 (oben) wird mit Abbildung 3f, Richtung 3 (links) mit Abbildung 3g und Richtung 4 (rechts) mit Abbildung 3h dargestellt.

[2] Grafiken von *1001com* auf `https://opengameart.org/content/sokoban-pack`
[3] `https://en.wikipedia.org/wiki/Sprite_(computer_graphics)`

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

Abbildung 3: Verschiedene Grafiken (Sprites) für die optische Aufbereitung.

Die Pfadangaben inklusive Dateinamen der Abbildungen sind wie folgt:

- Abbildung 3a: `"src/wall.png"`;
- Abbildung 3b: `"src/endpoint.png"`;
- Abbildung 3c: `"src/box.png"`;
- Abbildung 3d: `"src/box_goal.png"`;
- Abbildung 3e: `"src/figure_down.png"`;
- Abbildung 3f: `"src/figure_up.png"`;
- Abbildung 3g: `"src/figure_left.png"`;
- Abbildung 3h: `"src/figure_right.png"`;

Hinweis: Mit der Codezeile `Image imgEndpoint = Image.fromFile("src/endpoint.png");` kann zum Beispiel das Bild eines Zielpunktes eingelesen werden. Verwenden Sie danach die CodeDraw-Methode `drawImage(...)`, um das eingelesene Bild im CodeDraw-Fenster zu zeichnen. Auch die Skalierung auf die Größe von `SQUARE_SIZE`×`SQUARE_SIZE` Pixel kann mit der Methode `drawImage(...)` durchgeführt werden.

Vorbedingungen: `myDrawObj != null`, `level != null`, `level[i] != null` für alle gültigen Indizes `i`, `goals != null` und `goals[i] != null` für alle gültigen Indizes `i`. Der Wert von `direction` ist im Intervall [1, 4].