MINILABOR BFO
The maze game

Erster Teil - Anzeigen und aus dem Labyrinth herauskommen

In diesem ersten Teil werden Sie den Quellcode, den wir für Sie vorbereitet haben, sowie die Struktur des Java-Codes kennen lernen. Am Ende dieses Teils werden Sie über die nötigen Mittel verfügen, um ein Labyrinth als Text und Bild darzustellen. Sie werden auch in der Lage sein, den Weg von einem beliebigen Punkt zum Ausgang im Bild darzustellen.

Aufgabe 0 - Importieren des Projekts

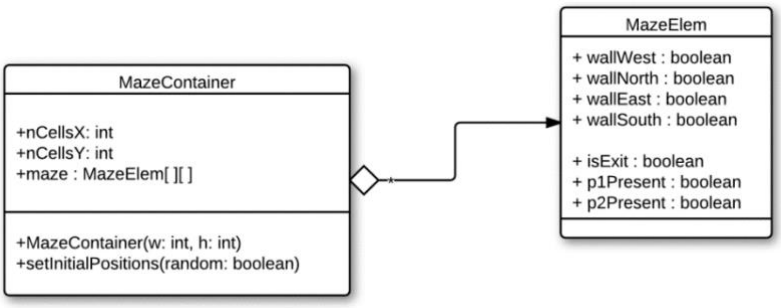
Wir haben für Sie ein Eclipse-Projekt vorbereitet, das die grundlegenden Elemente des Projekts enthält. Um es zu importieren führen Sie folgende Schritte aus:

1. Starten Sie Eclipse und führen Sie dann «*File->Import->General-> Projects from Folder or Archive*» aus.
2. Klicken Sie auf die Option «*Archive*», wählen Sie die Datei *maze_studentversion.zip* aus dem USB-Key und klicken Sie auf «*Finish*». Das *Maze-Projekt* existiert nun in Ihrer Projektliste.

Aufgabe 1 - Zeichnen des Labyrinths in Textform

Diese erste Aufgabe wird Ihnen zeigen, wie das Labyrinth im Projekt codiert ist, indem Sie es als Text in der Konsole anzeigen sollen. Dies wird Ihnen den ganzen Tag über nützlich sein, da wir immer wieder die gleiche Datenstruktur verwenden werden.

1. Öffnen Sie die Datei `TextDisplay.java`, die sich in `maze/display` befindet, und betrachten Sie die Datei.
2. Beachten Sie, dass es in der Datei eine `main` gibt. Wie in C/C++ stellt `main` einen Einstiegspunkt in das Programm dar.
3. In dem `main` wird ein `MazeContainer` erstellt. Diese Klasse stellt das Labyrinth her und ermöglicht Ihnen den Zugriff darauf in Form eines zweidimensionalen Arrays aus `MazeElem`. In der Klasse befinden sich auch die Informationen über die Dimensionen des Labyrinths (Felder `nCellsX`, `nCellsY`).
4. Öffnen Sie die Klasse `MazeElem` und beobachten Sie sie. Diese letzte Klasse enthält bereits alle Informationen darüber, was sich in einem Feld des Labyrinths befindet (Wände, Spieler, Ausgang ...).
5. Das UML-Klassendiagramm, d. h. wie die Klassen organisiert sind, sieht folgendermassen aus:



6. Starten Sie das entsprechende Programm (Rechtsklick auf die Klasse *TextDisplay→Run As→Java Application*).
7. Ändern Sie das Programm so, dass ein Labyrinth der Grösse 5x5 angezeigt wird.
8. Wie Sie sehen, wird das Labyrinth nicht vollständig angezeigt, sondern nur die Kreuzungen und die Wände ganz unten und ganz rechts. Ergänzen Sie den Code an den mit *TODO Aufgabe 1* bezeichneten Stellen, damit das Labyrinth korrekt wie unten angezeigt wird (`p1` steht für die Position des ersten Spielers und `e` für den Ausgang). Ihre Anzeige sollte genau :

```
*---*---*---*---*---*
| p1|                 |
*   *   *---*---*   *
|   |   |       |   |
*   *---*   *   *   *
|           |       |
*---*---*---*---*   *
|       |           |
*   *---*   *---*---*
|         e         |
*---*---*---*---*---*
```

Aufgabe 2 - Zeichnen des Labyrinths als Bild

Nachdem wir nun in der Lage sind, ein Labyrinth korrekt als Text anzuzeigen, ist es an der Zeit, unserem Programm ein wenig Grafik hinzuzufügen. Eine Klasse, die ähnlich wie `TextDisplay` funktioniert, finden Sie in der Datei `GraphicDisplay`.

