Elementare Algorithmen und Objektorientierte Programmierung

Übung 8

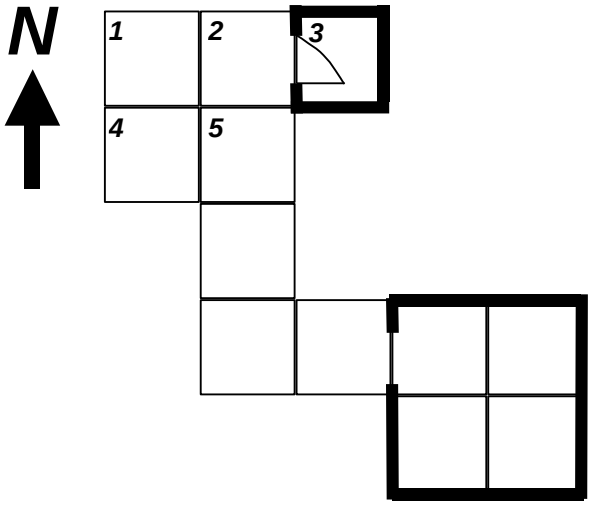
Auszuarbeiten bis 17.06.21

Noch vor dem Durchbruch des Internets, existierten schon Spiele, die online von Spielern aus aller Welt gemeinsam gespielt werden konnten. Über universitäre Netze konnten Studenten und andere Personen, die Zugang zu diesen Netzen hatten, in fiktiven, rein textuell dargestellten Welten *Quests* erfüllen, neue Gegenden erforschen, sich gegenseitig Kämpfe liefern usw.

Diese, unter dem Namen Multi-User-Dungeons (MUD) bekannten Spiele, basierten (und basieren noch immer) fast ausschließlich auf Objekt-Technologie und Sprachen wie C++ . An dieser Stelle wird es für uns interessant!

1. Mini-MUD (20 Punkte)

a) Entwerfen und implementieren Sie ein einfaches Spielfeld, das aus einer beliebigen Anzahl von Feldern besteht. Jedes Feld kennt seine direkten Nachbarn, die in den Haupthimmelsrichtungen (Nord, Ost, Süd und West) angeordnet sein können. Mögliche Nachbarn sind andere Felder, Wände und Türen zu anderen Feldern.



Neben einer Kurzbeschreibung und einer detaillierten Beschreibung wird jedes Feld über eine Nummer eindeutig identifiziert.

Jedes Element des Spielfelds muss mindestens die Aktion `Enter` anbieten, deren Bedeutung natürlich vom jeweiligen Element abhängt. Beim „Betreten“ eines Feldes ändert sich z.B. die aktuelle Position des Spielers. `Enter` für eine Tür bedeutet, dass man sich von Feld *A* zu Feld *B* bewegt, sofern die Türe geöffnet ist, usw.

Zusätzlich soll es möglich sein, die **Beschreibung eines Feld** und die möglichen Richtungen, in die man sich weiterbewegen kann, **abzufragen.**

Dokumentieren Sie Ihr Design in Form eines **UML-Klassendiagramms**. Zusätzlich soll das „Bewegen“ als **Sequenzdiagramm** dargestellt werden. Beachten Sie dabei auch die Sonderfälle

Um das Testen des Spielfeldes zu erleichtern, ist eine Funktion zu realisieren, die die gesamte **Spielfelddefinition ausgibt**. Verwenden Sie dazu das folgende Format:

```
...
-------------------------------------------------
Field: 2, Church forecourt
Description: ....

North: NONE
East: Door to Field 3, Local Church
South: Field 5
West: Field 1
-------------------------------------------------
Field: 3, Local church
Description: .....


North: Wall
East: Wall
South: Wall
West: Door to field 2, Church forecourt
-------------------------------------------------
...
```

b) Nachdem bereits das Spielfeld realisiert wurde, soll nun die folgende Funktionalität bereitgestellt werden, um das Mini-MUD auch wirklich spielen zu können:

- Neuer Spieler betritt am Startfeld das Spiel
- Spieler verlässt das Spiel
- Spieler bewegt sich auf dem Spielfeld (abhängig von den möglichen Richtungen der aktuellen Position)
- Es gibt Gegenstände (mindestens vier), die auf den Feldern herumliegen können, die vom Spieler mitgenommen bzw. angezogen werden können. Auch Nahrung und. Getränke können konsumiert werden.
- Spieler möchte Information zur aktuellen Position. d.h. Beschreibung des Feldes, mögliche Richtungen, in die sich der Spieler weiterbewegen kann, **andere Spieler**, die sich am gleichen Ort befinden und auch Gegenstände die „herumliegen“

Überlegen Sie sich gut, welche Klassen benötigt werden und welche Beziehungen zwischen diesen bestehen!

