mondrian

A solver for the puzzle game https://mondrianblocks.com/

1) Was wurde gemacht?

Wir haben für das Puzzle-Spiel Mondrian Blocks ein Lösungs-Programm entwickelt.

Bei Mondrian Blocks gibt es ein quadratisches, 8x8 Felder großes Spielfeld. Außerdem gibt es einige Blöcke, die das Spielfeld zusammen vollständig ausfüllen. Am Start des Spiels sind einige Blöcke jeweils bereits auf dem Spielfeld platziert, und die restlichen Blöcke müssen nun so angeordnet werden, dass alle auf dem Spielfeld Platz finden.
Mit unserem Programm kann nun diese Ausgangslage per Konsoleneingabe eingegeben, und alle Lösungen für diese angezeigt werden.
Der Lösungsalgorithmus basiert darauf, einen Block nach dem anderen an allen noch freien Positionen des Spielfeldes zu platzieren. Jede dieser Platzierungen wird als Kopie des Spielfelds gespeichert und als Ausgangslage für den nächsten Berechnungsschritt genutzt. In den meisten Fällen bleibt irgendwann kein Platz mehr für den nächsten Block, und dieser Lösungsansatz wird verworfen. Passiert dies nie, handelt es sich um eine valide Lösung, die schließlich zurück gegeben wird.
Im einem Berechnungsschritt wird also der ein Block auf jedes freie Feld des Spielbretts platziert. Für jede Möglichkeit wird eine Kopie des Spielfelds mit diesem platzierten Block erstellt. Auf jede dieser Kopien wird nun wieder dieser Berechnungsschritt angewandt. Dies wird wiederholt, bis keine Blöcke mehr übrig sind.
Wir haben auch Optimierungsmöglichkeiten für den Algorithmus probiert. Beispiel: Die kleinsten Blöcke zuerst platzieren, um, sollte es auf dem Spielfeld eine 1-breite Lücke, aber keinen so schmalen Block mehr geben, den Versuch direkt zu verwerfen. Es hat sich aber herausgestellt, dass es am effizientesten ist, die größten Blöcke zuerst zu platzieren, um so früh die Anzahl zu prüfender Varianten zu begrenzen.

2) Wie ist der Installationsprozess?

Es gibt keine Dependencies, das Programm kann einfach durch Aufruf von main in Game gestartet werden. Wichtig ist, dass das Terminal ANSI-Farbcodes unterstützt.

3) Wie ist die Bedienung?

Wir haben einen Algorithmus entwickelt, der alle möglichen Anordnungen der übrigen Blöcke findet, sodass alle Blöcke auf das Spielfeld passen. Die Positionen der vorgegebenen Blöcke können per Konsoleneingabe festgelegt werden, und anschließend werden alle gefundenen Lösungen angezeigt. (`$ runhaskell ./Game.hs`)
Außerdem haben wir eine weitere, allgemeinere Konsoleneingabe implementiert, bei der die Größe des Spielfelds und die Anzahl, Position und Maße aller Blöcke komplett frei angegeben werden kann. Der zugrunde liegende Lösungsalgorithmus ist hierbei der selbe. (`$ runhaskell ./Game.hs`)

Beispielprotokoll (Originalspiel)

```
$ runhaskell ./Game.hs
Enter x,y for block 1x1. Position 0,0 is at the upper left corner.
2,3
Enter x,y,orientation (v/h) for block 1x2
0,1,v
Enter x,y,orientation (v/h) for block 1x3
4,7,h
Initial board:

▉
▉
  ▉



    ▉▉▉
--------
Possible solutions:
```

Die Darstellung in der Konsole erfolgt wie im Screenshot zu sehen als farbige Blöcke. Dadurch lassen sich gleichfarbige Blöcke nicht immer zweifelsfrei voneinander abgrenzen, was den Limitierungen einer Konsolenausgabe geschuldet ist. Deshalb können auch Lösungen scheinbar wie Duplikate wirken. In der Praxis stellt dies jedoch kein Problem dar, da für das Lösen des Puzzles die genaue Platzierung der Blöcke irrelevant ist. Außerdem ist durch die verschiedenen Farben sichergestellt, dass die Lösung immer ausreichend eindeutig ist.

Sprachkonzepte

- Listen - ja

Listen werden in den meisten Funktionen verwendet. Bei der Berechnung aller Lösungsmöglichkeiten werden die verschiedenen Spielbretter, welche alle möglichen Kombinationen darstellen, als Liste gespeichert.

- list comprehension - ja

Wird verwendet, etwa in Collision#allPositionsBoard.

- Funktionen mit pattern matching - ja

In zahlreichen Funktionen wird Pattern Matching verwendet, etwa in Collision#placeOnBlocksIfLegal, wo es verwendet wird, um eine Liste zu Teilen und Rekursion zu vereinfachen. Auch in Collision#isInBounds, Collision#allOccupiedPositions, und anderen Funktionen wird es verwendet.

- Funktionen mit guards - ja

Wird etwa in `Collision#placePositionsIfLegal` verwendet.

- Rekursive Funktionen - ja

Rekursion wird an allen Stellen eingesetzt, wo wiederholte Ausführung nötig ist, etwa in `Collision#placeOnPositionsIfLegal` oder `CustomGame#getUserInput`.

- Funktionen höherer Ordnung wie map, filter, fold - ja

Zum Beispiel concatMap in `Collision#solveGame` und `Collision#allOccupiedPositionsBoard`.

- Fehlerbehandlung ggf. mit Either oder Maybe - ja

Der Algorithmus wurde so entwickelt, dass Fehlerbehandlung dort nicht notwendig ist. Wenn es keine validen Lösungen für den eingegebenen Spielstand gibt, dann ist das kein Fehler, entsprechend wird eine leere Liste zurück gegeben. Allerdings wird Maybe in `CustomGame#getUserInput` zurückgegeben, wenn mehr Blöcke angegeben werden, als auf das Board passen. Dort wird auch weitere Fehlerbehandlung der Eingabe durchgeführt, indem geprüft wird, ob die Anzahl der Argumente korrekt ist. Falls dies nicht der Fall ist, wird die Eingabe wiederholt.

- Eigene Datentypen - ja

Ja. Eigene Datentypen waren essenziell, um das Programm effizient umzusetzen. Analog zu den realen Spielelementen – Spielbrett und Blöcke – sind in GameElements Datentypen definiert.

- Ein/Ausgabe mit IO-Monaden - ja

Ja. Sowohl in Game als auch in CustomGame kommen sie im Kontext von getLine vor.

- Modularisierung - ja

Bereits erwähnt wurde, dass die Datentypen in GameElements ausgelagert wurde. Der eigentliche Algorithmus ist so gekapselt in Collision.

- Übersichtlicher Code (ggf. let / where verwendet) - ja

Damit der Code möglichst leserlich ist, haben wir den Ablauf in möglichst viele Teilfunktionen zerlegt und mit Kommentaren ergänzt. Wo eine weitere Zerlegung nicht möglich war, haben wir where und let verwendet, um den Ablauf innerhalb der Funktion deutlicher zu machen, etwa in `Collision#placeOnPositionsIfLegal` und `Collision#solveSingleBlock`.

- wichtigste Teile des Codes dokumentiert - ja

Alle Funktionen des Lösungsalgorithmus sind kommentiert, um die Implementierung nachvollziehbarer zu machen.