### Aufgabe 5: Hüpfburg (lib/task5.dart)

**Problem:** Ein Parcours wird von zwei Spielern gleichzeitig absolviert. Können die beiden Spieler sich treffen und wenn ja nach welcher Abfolge von Sprüngen? Der Parcours besteht aus Feldern. Jedes Feld ist zu bestimmten anderen Felder verbunden. Somit kann ein Spieler von diesem Feld nur zu den anderen verbundenen Feldern hüfpen.
**Lösungsidee:** Um zu bestimmen, ob der Parcours sich beide Spieler treffen und wenn ja wie, wird der Parcours "simuliert". Jeder Spieler startet auf einem Spielfeld. Ein Feld entspricht der Klasse `Punkt`. Ein Spieler entspricht der Klasse `Runner`. Es gibt zwei Arten von Runnern: `Runner` des Typ 0 (Spieler 1) und `Runner` des Typ 1 (Spieler 2). Falls auf einem Feld (`Punkt`) zwei `Runner`, welche nicht den gleichen Typ haben, stehen, haben die Spieler sich getroffen und der Parcours wurde bewältigt. Jeder `Runner` wird gleichzeitig auf das nächste Feld verschoben. Wenn es mehrere Felder zur Auswahl gibt, dann wird pro möglichem Feld ein `Runner` erzeugt, sodass jeder mögliche Bewegungsablauf gleichzeitig existiert.

Zunächst habe ich die Klasse `Punkt` (also ein Feld) konstruiert. Diese hält ein Set an Verbindungen, eine id (Nummer des Punktes) und ein Set mit den aktuellen Runnern, die sich auf dem Punkt befinden. Die `test` Methode überprüft, ob sich in dem Set der aktuellen Runner des Punktes `Runner` unterschiedlicher Typen befinden. Wenn ja, werden diese `Runner` als Set ausgegeben (wenn nicht, wird nichts ausgegeben). Runner können dem Punkt hinzugefügt und wieder entfernt werden. Beim Hinzufügen wird dem Set ein Runner hinzugefügt. Beim Entfernen wird die `runnerAbmelden` Methode genutzt. Dort wird die Anzahl an Runnern, die sich auf dem Feld / Punkt befindenen, mit der Anzahl an Verbindungen des Feldes geleichgesetzt. D.h. es werden entweder Runner entfernt oder neu hinzugefügt. Wenn ein Runner neu hinzugefügt wird, nutzt er die Werte eines anderen Runners als Basis (siehe l. 496).

```
    int i = 0;
    for (Runner runner in aktuelleRunner) {
      int verbindung = verbindungen.elementAt(i);
      runner.gelaufeneFelder.add(runner.naechsterPunkt);
      runner.naechsterPunkt = verbindung;
      ausgabeRunners.add(Runner(runner.typ, runner.gelaufeneFelder,
runner.naechsterPunkt));
      i++;
    }
```

Im nächsten Abschnitt (siehe oben) wird den aktuellen Runnern ihre Verbindung zugeorndet und die Liste der gelaufenen Felder aktualsiert. Dem Set `ausgabeRunners` werden nun Kopien der aktuellen Runner hinzugefügt. Die Liste der aktuellen Runner wird nun geklärt, sodass neue Runner das Feld neu besetzen können.

Die Punkte des Parcours werden zu Beginn der Analyse (`analysieren()`) samt ihrer Verbindungen erstellt und in die Liste `aktuellePunkte` aufgenommen.

Die Klasse des Runners beinhaltet die Liste an gelaufenen Feldern und den Typ des Runners. Außerdem gibt die Variable naechsterPunkt an, welchem Punkt der Runner als nächstes zugeorndet wird.

