Praktische Informatik 2 SoSe 2017

Übungsblatt 1

Abgabe bis: 2017-04-30, 23:59 Uhr MESZ

Auf diesem Übungsblatt sollt ihr die Wegfindung für das Spiel *PI2 verrücktes Labyrinth* erstellen. Daher hier zunächst die Spielregeln.

Spielregeln

In Abbildung 1 seht ihr im linken Teil das Spielfeld. Es besteht aus Plättchen, die in sieben Zeilen und sieben Spalten als Gitter angeordnet sind. Auf diesen Plättchen sind Wege eingezeichnet. Oben rechts und oben links seht ihr jeweils eine Spielfigur. Ziel des Spiels ist es, das jeweils diagonal gegenüberliegende untere Plättchen zu erreichen, d. h. die blaue Spielfigur muss das Plättchen unten rechts und die schwarze Spielfigur das Plättchen unten links erreichen.

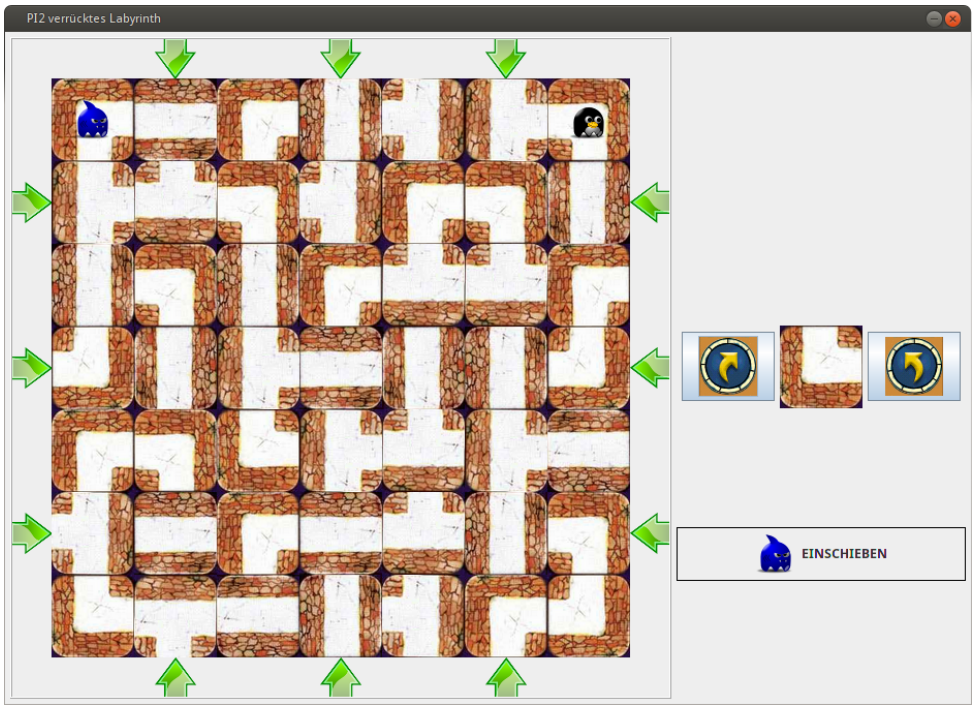

Abbildung 1: PI2 verrücktes Labyrinth

Die SpielerInnen sind nun abwechselnd am Zug, wobei immer diejenige mit der blauen Spielfigur beginnt. Zunächst muss das SchiebePlättchen in das Spiel eingeschoben werden. Das geht an all den Stellen, an denen sich im Spielfeld ein grüner Pfeil befindet. Wird das Plättchen z. B. am Pfeil oben rechts eingeschoben, dann schieben sich alle Plättchen in dieser Zeile um eine Position nach links, wobei das Plättchen ganz links aus dem Spielfeld herausfällt. Dieses Plättchen wird das neue SchiebePlättchen, das im Zug der nächsten SpielerIn eingeschoben werden muss. Das SchiebePlättchen darf vor dem Einschub beliebig gedreht werden (um 90, 180 oder 270 Grad). Eine Spielfigur wird ggf. mit ihrem Plättchen verschoben, außer sie wird aus dem Spielfeld geschoben. In diesem Fall wird sie auf das gerade eingeschobene Plättchen versetzt.

