# 34. Bundeswettbewerb Informatik – Aufgabe 1

*Team „ByteSector“*

## Lösungsidee

Da es sich bei den Beispielen um relativ kleine Karten handelt und die Anzahl an möglichen Aktionen pro Zug auf maximal vier beschränkt ist, bietet es sich an, dieses Problem mittels Breitensuche zu lösen.

Dazu wird jeder Zustand von Quadratien mit einem eigenen Objekt beschrieben, das eine eigene Version der Karte, Kassiopeias Position und den Zug, der zu diesem Zustand geführt hat, speichert. Auf Befehl erstellt dieses Objekt alle gültigen der vier Zustände, die mit einem weiteren Zug erreicht werden können und speichert Referenzen auf diese in einer eigenen Liste. Anschließend fragt der vorherige Zustand seine Unterzustände, ob mit der Erweiterung eine Lösung gefunden wurde.

Die in der Klasse der Zustände enthaltene Methode zur Erfolgsprobe untersucht, ob es noch unbesuchte Felder gibt. Sollte es sie nicht geben, gilt der Zustand als Lösungszustand. Im dem Falle wird der Zug, der zu diesem Zustand geführt hat, sprich „N“, „O“, „S“ oder „W“, in einer globalen Variable gespeichert und dem Oberzustand, der die Prüfung ausgelöst hat, wird Erfolg signalisiert.

Sollte es vorkommen, dass keine der vier Züge zu einem gültigen Zustand führen würde, wobei es selbstverständlich ist, dass der Umkehrzug zum vorherigen Zustand immer ungültig ist, wird ein Fehlschlag zurückgegeben.

Erreicht der Befehl zur Erweiterung den Zustand im Laufe der nächsten Iteration ein weiteres Mal, gibt es diesen Befehl einfach an seine Folgezustände, die wiederum für das Erstellen weiterer Zustände sorgen, weiter und handelt entsprechend deren Rückgabewerte:

1. Meldet ein Zustand einen Fehlschlag, war er nicht in der Lage, einen weiteren gültigen Zug zu machen, und wird samt all seinen Unterzuständen gelöscht.
2. Meldet ein Zustand Fortschritt, wurden neue Zustände erstellt, von denen allerdings noch keiner zur Lösung führt. In diesem Falle bleibt der Zustand erhalten und es wird, sollte es nicht zu Möglichkeit 3 kommen, ebenfalls Fortschritt an der Oberzustand gemeldet.
3. Meldet ein Zustand Erfolg, hat dieser beim Erstellen von Zuständen oder beim Weiterleiten des Befehls eine Lösung gefunden und bereits alle nötigen Züge zur Reproduktion des Lösungsweges vom Ende bis zu diesem Zustand in der globalen Variable gespeichert. In diesem Falle wird die globale Variable um den eigenen Zug erweitert und die Meldung an der Oberzustand weitergeleitet. Im Laufe dieser Rückmeldungskette werden so alle Züge, die vom Startzustand zum Lösungszustand geführt haben, zusammengetragen.

## Umsetzung

Die Umsetzung erfolgt als ein in Java geschriebenes Programm und erweitert den Lösungsvorschlag zur Junioraufgabe 2. Exportiert wird diese Version ebenfalls als Jar-Datei gepaart mit einer Batch-Datei zur Ausführung. Getestet wurde das Programm unter Windows 7 Home Premium SP1 64-Bit mit JDK 1.8.0_65 64-Bit.

Ähnlich zum Lösungsvorschlag zur Junioraufgabe 2 erfolgt die Annahme der Datei, welche Informationen über die Beschaffenheit von Feldern und Kassiopeias Startposition beinhaltet, und die Speicherung der Informationen in Variablen am Anfang des Konstruktors der Main-Instanz. Anschließend wird die in Junioraufgabe 1 erstellte Methode zur Prüfung der Erreichbarkeit aller Felder verwendet, um Beispiel-Welten auszuschließen, in denen Kassiopeia selbst mit mehrfachem Betreten nicht alle Felder erreichen kann.

Die Klasse der Zustände ist in dieser Umsetzung **WalkState**, deren Instanzen jeweils einen zweidimensionalen Boolean-Array zur Beschreibung ihrer Version der

Karte, zwei Integer-Variablen zur Beschreibung von Kassiopeias aktueller Position, einen String zum Speichern des eigenen Zuges und eine ArrayList vom Typ WalkState zur Auflistung aller Unterzustände besitzen. Außerdem besitzen sie neben der Funktion, neue Unterzustände zu bilden bzw. den Befehl weiterzuleiten, Methoden zur Erfolgsprüfung und zum Klonen des Karten-Arrays.