Die Simulation wird von der rekrusiven `handleRunner` Methode gesteuert. Diese wiederholt sich solange, bis ein Runner-Paar gefunden wurde oder die Anzahl an gelaufenen Feldern größer ist als die Anzahl an Punkten zum Quadrat. Dieser Wert wurde als Maximalwert für die Weglänge bestimmt. Bei nicht rekursiven Wegen (also Wegen, bei

denen sich kein Feld wiederholt) entspricht die maximale Weglänge die Anzahl an Feldern. Wenn sich die Felder allerdings wiederholen können, erhöht sich diese Anzahl. Beispiel: Bei huepfburg1.txt treffen sich beide Spieler nur nach mehreren Durchläufen (durch den Parcours). Wenn Sasha auf Feld 1 und Mika auf Feld 2 startet, so gelingt es nach 121 Feldern, dass beide auf ein Feld kommen. Bei 17 Feldern sind 121 Felder weniger als der Maximalwert 17 * 17 (=289).

Die Methode `handleRunner` iteriert durch die Liste der aktuellen Runner und fügt dem Punkt (aus `aktuellePunkte`), welcher bei Runner als naechsterPunkt bestimmt ist, den Runner hinzu (`runnerAddieren`). Dann iteriert das Programm durch die aktuellen Punkte (`aktuellePunkte`). Bei jedem Punkt wird die `test()` Methode gerunnt. Es wird bei jedem Punkt überprüft, ob sich Runner mit unterschiedlichen Typen gelichzeitig dort befinden. Wenn dies bei keinem Punkt der Fall ist, werden alle Runner wieder von dem jeweiligen Punkt getrennt (`runnerAbmelden`) und die Methode wiederholt sich.

Am Ende wird das Runner-Paar, falls eines gefunden wurde, in die UI übertragen.

**Beispiel:** In huepfburg0.txt startet Sasha an Feld 1 und Mika an Feld 2. Zunächst wird die Liste der Punkte erstellt (Methode: `analysieren`):

```
[Punkt: 1, {18, 8, 4}, Punkt: 2, {3, 19}, Punkt: 3, {19, 6}, Punkt: 4, {}, Punkt: 5, {15, 12, 13}, Punkt: 6, {2, 12}, Punkt: 7, {5, 8}, Punkt: 8, {18, 4}, Punkt: 9, {4, 1, 14}, Punkt: 10, {16, 3, 12, 18}, Punkt: 11, {19}, Punkt: 12, {3}, Punkt: 13, {7, 10}, Punkt: 14, {17, 16, 10, 18, 1}, Punkt: 15, {12}, Punkt: 16, {11, 20, 17, 19}, Punkt: 17, {11, 9}, Punkt: 18, {13, 7}, Punkt: 19, {20}, Punkt: 20, {3, 10}]
```

[Punkt: Nummer, Set der Verbindungen]

Als nächstes werden beide Runner erstellt (Methode `runnerErstellen`):

```
[Runner: 0 [] 2, Runner: 1 [] 1]
```

[Runner: Typ GelaufeneFelder StartFeld]
Dann wird die "Simulation" gestartet... Nach 4 Iterationen liegen folgende Runner vor:

```
[Runner: 0 [2, 3, 6] 2, Runner: 0 [2, 3, 6] 12, Runner: 1 [1, 18, 7] 5, Runner: 1 [1, 18, 7] 8, Runner: 1 [1, 18, 13] 7, Runner: 1 [1, 18, 13] 10, Runner: 1 [1, 8, 18] 13, Runner: 1 [1, 8, 18] 7, Runner: 0 [2, 3, 19] 20, Runner: 0 [2, 19, 20] 3, Runner: 0 [2, 19, 20] 10]
```

Die `test()` Methode stellt nun fest, dass `Runner: 1 [1, 18, 13] 10` und `Runner: 0 [2, 19, 20] 10` sich auf Feld 10 treffen. -> Damit ist der Parcours möglich und der Weg beider Runner wird in der UI dargstellt:

```
Ausgabe:
Mika: 2 -> 19 -> 20 -> 10
Sasha: 1 -> 18 -> 13 -> 10
```

Ob ein Parcours möglich ist, wird also durch das Ausführen der Simulation überprüft. Das Programm kann auf alle Beispielaufgaben angewendet werden.