Lehrstuhl Computergrafik
Prof. Dr.-Ing. Hendrik P.A. Lensch
Patrick Wieschollek, Sebastian Herholz, Fabian Groh

Eberhard Karls
Universität
Tübingen

Informatik II

Übungsblatt 7

Ausgabe: Di, 07. Juni 2016 - **Abgabe: Di, 14. Juni 2016 - 12:00 Uhr**

Aufgaben

7.1 Vier Gewinnt (18 Punkte)

In dieser Aufgabe werden Sie ein vollständiges Spiel implementieren. Eine Übersicht der Klassen befindet sich in Abbildungen 1, 2. Viele Teile sind bereits modular für Sie implementiert. Ihre Aufgabe besteht aus der Vervollständigung der Spiellogik und die Erweiterung der Spielmöglichkeiten mittels Abstraktion und Vererbung. Bei dem Spiel handelt es sich um "Vier gewinnt" und eine davon modifizierte Spielvariante. Das klassische *"Vier gewinnt"* ist ein Spiel zwischen zwei Spielern bei dem abwechselnd ein Stein in einer Spalte fallen gelassen wird, sodass dieser den untersten freien Platz belegt. Gewinner ist der Spieler, welcher es als Erster schafft, vier oder mehr seiner Spielsteine senkrecht, waagerecht oder diagonal in eine Linie zu bringen. Ist das Spielbrett voll, so endet das Spiel unentschieden. Die Spielvariante *Linetris* läuft analog zur klassischen Variante ab. Jede unterste Reihe, die voll belegt ist, ohne dass eine Viererlinie für einen Spieler gebildet wurde, wird geleert.
Die folgende Abbildung veranschaulicht das Umsetzungskonzept vom Spiel. Viele Bestandteile sind bereits implementiert. Sie sollten sich vor dem Programmieren mit den bereits vorhanden Klassen vertraut machen.

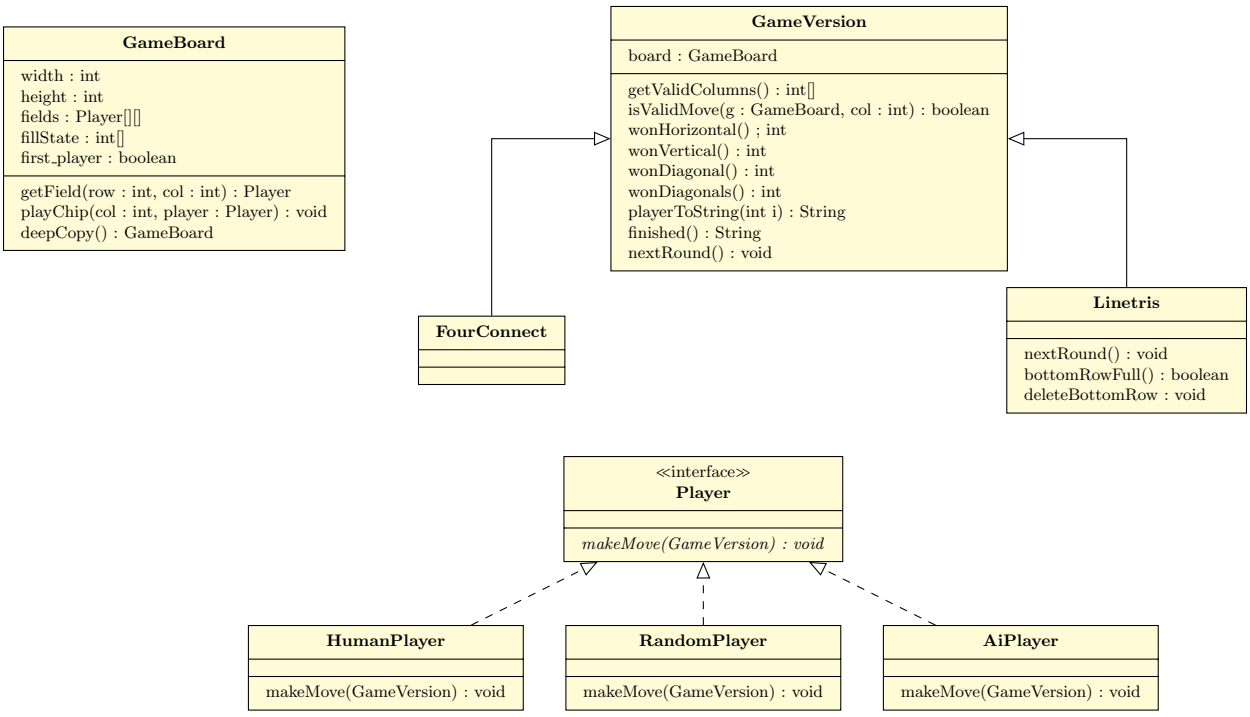

Abbildung 1: Klassenkonzept für "Vier gewinnt".