Hinweis:

Achten Sie beim Entwurf darauf, dass das System einfach erweitert werden kann!

1 Mini-MUD

Lösungsidee

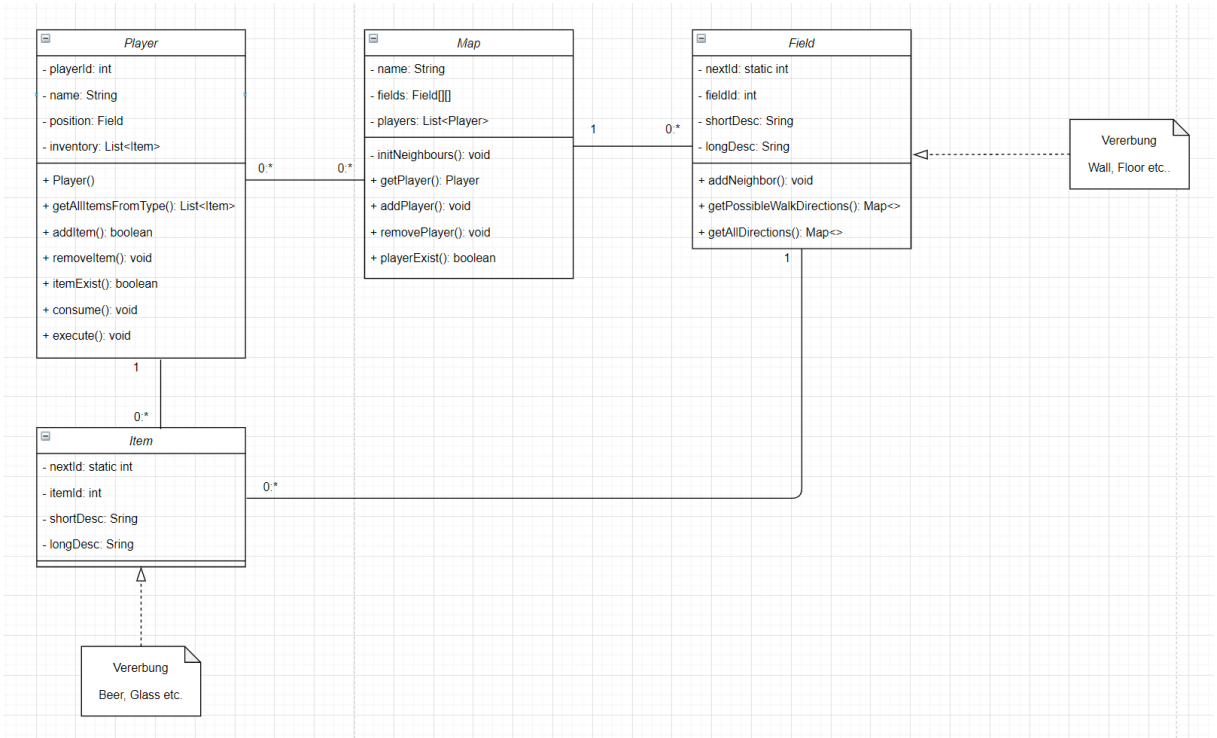
Ziel ist es eine Map aufzubauen und darin spieler spawnen zu lassen. Der Spieler kann sich innerhalb der map bewegen, items aufnehmen und in andere Räume eintreten. Für manche Räume braucht man einen Schlüssel. Ein Feld halt 1 bis 4 nachbaren. Nur jenes Feld ist betretbar, dass das Interface **IAccessable** hat. Nur bei Felder die das Interface **IExecuteable** haben, kann was gemacht weden und nur die Items die das Interface **IConsumable** haben können consumiert werden.