Nun darf dieselbe SpielerIn ihre Spielfigur auf ein anderes Plättchen setzen (muss das aber nicht). Das Zielplättchen darf beliebig weit weg sein, muss allerdings durch einen Weg auf dem Spielfeld vom Ausgangsplättchen erreichbar sein (z. B. darf die blaue Spielfigur in Abbildung 1 drei Plättchen weiter nach unten ziehen, die schwarze Spielfigur aber nicht zwei Plättchen weiter nach links). Die Spielfeldgrenzen dürfen dabei nicht überschritten werden.

Nach dem Versetzen oder Stehenlassen der Spielfigur ist die andere SpielerIn an der Reihe und muss nun zunächst wieder das SchiebePlättchen einschieben.

Sobald eine der beiden Spielfiguren auf ihr entsprechendes Plättchen unten rechts bzw. links gesetzt wird, ist das Spiel beendet und die SpielerIn dieser Spielfigur hat gewonnen.

Vorbemerkung

Ihr findet bei Stud.IP ein gepacktes Eclipse-Projekt *PI2Laby*. In diesem Projekt befinden sich mehrere Interfaces und diverse Klassen, die die wesentliche Logik des Spiels und die in Abbildung 1 dargestellte GUI (Graphical User Interface = grafische Benutzeroberfläche) realisieren.

Aufgabe 1 PushPopy [30 Punkte]

Im Paket *pi2.uebung01.spec* findet ihr u. a. die Interfaces `PushPopyInterface`, `LiFoInterface` und `FiFoInterface`. Diese Interfaces beschreiben Schnittstellen für konkrete Sammlungen, die Elemente eines beliebigen Typs aufnehmen können. Programmiert eine Klasse, die das Interface `LiFoInterface` implementiert und eine weitere Klasse, die das Interface `FiFoInterface` implementiert. Da es sich bei den Sammlungen um beschränkte Sammlungen handelt, müsst ihr als unterliegende Datenstruktur Arrays verwenden, d. h. ihr speichert die Elemente der Sammlungen jeweils in einem entsprechenden Array ab.

Erzeugt zu beiden Klassen JUnit-Tests und testet damit eure Implementierung. Das Testprotokoll besteht dann nur aus dem Ergebnis dieser Tests.

Aufgabe 2 Labyrinth [70 Punkte]

Aufgabe 2.1 Plättchen [20 Punkte]

Die abstrakte Klasse `Tile` realisiert eine Oberklasse für die Plättchen, die sich auf dem Spielfeld befinden können. Sie fasst die gemeinsamen Attribute und Methoden zusammen und gibt für konkrete Unterklassen die Methode *boolean hasExit(final Direction pDirection)* vor. Die Aufzählung `Direction` definiert die Aufzählungselemente `NORTH`, `EAST`, `SOUTH` und `WEST`.

Implementiert nun in den drei konkreten Unterklassen `Curve`, `Straight` und `Junction` jeweils die Rümpfe der Methode *hasExit()*. Diese Methode gibt *true* zurück, falls das jeweilige Plättchen einen Ausgang in der gegebenen Richtung hat. Das Ergebnis ist also abhängig von der Orientierung (d. h. Rotation) des Plättchens. In Abbildung 2 seht ihr die drei Plättchenarten mit der Orientierung `Direction.NORTH`.

Für das T-Plättchen in Abbildung 2a liefert die Methode *hasExit()* nur für den Parameterwert `Direction.NORTH` *false* zurück, ansonsten *true*. Die Orientierung `Direction.EAST` entspricht einer Drehung der Plättchen in Abbildung 2 um 90 Grad im Uhrzeigersinn, `Direction.SOUTH` um 180 Grad im Uhrzeigersinn und `Direction.WEST` um 270 Grad im Uhrzeigersinn. Wenn also

(a) Junction (b) Curve (c) Straight

Abbildung 2: Die drei Arten von Plättchen

das Plättchen *Curve* aus Abbildung 2b die Orientierung `Direction.SOUTH` hat, dann liefert die Methode *hasExit()* für die Parameterwerte `Direction.NORTH` und `Direction.WEST` *true* zurück, ansonsten *false*.

