Hochschule Darmstadt
Prof. Dr. Benjamin Meyer

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

22. Dezember 2020

Programmieren, Algorithmen und Datenstrukturen I, WS 2020/21
Aufgabenblatt 4

4 Minesweeper

In diesem Aufgabenblatt werden wir sukzessive an einem größeren Projekt arbeiten mit dem Ziel, unser eigenes Minesweeper-Spiel zu entwickeln! Als Vorlage soll hierfür das alte Windows-Spiel dienen, wie beispielsweise in Abbildung 1 zu sehen. Falls Sie das Spiel nicht kennen, können Sie sich beispielsweise unter `http://minesweeperonline.com/` damit vertraut machen. Lesen Sie die einzelnen Aufgabenschritte sorgfältig, da alle Teilaufgaben aufeinander aufbauen werden. Vor allem die ersten Teilaufgaben sind daher sehr detailliert beschrieben, können aber mit relativ wenig Quellcode gelöst werden.

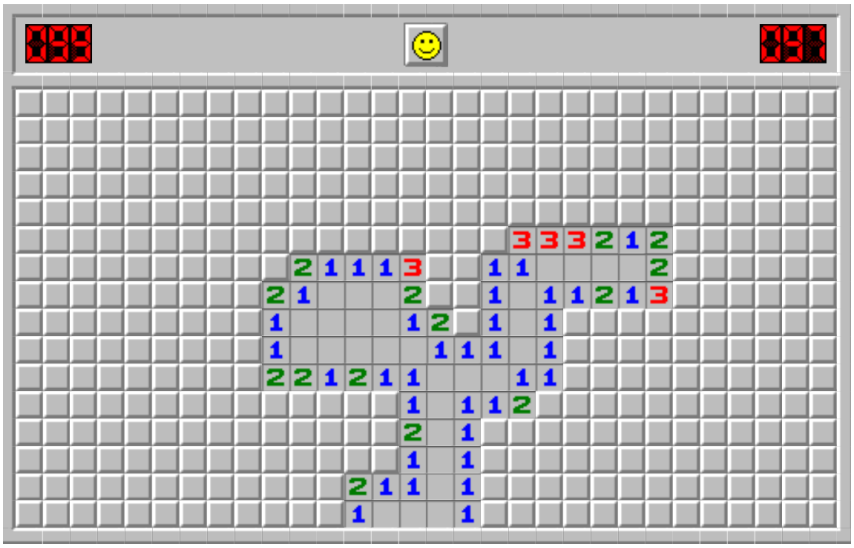

Abbildung 1: Minesweeper-Beispiel

4.1 Das Spielfeld

Der erste Schritt zum vollendeten Spiel besteht aus der Definition sowie Initialisierung des Spielfeldes. Eine Kachel des Spielfeldes soll durch den `struct Feld` beschrieben werden und soll die `Belegung` des Feldes sowie die Anzahl der Minen in den Nachbarfeldern speichern können. Legen Sie hierfür das besagte struct an sowie ein enum, um die möglichen Belegungen `offen, mine` und `aufgedeckt` zu definieren.
Wie in der Vorlesung gelernt, wollen wir zudem erstmals unser Programm aufteilen in eine `.cpp`-Datei, die den eigentlichen Quellcode enthält, sowie eine `.h`-Datei, in der wir alle wichtigen Deklarationen vornehmen und zusätzliche Bibliotheken inkludieren können. Die `.cpp` muss dann nur noch per include auf den dazugehörigen Header aufmerksam gemacht werden. Überlegen Sie sich, in welcher der beiden Dateien Sie das struct und das enum anlegen wollen.

Das Spielfeld, welches über das gesamte Programm hinweg immer wieder benötigt und verändert werden muss, ohne dass wir ständig lokale Kopien davon anlegen wollen, ist hier ein seltenes Beispiel, wo sich globale Variablen tatsächlich anbieten. Legen Sie daher Ihr Spielfeld als zweidimensionales, globales Array vom Typ `Feld` an. Die Dimensionen können Sie sich selber aussuchen, zB 12x12.