Die immer wieder aufgerufene Methode zur Erweiterung des Zustandsbaumes lautet *goDown()* und ist wie folgt aufgebaut:

```
public int goDown(){
	// Integer zum Notieren von Fortschritt (0 = kein Fortschritt; 1 = Fortschritt; 2 = Lösung gefunden)
	int progress = 0;

	// Wenn es noch keine Unterzustände gibt, wird versucht, welche zu erschaffen.
	if (subStates.size() == 0){
		// Nord
		// Nur weitergehen, wenn das nördliche Feld weiß ist.
		if (!map[posX][posY-1]){
			// Neue Anweisung notieren
			String newCmd = "N";
			// Neuen Zustand mit eigener Kopie der Karte etc. erstellen
			subStates.add(new WalkState(cloneMap(map), posX, posY-1, newCmd));
			// Notieren, dass Fortschritt gemacht wurde
			progress = 1;
		}
		// Osten
		if (!map[posX+1][posY]){
			String newCmd = "O";
			subStates.add(new WalkState(cloneMap(map), posX+1, posY, newCmd));
			progress = 1;
		}
		// Süden
		if (!map[posX][posY+1]){
			String newCmd = "S";
			subStates.add(new WalkState(cloneMap(map), posX, posY+1, newCmd));
			progress = 1;
		}
		// Westen
		if (!map[posX-1][posY]){
			String newCmd = "W";
			subStates.add(new WalkState(cloneMap(map), posX-1, posY, newCmd));
			progress = 1;
		}

		// Nach der Erstellung der neuen Zustände wird überprüft, ob mindestens einer von ihnen zum Lösungszustand führt.
		// Wenn keine neuen Zustände erstellt werden konnten, ist subStates leer und dieser Test wird übersprungen.
		for (WalkState curState : subStates){
			if (curState.done()){		// Erfüllt einer der Unterzustände die Bedingung
				solution = cmd + solution;	// wird der Lösungsweg um die eigene Anweisung erweitert
```

```
                        return 2;                                   // und zurückgegeben, dass eine Lösung gefunden wurde
                    }

                }

            } else {        // Falls es bereits Unterzustände gibt, wird das Kommando weitergeleitet

                // Eine Abschussliste wird verwendet, da während der Iteration über die Liste an
                // Unterzuständen keine Zustände aus ihr gelöscht werden dürfen.
                List<WalkState> deadStates = new ArrayList<WalkState>();
                for (WalkState curState : subStates){
                    int curProgress = curState.goDown();
                    switch (curProgress){
                    case 0: // Wenn der Unterzustand keinen Fortschritt meldet...
                        deadStates.add(curState);    // wird er auf die Abschussliste geschrieben
                        break;
                    case 1:// Wenn der Unterzustand Fortschritt meldet...
                        progress = 1;                       // wird dieser auch hier verzeichnet
                        break;
                    case 2: // Wenn der Unterzustand eine Lösung gefunden hat...
                        solution = cmd + solution;   // wird der Lösungsweg um die eigene Anweisung erweitert
                        return 2;                                   // und zurückgegeben, dass eine Lösung gefunden wurde
                    }

                }

                // Zustände auf der Abschussliste werden dem GC überlassen
                for (WalkState curState : deadStates)
                    subStates.remove(curState);

            }

            // Sollte bis zu diesem Zeitpunkt kein neuer Zustand erfolgreich erstellt worden sein
            // bzw. kein Unterzustand Fortschritt verzeichnet haben, ist es von diesem Zustand aus
            // unmöglich, eine Lösung zu erreichen. Es wird der Standardwert 'kein Fortschritt'
            // (progress = 0) an den Oberzustand zurückgegeben.

            return progress;
        }
```

BEISPIELE

| Beispiel-Welt | Ausgabe |
|---|---|
| kassiopeia0 | WNNWSSSOOONNNOOOSSSWNN |
| kassiopeia1 | Keine Lösung möglich, da Kassiopeia nicht alle Felder erreichen kann. |
| kassiopeia2 | Keine Lösung möglich. Kassiopeia kann nicht alle Felder erreichen, ohne mindestens eines mehrfach zu betreten. |
| kassiopeia3 | OOOOSSWNWSWWWWNNO |
| kassiopeia4 | Keine Lösung möglich. Kassiopeia kann nicht alle Felder erreichen, ohne mindestens eines mehrfach zu betreten. |
| kassiopeia5 | WWWWWWWWWWWW |
| kassiopeia6 | Keine Lösung möglich. Kassiopeia kann nicht alle Felder erreichen, ohne mindestens eines mehrfach zu betreten. |
| kassiopeia7 | Keine Lösung möglich. Kassiopeia kann nicht alle Felder erreichen, ohne mindestens eines mehrfach zu betreten. |