Erzeugt JUnit-Tests, die für jede der drei Klassen jeweils die Methode *hasExit()* testen. Das Testprotokoll besteht dann nur aus dem Ergebnis dieser Tests.

Aufgabe 2.2 Spielfeld [50 Punkte]

Die Klasse `Board` realisiert das eigentliche Spielfeld. Auf dem Spielfeld befinden sich immer 7 Plättchen pro Zeile in 7 Zeilen, also 7 mal 7 Plättchen (siehe Konstanten in `PI2Laby`). Leider fehlt in der Klasse die entscheidende Funktionalität. Die Methode *boolean existsPath(final PieceInterface pPiece, final int pRow, final int pColumn)*, die feststellen soll, ob die gegebene Spielfigur das Plättchen an den gegebenen Koordinaten erreichen kann, löst in ihrem Rumpf nur eine `UnsupportedOperationException` aus. Ersetzt den Rumpf durch eine funktionierende Wegfindung.

Dabei wendet ihr folgendes Verfahren an:

Zunächst erzeugt ihr eine Sammlung vom Typ `PushPopyInterface` für die noch zu bearbeitenden Plättchen (hier genannt `Rand`). Fügt jetzt das Plättchen, auf dem sich die gegebene Spielfigur befindet, in die Sammlung `Rand` ein. Nun beginnt eine Schleife, die so oft wiederholt wird, wie sich noch Elemente in der Sammlung `Rand` befinden. Innerhalb der Schleife entnehmt ihr mittels *pop()* ein Plättchen `P` aus dem Rand. Wenn dieses Plättchen bereits das Ziel ist, seid ihr fertig und könnt *true* zurückgeben. Anderenfalls bestimmt ihr für `P` alle Nachbarplättchen, zu denen es einen Weg gibt (d. h. ihr müsst für alle Ausgänge von `P` das jeweilige Nachbarplättchen untersuchen und feststellen, ob es einen Ausgang in Richtung `P` hat). Diese Nachbarplättchen sind also alle vom aktuellen Plättchen aus erreichbar (und umgekehrt). Fügt diese Nachbarplättchen daher in die Sammlung `Rand` ein. Falls das Zielplättchen nicht erreichbar ist, wird es auch niemals als Nachbarplättchen bearbeitet, also nie in den Rand gelangen. Wenn der Rand dann schließlich leer ist, ist eure Schleife beendet und ihr könnt *false* zurückgeben.

Wenn ihr den Algorithmus exakt wie oben umsetzt, kann es passieren, dass die Wegfindung konzeptionell sehr sehr lange dauert bzw. dass euch eure `PushPopyInterface`-Sammlung auseinanderplatzt. Das liegt daran, dass bereits bearbeitete Plättchen später als Nachbarplättchen erneut in den `Rand` eingefügt werden. Erzeugt daher eine beliebige konkrete Sammlung vom Typ `Collection`, die die im Laufe des Algorithmus bereits bearbeiteten Plättchen enthält. Ein Plättchen sollte nur dann in den `Rand` gelangen bzw. nur bearbeitet werden, wenn es nicht in der Sammlung der bereits bearbeiteten Plättchen enthalten ist.

Verwendet für den Rand bei der Wegfindung einmal eine Sammlung vom Typ `LiFoInterface` und einmal vom Typ `FiFoInterface` und vergleicht deren Nutzung (in dem ihr z. B. die Zahl

der Schleifendurchläufe zählt). Gibt es Unterschiede? Erläutert, warum es keinen Unterschied macht, welche Datenstruktur eingesetzt wird oder worin die Unterschiede bestehen und wie sie zustande kommen.

JUnit-Tests sind hier nicht nötig. Die Tests erfolgen durch Spielen des Spiels: Testet eure Implementierung, in dem ihr ein oder mehrere Spiele mit der GUI spielt. Protokolliert dabei knapp(!), was funktioniert und was evtl. nicht.

Hinweise

Wenn eine SpielerIn die eigene Spielfigur nicht setzen möchte, klickt sie einfach auf das Plättchen, auf dem sich die Spielfigur gerade befindet. Dieser Fall sollte also auch getestet werden!