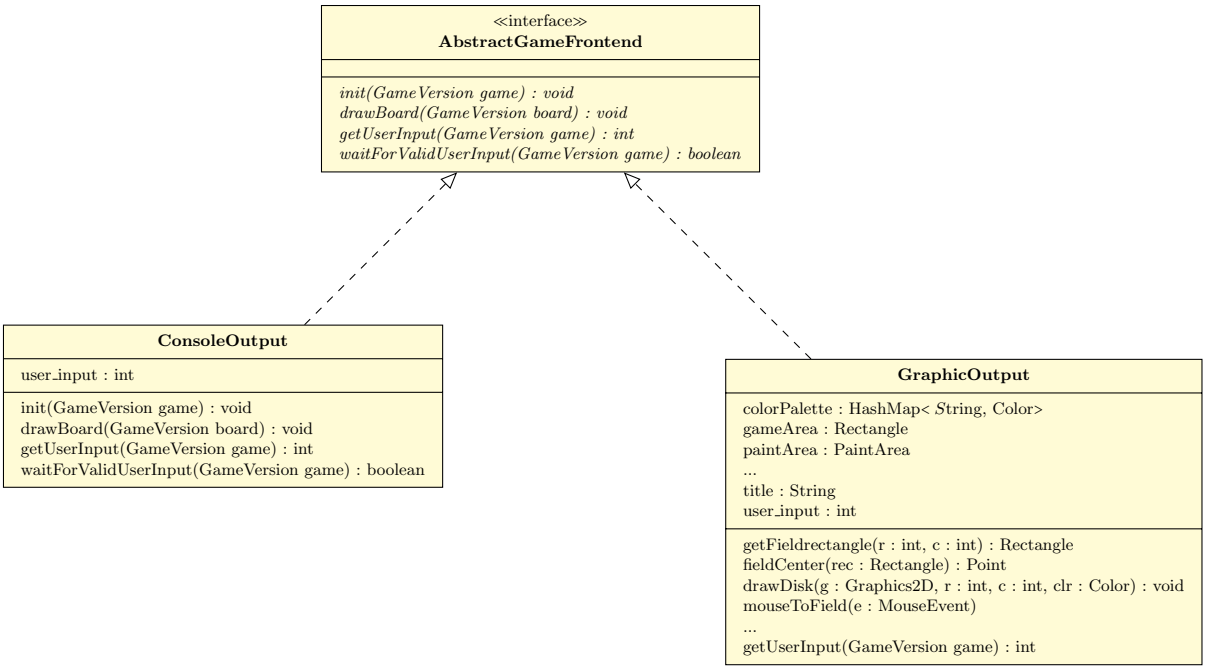

Abbildung 2: Klassenkonzept der Frontends für "Vier gewinnt".

Die Implementation ist in drei Teile gegliedert: Spiellogik, Spielerinteraktion und Frontend für die Ausgabe.

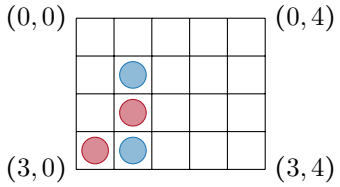

Abbildung 3: Das Feld in der linken unteren Ecke hat die Koordinaten $(3, 0)$. Wir verwenden stets (r, c), wobei r die Reihe und c die Spalte bezeichnet.

Spielogik Die Spiellogik verwaltet die aktuelle Spielsituation und testet auf die Spielenden.

a) In der Vorlage befindet sich die Klasse *GameBoard*. Es sind drei Methoden zu vervollständigen: `getField, playChip` und `deepCopy`. Die Methode `getField` die Belegung vom Feld an der Position (r, c) zurückliefern. Mittels der Methode `playChip(c,p)` soll in Spalte c der Chip vom Spieler p fallen gelassen werden. Beachten Sie, dass der Chip bis auf den obersten vorhandenen Chip bzw. die unterste Reihe herunterfällt! Um die komplette Spielsituation zu kopieren, wird eine Methode `deepCopy` benötigt, welche eine neue Instanz vom Typ *GameBoard* als exakte Kopie zurückliefern.

b) Der Spielmodi wird in der Eltern-Klasse *GameVersion* verwaltet. Hier sind die Methoden `getValidColumns` und `isValidMove` zu vervollständigen, wobei erstere ein Array der Länge `width` zurückliefert und eine `1` bzw. `0` für jede erlaubte bzw. ungültige Zugspalte repräsentiert. Die Methode `isValidMove` überprüft, ob ein Spielzug in der entsprechenden Spalte ausgeführt werden darf. Um auf das Spielende zu testen, muss lediglich die Methode `wonHorizontal` ausgefüllt werden. Orientieren Sie sich an den bereits implementierten Methoden `wonVertical, wonDiagonals`.

