IMS
Institut für Mikroelektronische Systeme
Leibniz Universität Hannover

Programmierpraktikum Technische Informatik (C++)
Aufgabe 07

Hinweise
Abgabe: Stand des Git-Repositories am 23.6.2020 um 9 Uhr.
Die Dateien zur Bearbeitung dieser Aufgabe erhalten Sie, indem Sie die neue Aufgabe aus dem Aufgabenrepository in Ihr lokales mergen. Dies geschieht mit `git pull cpp2020 master` innerhalb Ihres Repositories. Die Lösungen committen Sie bitte in Ihr lokales Repository (`git commit -a` oder `git add` gefolgt von `git commit`) und pushen sie in Ihr Repository auf dem git-Server des Instituts (`git push`).

Teilaufgabe 1 (3,5 Punkte)

Vervollständigen Sie das Programm im Verzeichnis `mastermind`! Wie Sie ahnen, handelt es sich um das gleichnamige Spiel (falls es Ihnen nicht bekannt ist, können Sie unter `http://de.wikipedia.org/wiki/Mastermind` die Spielregeln nachlesen).

a) Implementieren Sie die Methode `Code::evaluate`! Die Funktion soll ermitteln, wie viele schwarze und weiße Steine resultieren, wenn ein Code mit einem anderen verglichen wird.

b) Implementieren Sie die Methode `Code::createAllPossibleCombinations`! Diese Funktion soll einen `vector` mit **allen** möglichen Codes der entsprechenden Länge und Anzahl Faben zurückgeben.

c) Die Methode `Mastermind::considerResult` soll aus `possibleCodes` alle Codes entfernen, die ausgeschlossen werden können, wenn eine Evaluierung von `guess` mit dem gesuchten Code zu dem Ergebnis `result` führt. Implementieren Sie diese Funktionalität!

   Hinweis: Ein Code `c` kann ausgeschlossen werden, wenn das Ergebnis der Evaluierung von `c` und `guess` zu einem anderen Ergebnis als `result` führt.

d) Bisher kann der Nutzer nur gegen sich selber spielen. Um dies zu ändern, implementieren Sie zunächst die Methoden `Mastermind::guess` und `Mastermind::evaluate`! `Mastermind::guess` soll aus den noch möglichen Codes zufällig einen auswählen und diesen zurückgeben. `Mastermind::evaluate` soll den übergebenen Code mit dem gesuchten Code `solution` evaluieren.

Appelstraße 4 | 30167 Hannover
fon 0511 762-19691 | fax 0511 762-19694
www.ims.uni-hannover.de | info@ims.uni-hannover.de

Implementieren Sie anschließend eine von `Player` abgeleitete Klasse `AutomatedPlayer`, die das Raten bzw. Evaluieren von den soeben implementierten Methoden der übergebenen `Mastermind`-Instanz durchführen lässt. Wenn Sie die Klasse implementiert haben, müssen Sie den `typedef` entfernen, der `AutomatedPlayer` als Synonym für `ManualPlayer` definiert.

Hinweise: In dieser Teilaufgabe müssen auch Klassen geändert werden. Daher dürfen und müssen Sie auch die Header ändern. Sie können selbstverständlich auch eigene (Hilfs-)Funktionen definieren, wo Sie es für sinnvoll halten.

Teilaufgabe 2 (2 Punkte)

Im Verzeichnis `minesweeper` soll ein Programm vervollständigt werden, welches eine textbasierte Variante des Klassikers Minesweeper ohne `flagging` umsetzt. Die Spielregeln können unter `http://de.wikipedia.org/wiki/Minesweeper` nachgelesen werden. Ein Aufruf des Programms kann mit einem Parameter (`easy`, `medium` oder `hard`) zur Auswahl von einem der drei üblichen Schwierigkeitsgrade erfolgen. Das Aufdecken eines Feldes erfordert die Eingabe von Koordinaten der Form *<Zeile><Spalte>*, wobei die Zeile durch einen Buchstaben und die Spalte durch eine Zahl angegeben wird. Beispielsweise bewirkt die Eingabe der Koordinate *A1* ein Aufdecken des ersten Feldes. Die Koordinaten sind in der bereits implementierten Ausgabe deutlich erkennbar.

a) Programmieren Sie in `minesweeper/src/minesweeper.cpp` die Methode `incrementMineNeighbors`! Diese soll bei allen angrenzenden Nachbarn des übergebenen `Square` den Zähler für angrenzende Minen um eins erhöhen.

   Hinweis: Die Hilfsmethode `getSurroundingCoordinates` kann Ihnen die Iteration über die Nachbarn eines Feldes erheblich erleichtern. Dieser Methode werden die Indizes des Ausgangsfeldes sowie die Größe des Spielfeldes in Zeilen und Spalten übergeben und sie gibt die oberen linken sowie die unteren rechten Indizes einer Iteration unter Berücksichtigung der Spielfeldgrenzen als `std::pair` von `Point2D` zurück.

b) Im Konstruktor der Klasse `Minesweeper` wird die für das Spielfeld vorgesehene Datenstruktur `grid` mit unaufgedeckten Feldern befüllt, die noch keine Minen enthalten.
   Implementieren Sie die Methode `placeMines`! Diese soll die übergebene Anzahl Minen **zufällig** auf dem Spielfeld verteilen. Verwenden Sie zur Vermeidung potenziell unendlicher Laufzeit den übergebenen Vektor `indices` in Kombination mit `std::shuffle`!

   Hinweis:

   - Die Methode `Minesweeper::print` besitzt einen Parameter `reveal`, der

bestimmt, ob das Feld in unaufgedecktem oder in aufgedecktem Zustand ausgegeben werden soll. Dies kann zum Debugging nützlich sein.

- Zur Platzierung einer Mine dient die Methode `Square::layMine`. Diese ruft automatisch die Methode `incrementMineNeighbors` der übergebenen Minesweeper-Instanz auf und inkrementiert deren `mineCount`.

c) Ergänzen Sie die Methode `revealMinelessNeighbors`! Diese soll für das übergebene `Square` prüfen, ob es an eine Mine grenzt. Ist dies nicht der Fall, so sollen entsprechend dem Verhalten von Minesweeper ausgehend vom übergebenen `Square` alle Nachbarn, die noch nicht aufgedeckt sind, aufgedeckt werden.
Hinweise:

- Die Methode `Square::reveal` gibt `false` zurück, wenn der aufrufende `Square` bereits aufgedeckt gewesen ist.
- Denken Sie daran für neu aufgedeckte Felder, die nicht an Minen grenzen, rekursiv auch deren Nachbarn aufzudecken.