Alle weiteren benötigten Arbeitsschrittes des Programmes werden dieses Mal durch Funktionen beschrieben. Die erste dieser Funktionen soll die Initialisierung des Spielfeldes vornehmen. Funktionsnamen werden immer klein geschrieben, `initialisieren()` oder `initialize()` bieten sich daher an. Die Funktion soll zudem als Parameter die Anzahl der im Spielfeld verteilten Minen übergeben bekommen, einen Rückgabetyp braucht die Funktion nicht. `initialisieren` muss drei Arbeitspakete erledigen, in dieser Reihenfolge:

- alle Felder des Spielfeldes mit einem `offenen` Feld initialisieren, die Anzahl der Minen in der Nachbarschaft des jeweiligen Feldes kann noch auf 0 gelassen werden.
- Für jede zu verteilende Mine soll eine zufällige Position mittels `rand()` bestimmt werden, hier wird dann die Feld-Belegung auf `mine` umgestellt. Achten Sie darauf, keine bereits erzeugten Minen zu überschreiben.
- Für jedes Feld muss die Anzahl der Minen in der Nachbarschaft ermittelt und in dem entsprechenden Attribut des structs gespeichert werden. Hierfür benötigen Sie eine Hilfsmethode, die für eine gegebene Position x und y im Spielfeld sich die direkten acht (oder am Rand weniger) Nachbarn anschaut und zurückgibt, wie viele Minen hier gefunden wurden. Diese Hilfsmethode kann dann für jedes Feld des Spielfeldes aufgerufen werden.

Denken Sie zudem daran, die Funktionen vor der Nutzung zu deklarieren! Rufen Sie nun die Initialisierungsmethode in der `main()` auf, um das Spielfeld mit Startwerten zu füllen. Bei einem 12x12-Spielfeld wären ca. 25 Minen passend.

4.2 Die Darstellung

Um den aktuellen Stand des Spieles immer vor Augen zu haben, bedarf es einer geeigneter Visualisierung. Grafische Ausgaben sind noch nicht Teil der Vorlesung gewesen, daher behelfen wir uns mit einer Ausgabe auf der Konsole. Vor jedem Zug soll das aktuelle Spielfeld ausgegeben werden, mit den passenden Indices an der Seite. Zur einheitlichen Darstellung einigen wir uns auf folgende Symbole:

- Noch nicht aufgedeckte Felder werden mit einem * dargestellt
- Aufgedeckte Felder ohne direkt angrenzende Minen erhalten einen einfachen Punkt.
- Für aufgedeckte Felder mit angrenzenden Minen schreiben wir statt dessen die Anzahl der Minen.

Gehen Sie in der Funktion `Darstellung` daher alle Felder des Arrays durch und geben Sie das entsprechende Symbol auf der Konsole aus, um das dargestellte Ergebnis zu erzielen. Erzeugen Sie zudem die Indices an der linken Seite sowie unten, damit später die einzelnen Felder eindeutig adressiert werden können. Schaffen Sie es, die Darstellung variabel von der Spielfeldgröße abhängig zu machen?
Zusätzlich soll die Funktion `darstellen()` noch einen boolschen Parameter erhalten, um als Debug-Ausgabe in das Spielfeld die Minen direkt einzuzeichnen.

Rufen Sie zu Spielstart beide Visualisierungs-Methoden einmal in der `main()` auf.

Ein Beispiel für eine funktionierende Visualisierung könnte zu Spielstart daher so aussehen (links die Ansicht für den Spieler, rechts die Debug-Ansicht mit eingezeichneten Minen):

```
01| * * * * * * * * * * * * *
02| * * * * * * * * * * * * *
03| * * * * * * * * * * * * *
04| * * * * * * * * * * * * *
05| * * * * * * * * * * * * *
06| * * * * * * * * * * * * *
07| * * * * * * * * * * * * *
08| * * * * * * * * * * * * *
09| * * * * * * * * * * * * *
10| * * * * * * * * * * * * *
11| * * * * * * * * * * * * *
12| * * * * * * * * * * * * *
   ---------------------------
    1 2 3 4 5 6 7 8 9 10 11 12
```

```
01| * * * X * * * * * X * * *
02| X * * * * * X * * * * * *
03| X * * * * * X * X X * * *
04| * * * * * * * * X * X X *
05| * * * * * * * * * * * * *
06| X * * * X * * * X * * * *
07| X * * * * * * * * * * * *
08| * * * * X * * * * X X * *
09| * * * * X * X X * * * * *
10| * * * * X X * * * * * * *
11| * * * * * X * * * * * * *
12| * * * * * * * X * * * * *
   ---------------------------
    1 2 3 4 5 6 7 8 9 10 11 12
```

4.3

Das Schlimmste ist geschafft. Nun benötigen wir noch eine Funktion, die sich um die Nutzereingabe kümmert und als Boolean zurückgibt, ob das Spiel weiterläuft (true) oder eine Mine getroffen wurde (false). Nach bekannter Manier soll also der Nutzer aufgefordert werden, die Koordinaten eines Feldes einzugeben (achten Sie auf Falsch-Eingaben). Trifft er eine Mine, Booom. Ansonsten wird das Feld aufgedeckt (also der Belegung-Eintrag des Feldes auf `aufgedeckt` geändert und die Visualisierungsmethode erneut aufgerufen.
Rufen Sie die Eingabe-Funktion innerhalb einer Schleife in der `main()` auf und brechen Sie das Programm ab, sobald eine Mine getroffen wurde.