1. Öffnen Sie die Datei `GraphicDisplay`.
2. Beobachten Sie die Methode `main` dieser Klasse und führen Sie sie aus.
3. Ändern Sie den Code so, dass ein 10x10-Labyrinth angezeigt wird.
4. Sie können die Grösse jedes kleinen Quadrats ändern, indem Sie einen Parameter im Konstruktor von `GraphicDisplay` verändern. Versuchen Sie es mit anderen grösseren und kleineren Werten.
5. Es ist möglich, das Labyrinth ohne die Fensterränder anzuzeigen. Finden Sie heraus, wie das geht, und führen Sie Ihr Programm aus. Sie werden etwas Ähnliches erhalten wie das, was rechts in der nebenstehenden Abbildung zu sehen ist. Sie können entscheiden, ob Sie eher diese oder die andere Methode verwenden möchten, je nachdem, was Ihnen lieber ist. Versuchen Sie auch, das Hintergrundgitter sichtbar zu machen (eine Variable, die Sie ändern können).

6. Um sicherzustellen, dass die Anzeige korrekt ist, lassen Sie sich auch eine Textversion desselben Labyrinths aus dem `main` anzeigen.

Aufgabe 3 - Lees Propagierungsalgorithmus (A*)

In dieser Phase werden Sie eine Möglichkeit implementieren, den Ausgang des Labyrinths mithilfe eines Routing-Algorithmus automatisch zu finden.

1. Öffnen Sie die Klasse `AStar.java`, die das Skelett der Solverklasse für das Labyrinth nach der Methode von Lee enthält, die am frühen Morgen vorgestellt wurde. Die Ausbreitungsphase des Algorithmus (wenn der Pfad vom Startpunkt aus in alle Richtungen wächst) kann formal wie folgt beschrieben werden:

> Algorithmus 1 - Propagation von Lee (Algorithmus A*)
>
> Markieren Sie den Startpunkt mit 1
> m ← 1
> // Wellenförmige Ausbreitung
> **wiederholen**
> Markiert alle nicht markierten Nachbars Zellen vom den markierten Punkten m mit m+1
> m ← m+1
>
> **bis** zum gefundenen Ziel **oder** es gibt keine weiteren Punkte die markiert werden können

1. Um Ihre Methode zu testen, wurde hier auch ein `main` erstellt. Sie können es verwenden, um Ihren Code leichter zu debuggen.
2. Beobachten Sie die Methode `solve`. Sie beginnt damit, eine leere Lösung zu erstellen (die eigentlich ein Array von Ganzzahlen ist). Sie fährt fort, indem sie Lees Propagation und dann das Backtracking durchführt. Das Ziel des gesamten Algorithmus ist es, überall dort eine 1 zu setzen, wo sich der Lösungsweg zwischen dem als Argument übergebenen Punkt und der Ausgabe befindet. Zum Beispiel das Labyrinth:

```
*---*---*---*---*
| p1|           |
*   *   *---*   *
|   |   |       |
*   *---*   *   *
|           |   |
*---*---*---*   *
|     e         |
*---*---*---*---*
```

wird am Ende ergeben

```
1 - 0 - 0 - 0
1 - 0 - 1 - 1
1 - 1 - 1 - 1
0 - 1 - 1 - 1
```

3. Wenn wir das Backtracking entfernen (indem wir die Methode `backtrace()` in `solve` auskommentieren), können wir die Schritte der Ausbreitung sehen, was sehr praktisch ist, um den Code zu debuggen. Im obigen Fall sieht das dann so aus:

```
01 - 10 - 09 - 08
02 - 11 - 06 - 07
03 - 04 - 05 - 08
00 - 11 - 10 - 09
```

4. Um dieses Ergebnis erzielen zu können, müssen Sie die Methode `expansion` implementieren. Diese wird von `solve` aufgerufen, solange sie nicht *true* zurückgibt (d. h. solange die Ausbreitung der Welle das Ziel noch nicht erreicht hat). Beachten Sie auch, dass die Methode bei jedem Aufruf den Status von *m* erhält, der die Nummer des aktuellen Schritts ist. Es liegt an Ihnen, herauszufinden, wie Sie das machen (mit unserer Hilfe)! Zögern Sie nicht, die Methode `access_solution()` zu verwenden (indem Sie verstehen, wozu sie dient...).

Aufgabe 4 - Anzeige der Lösung im Grafikfenster

Ihre Implementierung des vorherigen Punktes liefert Ihnen eine Tabelle, die eine Lösung ab einem bestimmten Punkt enthält. Sie haben die Möglichkeit, diese direkt in der grafischen Darstellung des Labyrinths anzuzeigen. Dies werden Sie in dieser Aufgabe umsetzen.