Frontend für Ausgabe Für das Debugging ist es oft hilfreich statt der 2D-Grafikausgabe eine Ausgabe auf der Konsole zu verwenden. Diese werden Sie in den folgenden Teilschritten umsetzen. Wie Sie aus

dem Interface *AbstractGameFrontend* entnehmen können, muss jedes Frontend in dieser Implementation vier Methoden bereitstellen. Erstellen Sie eine Klasse *ConsoleOutput*, welches *AbstractGameFrontend* implementiert.

c) Die Klasse benötigt ein privates Feld `user_input` vom Typ `int`, welches die Benutzereingabe für den nächsten Zug abspeichert. Initialisieren Sie dieses Feld mit −1.

d) Die Methode `drawBoard` soll das Spielfeld mittels ASCII darstellen. Geben Sie `x,o` bzw. `.` für Spieler 1,2 bzw. ein leeres Feld aus.

e) Die Methode `getUserInput` liefert lediglich `user_input` zurück und setzt dieses danach auf wieder auf −1.

f) Mittels `waitForValidUserInput` wird solange per *Scanner* einer Spalte für einen Zug gefragt, bis die Eingabe einem gültigen Zug entspricht.

Wenn Sie nun in der `Main.java` die Klasse vom Typ *AbstractGameFrontend* mit einem Objekt vom Typ *ConsoleOutput* initialisieren, so können Sie auf der Konsole spielen.

Computergegner Wie Sie aus der Vorlage entnehmen können, ist momentan nur die Möglichkeit gegeben, dass zwei menschliche Spieler *HumanPlayer* gegeneinander antreten. Analog zur vorherigen Teilaufgabe wollen wir in den folgenden Schritten einen einfachen Computergegner erstellen. Gehen Sie dazu wie folgt vor:

g) Um einen weiteren Spieler zu implementieren ist es hilfreich eine Schnittstelle (Interface) vorzubereiten. Für das Frontend hatten wir das bereits mittels *AbstractGameFrontend* für Sie übernommen. Für die Schnittstelle von Spielern müssen Sie selbst Hand anlegen. Erstellen Sie dieses Interface *Player* mit der Methode `makeMove`, welche kompatibel zur entsprechenden Methode aus *HumanPlayer* ist und passen Sie in der `Main.java` die entsprechenden Stellen ebenfalls an, indem Sie die Typen *HumanPlayer* mit der Interface-Bezeichnung ersetzen. Vergessen Sie nicht, das *HumanPlayer* nun ebenfalls die Schnittstelle implementiert.

h) Nun kann eine weitere Klasse *RandomPlayer* angelegt werden. Dazu können Sie die Klasse *HumanPlayer* kopieren. Lediglich die Methode `makeMove` soll anstatt der Benutzerinteraktion, einen zufälligen gültigen (!) Zug zurückgeben.

i) Um den Spieler ins Spiel aufzunehmen, sollte in der `Main.java` ein Spieler als *RandomPlayer* initialisiert werden.

Sollten Sie alle Teilschritte gelöst haben, so kann in der kommenden Woche im Helpdesk besprochen werden, wie man eine kleine intelligentere AI als Computergegner entwickelt. Alle notwendigen Funktionen haben Sie mit diesem Übungsblatt dann bereits vorbereitet.

Spielmodi Da die klassische Spielvariante von *"Vier gewinnt"* mit der Zeit etwas langweilig werden könnte, wollen wir eine weitere Spielvariante *"Linetris"* hinzufügen.

j) Analog zur Klasse *FourConnect* sollen Sie eine Klasse *Linetris* erstellen. Kopieren Sie dazu einfach die Klasse *FourConnect*. Passen Sie den Quelltext an und überschreiben Sie die gerbte Methode `nextRound`. Diese wird bereits in der `Main.java` nach jedem Spielzug aufgerufen. Ihre Aufgabe besteht darin, diese Methode zu nutzen, um die unterste Reihe zu entfernen, falls diese voll belegt ist.

7.2 Countdown (12 Punkte)

In dieser Aufgabe werden Sie mit *Swing* eine kleine Fensteranwendung erstellen, die als Kurzzeitwecker fungiert. Diese beinhaltet drei Komponente, ein Panel für das Anzeigen der verbleibenden Zeit im Format `mm:ss` und ein Button `start`, sowie ein Button `reset` . Gehen Sie wie folgt vor:

a) Machen Sie sich mit der Klasse *javax.swing.Timer* vertraut.

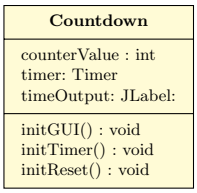

Abbildung 4: Klassenkonzept für "Countdown".

b) Erstellen Sie eine Klasse mit der Bezeichnung *Countdown*. Diese soll für eine vorgegebene verbleibende Zeit (counterValue) die entsprechende Ausgabe in *timeOutput* anzeigen.

c) Erstellen Sie eine Methode `initGUI`, welche die Fensteranwendung erstellt.

d) Erstellen Sie eine Methode `initTimer`, welche beim drücken des Button `start` aufgerufen wird und den Countdown beginnen lässt.

e) Erstellen Sie eine Methode `initReset`, welche beim drücken des Button `reset` aufgerufen wird und den timeOutput zurücksetzt auf den Anfangswert.

Hinweis:

Benutzen Sie ActionListener. Achten Sie auf die Richtigkeit bei mehrfachen Drücken des Buttons. Bei Problem diesbezüglich können Sie gerne in die Sprechstunde kommen.