4.4 Harmlose Nachbarn

Ein Problem bleibt jedoch noch. Wenn Sie in der derzeitigen Version des Spieles so glücklich sind, ein Feld auszuwählen, welches gar keine Minen in der Nachbarschaft hat, wird dieses zwar auf `aufgedeckt` gesetzt. Wie Sie in der Online-Version jedoch bestimmt gesehen haben, werden hier jedoch zusätzlich zeitgleich alle Nachbarn dieses Feldes aufgedeckt. Denn, gibt ein Feld an, keine angrenzende Mine zu haben, können natürlich auch alle 8 Nachbarn gefahrlos aufgedeckt werden. Sollten durch diesen Vorgang wieder Felder ohne Minen-Nachbarn aufgedeckt werden, werden auch von denen wieder alle Nachbarfelder aufgedeckt, und so weiter.
Schreiben Sie daher eine letzte Funktion, die über das gesamte Array läuft und alle Felder sucht, die aufgedeckt sind und keine Minen in der Nachbarschaft haben. Für jedes dieser Felder sollen dann die Nachbarfelder auch aufgedeckt werden, falls sie es nicht schon sind.
Geben Sie als boolean zurück, ob durch den Funktionsaufruf noch Felder aufgedeckt wurden (sich also das Spielfeld noch verändert hat). Dann kann nach jedem Zug des Spielers diese Funktion so oft aufgerufen werden, bis das Spielfeld konstant bleibt.
Zusätzlich soll in der `main()` die Anzahl der noch offenen Felder verwaltet werden, damit als Gewinnbeindung geprüft werden kann, wann alle Felder gezogen wurden (ohne eine Mine zu treffen, versteht sich). Berechnen Sie daher einen Initialwert der offenen Felder und übergeben Sie die Variable per Call by Reference als Funktiopsparameter. Passen Sie dann bei jedem Aufdecken eines zusätzlichen Feldes diesen Wert an, ohne ihn explizit als Return-Wert zurückgeben zu müssen.

Sollte alles funktioniert haben, könnte die Ausgabe wie folgt aussehen:

```
Verbleibende offene Felder: 55
01| * * * * * * * * * * * *
02| * 2 1 1 1 2 * * * * * *
03| * 2 . . . 2 * * * * * *
04| * 1 . . . 1 * * * * * *
05| * 1 . 1 1 1 * * * * * *
06| * 2 . 1 * * * * * * * *
07| * 2 . 2 * * * * * * * *
08| 1 1 . 2 * * * * * * * *
09| . . . 3 * * * * 2 2 2 1
10| . . . 2 * * * * 1 . . .
11| . . . 1 3 * * * 1 . . .
12| . . . . 1 * * * 1 . . .
   ------------------------
    1 2 3 4 5 6 7 8 9 101112

Bitte geben Sie nun Ihren naechsten Zug ein.
Zeile (1-12) _Leerzeichen_ Spalte: (1-12):
10 5

**** BOOOOM ****
Leider verloren.

Verbleibende offene Felder: 54
01| * * * X * * * * * X * *
02| X 2 1 1 1 2 X * * * * *
03| X 2 . . . 2 X * X X * *
04| * 1 . . . 1 * * X * X X
05| * 1 . 1 1 1 * * * * * *
06| X 2 . 1 X * * * X * * *
07| X 2 . 2 * * * * * * * *
08| 1 1 . 2 X * * * * X X *
09| . . . 3 X * X X 2 2 2 1
10| . . . 2 X X * * 1 . . .
11| . . . 1 3 X * * 1 . . .
12| . . . . 1 * * X 1 . . .
   ------------------------
    1 2 3 4 5 6 7 8 9 101112


Process finished with exit code 0
```

4.5 Bonusaufgabe: Eingabe von potentiellen Minen

Im richtigen Minesweeper-Spiel gibt es zudem die Möglichkeit, unaufgedeckte Felder zu markieren, falls diese vermutlich eine Mine enthalten. Erweitern Sie das Spiel, sodass Sie auch Minen tippen können. Markieren Sie diese im Spielfeld durch ein `0`. Bauen Sie zudem einen Timer ein, wie lange der Spieler gebraucht hat, um das Spiel zu gewinnen. Was ist Ihre Highscore?