1. Lassen Sie sich von dem bisherigen Vorgehen inspirieren und zeigen Sie vom `main` Klasse `AStar` die grafische Version des Labyrinths an.
2. Das Objekt der Klasse `GraphicDisplay`, das Sie erstellt haben, verfügt über eine Methode `setSolution`, die als Argument ein Ganzzahl-Array entgegennimmt, das einer Lösung von einem Punkt aus entspricht. Verwenden Sie diese Methode, um die Lösung anzuzeigen, die mit dem im vorherigen Schritt erstellten Algorithmus A* berechnet wurde. Wenn Sie die Lösung aus der Anzeige entfernen möchten, verwenden Sie die Methode `clearSolution();`

Optionale Aufgaben

1. Ändern Sie die Art und Weise, wie der Spieler gezeichnet wird (andere Form wie ein Pfeil, der den letzten Zug zeigt, Farben, ...).

2. Zeigen Sie eine andere Zeichnung für den Ausgang an (z. B. einen Pfeil, der nach aussen zeigt).

Teil 2 - Spielerbewegung und Spiellogik

Sie haben nun ein Labyrinth, das Sie anzeigen können und aus dem Sie einen Ausweg finden können. Das Ziel dieses zweiten Teils wird es sein, die Bewegung einer Spielfigur durch das Labyrinth sowie die Anzeige einer Nachricht, wenn der Ausgang erreicht ist, zu realisieren.

Aufgabe5 - Bewegung des Spielers

Die Spiellogik für die Anzeige von Meldungen, wenn der Ausgang gefunden wurde, befindet sich in der Klasse `MazeGame`. Die Idee ist relativ einfach: Der Spieler kann seine Spielfigur mithilfe der Tastaturtasten bewegen. Wenn eine Bewegung zum Ausgang führt (das ist die gleiche Klasse), hat der Spieler gewonnen und ein neues Spiel beginnt.

1. Machen Sie sich mit der Klasse `MazeGame` vertraut.
2. Vervollständigen Sie die Methode `findPlayer(Player p)`, die die Koordinaten eines Spielers auf dem Spielfeld zurückgibt. Die Koordinaten werden in Form eines `Punktes` zurückgegeben. Wenn der Spieler nicht im Spiel ist, wird *null* zurückgegeben, was ein spezieller Wert ist, der "nichts"[1] bedeutet.
3. Ergänzen Sie die Methode `movePlayer(Richtung d)`, die die Spielfigur in die vorgegebene Richtung bewegt. Beachten Sie, dass Sie die Wände korrekt berücksichtigen müssen. Um diese Methode zu debuggen, verwenden Sie sie direkt im `main`, um zu sehen, ob alles so läuft, wie Sie es sich vorstellen. Wenn Sie denken, dass Ihre Methode funktioniert, versuchen Sie, Ihre Spielfigur in verschiedene Richtungen zu bewegen.
4. Probieren Sie den Code, den Sie geschrieben haben, mit Labyrinthen aus, die nicht quadratisch sind. Warum kann genau dieser Punkt potenziell Probleme verursachen?

Aufgabe 6 - Tastaturverwaltung

Ein paar Worte zur Theorie - Umkehrung des Kontrollflusses

Da Java sehr stark objektorientiert ist, ist es möglich, die einzelnen Schritte klar zu trennen. So findet die Verwaltung der Tasten in der Klasse `KeyboardListener` statt. In dieser Klasse *wird* der *Kontrollfluss umgekehrt*. Das bedeutet, dass nicht Sie entscheiden, wann eine Methode aufgerufen wird, sondern die virtuelle Maschine (ähnlich wie bei Interrupts). Bei einer grafischen Benutzeroberfläche weiss man beispielsweise nie, wann der Benutzer einen Knopf oder eine Taste drücken wird. Anstatt *aktiv zu warten*, ermöglicht diese Art der Kontrollumkehr, dass man reagieren kann, wenn es nötig ist.

