Fachbereich Mathematik und Informatik
Prof. Dr. Klaus Ostermann

Paolo Giarrusso

28. April 2011

2. Abgabe
Software-Praktikum 2011

Mit dieser Abgabe erzeugt ihr Version 0.2 eures Programms. In dieser Version implementiert ihr die Spiellogik weiter und testet ihr diese, und wenn möglich fangt ihr an, die Netzwerk-Unterstützung zu entwickeln. Wenn ich eine Aufgabe als „*momentan optional*“ beschreibe, ermuntere ich dazu, dass ihr das bis zur Abgabe schon entwickelt. Auf jeden Fall, soll eine solche Aufgabe bis zur darauf folgenden Abgabe fertig sein.

Abgabe:

- Bis Dienstag 10. Mai, 12:00, ins SVN-Repository eurer Gruppe, in einen Branch `version-0.2`.

1 Austauschbare Text-Testfälle erzeugen und testen

Schreibt vorschriftsmäßig Testfälle-Dateien für eure Implementation (Eingabe und erwartete Ausgabe). Jeder Testfall umfasst eine Sequenz von Zügen, das heißt ein `Board` (welches in der Beschreibung des Netzwerk-Protokolls angegeben ist). Für jeden Zug wird eine Zeile ausgegeben, die den neuen Punktestand zeigt, wenn der Zug gültig ist. Die Zeile richtet sich nach Format `Option[Score]` (auch siehe Netzwerk-Protokolls Beschreibung).

Ihr sollt auch ein kleines Test-Framework schreiben, dass die Eingabe und erwartete Ausgabe liest und dann bestätigt, ob eure Implementation die korrekte Ausgabe erzeugt. Dann sollt ihr mit diesem Framework Testfälle von mir und von euch (demselben Muster entsprechend) testen. Dieses Test-Framework kann als ein JUnit-Test implementiert werden; auf jeden Fall muss es die Ergebnisse ausgeben. Ihr sollt Testfällen schreiben, so dass die besondere Schwierigkeiten (z.B. Grenzfälle wie Arraygrenzen) für eure Implementation auch getestet sind.

Diese sind Integrations-Testfälle, weil sie die komplette Spiellogik testen. Andere JUnit-Testfälle testen individuelle Methoden. Beide Arten von Testfälle sind wichtig.

Abgabe:

- In das Verzeichnis `tests` unter den Branch `version-0.2`. Jeder Testfall besteht aus `inputN.txt` und `outputN.txt`, z.B. `input5.txt` und `output5.txt`.

2 JUnit Tests für Koordinaten-Umrechnung

Wenn ihr ein anderes internes Koordinatensystem für das Gitter implementiert habt, schreibt ihr Methoden für die Umrechnung und Testfälle für diese Methoden.

3 Spiellogik und JUnit Testen

Die Spiellogik soll:

- bestätigen, ob ein Feld im Gitter ist;
- bestätigen, ob zwei Felder benachbart sind;

- bestätigen, ob ein Zug gültig ist; auch die Regeln für den ersten Zug sollen korrekt implementiert sein;
- den Gitter-Zustand speichern, um zu wissen, was es in jedem Feld gibt;
- den Gitter-Zustand bei Durchführung eines Zuges ändern;
- Usw.

Für die Abgabe sollt ihr die Spiellogik fertig machen, und auch die Punkte berechnen. Ihr sollt eine Test-Abdeckung von 75% erreichen. Tests sind besonders wichtig für die komplexeren Methoden.

Die Schnittstelle für eure Punkte-Berechnung soll auch zurückgeben, ob ein Bonus-Zug kommt. Der korrekte Berechnung von Bonus-Züge ist *momentan optional*, aber es ist besser (und erfordert) die korrekte Schnittstelle schon zu entwickeln.

(*Momentan optional*) Für die Künstliche-Intelligenz soll eure Spiellogik auch gültige Züge erzeugen.

4 Netzwerkkommunikation Unterstützung anfangen

(*Momentan optional*) Login komplett ausführen und einen ersten Zug schicken. Für diese Abgabe, ist es noch nicht wichtig, ob der geschickte Zug gültig ist. Deswegen sollt ihr noch nicht die Netzwerk-Unterstützung und Spiellogik verbinden. Natürlich sollt ihr trotzdem dieselben Klassen benutzen, um Züge usw. zu darstellen!