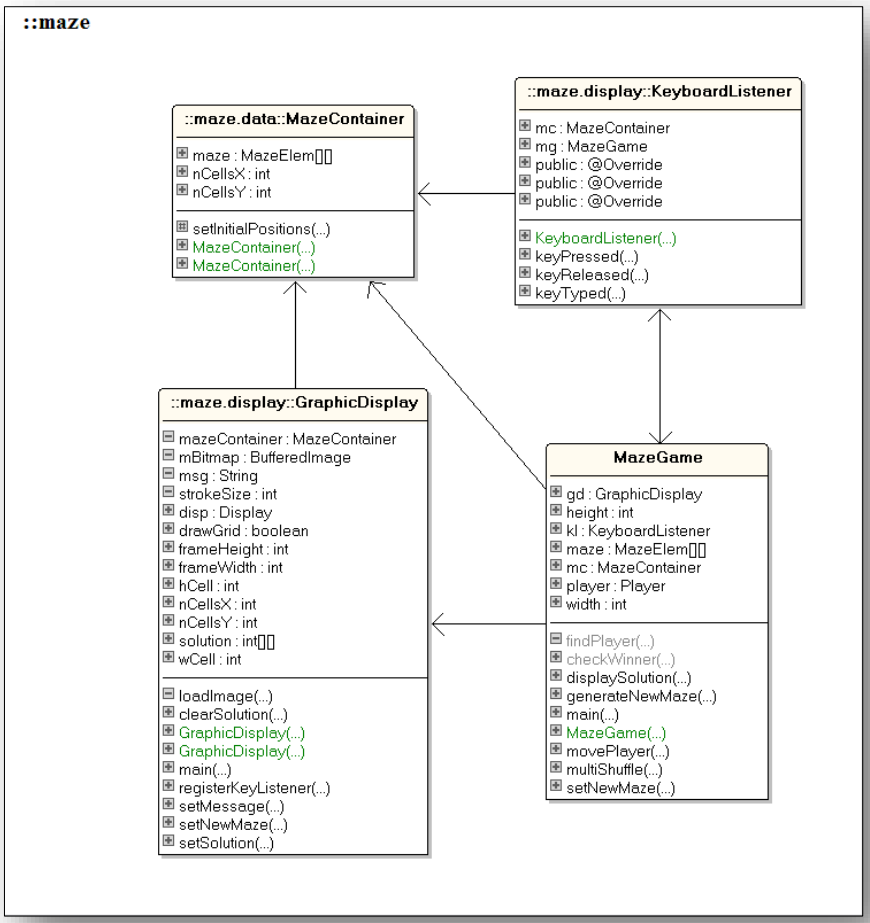
Konkret bedeutet dies, dass Methoden von der Java Virtual Machine aufgerufen werden, wenn der Benutzer eine Taste gedrückt hat. Wenn der Benutzer beispielsweise eine Taste drückt und wieder loslässt, wird die Methode `keyPressed` mit etwas aufgerufen, mit dem die gedrückte Taste als Argument abgerufen werden kann. Wir empfehlen Ihnen, sich das Skelett anzusehen, das in der Klasse `KeyboardListener` bereitgestellt wird, um weitere Informationen zu erhalten.

1. Beobachten Sie die Klasse, insbesondere wie es möglich ist, auf spezielle Tasten der Tastatur wie die Funktionstasten (HOME, F1-F12...) zu reagieren.
2. Beachten Sie, dass eine Instanz des Spiels und eine Instanz des Labyrinths in der Klasse vorhanden sind. Wozu dienen sie Ihrer Meinung nach?
3. Stellen Sie sicher, dass Sie den Spieler mithilfe der Tasten bewegen können.
4. Sorgen Sie dafür, dass es möglich ist, die Lösung einzublenden. Dazu müssen Sie eine Taste mit der Ihnen zur Verfügung gestellten Methode `displaySolution` verknüpfen.

Bemerkung :

Das folgende Klassendiagramm kann Ihnen helfen, wenn Sie sich über die Beziehungen zwischen den Klassen im Unklaren sind :

[1] Dies ist nicht die eleganteste Art, dies zu tun. Was passiert nämlich, wenn Sie die X-Position von Spieler 2 suchen und er nicht da ist? Versuchen Sie es, wenn Sie Zeit haben. Angesichts der heute verfügbaren Zeit ist dies jedoch ein vernünftiger Weg.

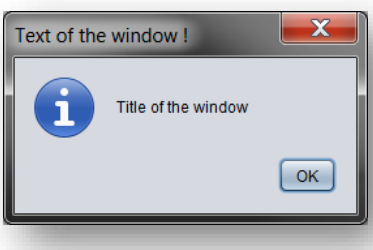


Aufgabe7 - Ausgang aus dem Labyrinth

Nachdem die Tastatur nun korrekt funktioniert, muss es möglich sein, eine Meldung anzuzeigen, wenn der Benutzer den Ausgang gefunden hat. Das ist der Zweck der Methode `checkWinner`, die Sie implementieren müssen. Um eine Nachricht in einem neuen Fenster anzuzeigen, können Sie die folgende[2] Technik verwenden:

```
JOptionPane.showMessageDialog(null, "Titel des Fensters", "Text des Fensters! ",
,JOptionPane.INFORMATION_MESSAGE);
```

was zu einem Fenster wie unten führt:



Wir wünschen Ihnen viel Spass bei der Umsetzung dieses Projekts!

[2] Dieses Beispiel befindet sich als Kommentar am Ende von `MazeGame`, so dass Sie diesen direkt kopieren können.