Die map wird in so einem Schema aufgebaut.
W --> Wall F --> Floor I --> ItemField D --> Door

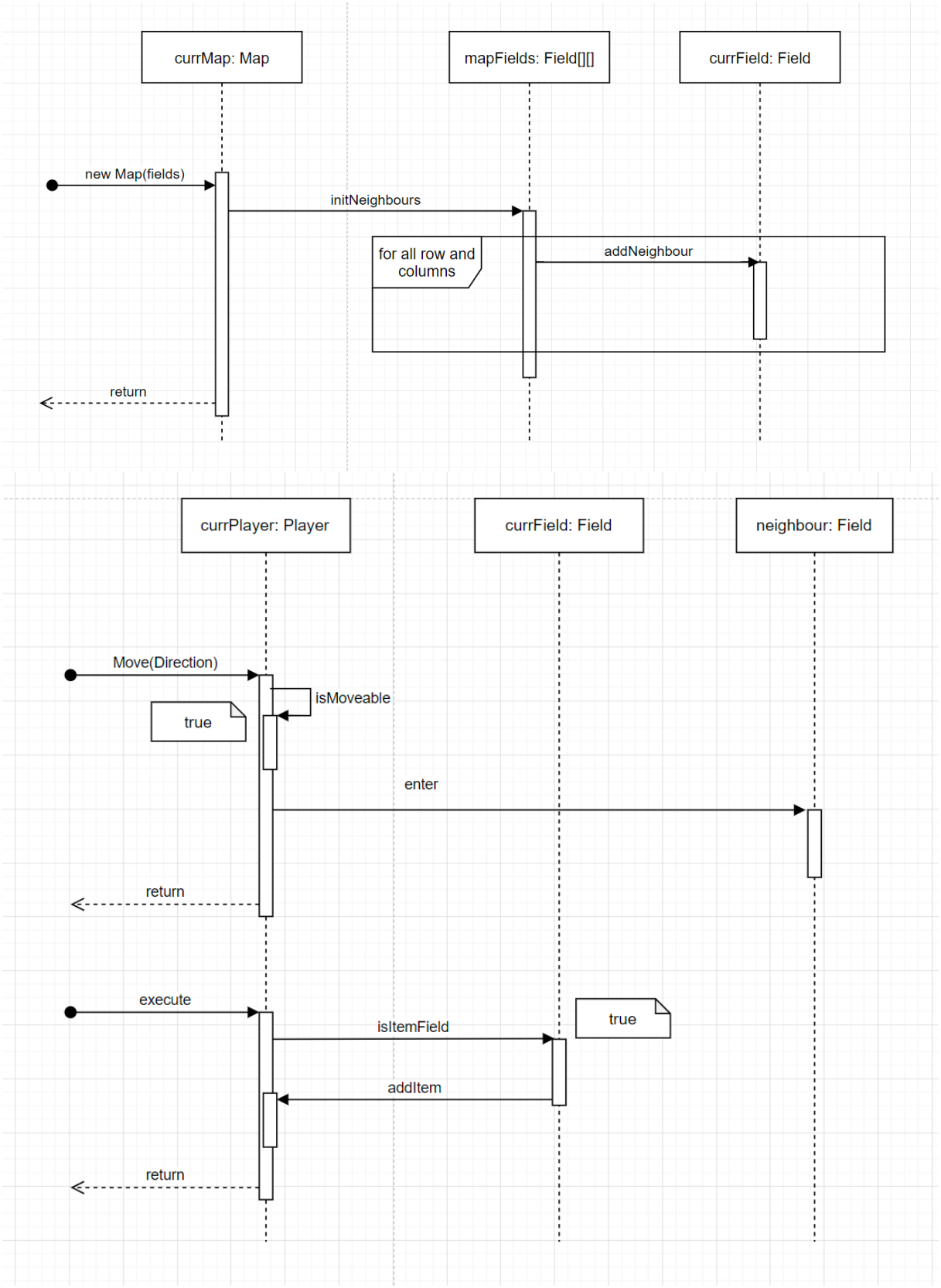
```
char[][] mapScheme = new char[][]{
        {'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W'},
        {'W', 'F', 'F', 'W', 'W', 'W', 'W', 'F', 'F', 'W'},
        {'W', 'I', 'F', 'W', 'I', 'F', 'W', 'F', 'F', 'W'},
        {'W', 'W', 'D', 'W', 'W', 'F', 'W', 'F', 'F', 'W'},
        {'W', 'F', 'F', 'F', 'W', 'F', 'W', 'F', 'F', 'W'},
        {'W', 'F', 'F', 'F', 'W', 'F', 'W', 'F', 'F', 'W'},
        {'W', 'F', 'F', 'F', 'W', 'F', 'W', 'F', 'F', 'W'},
        {'W', 'F', 'F', 'F', 'D', 'F', 'W', 'F', 'F', 'W'},
        {'W', 'I', 'F', 'F', 'W', 'F', 'D', 'F', 'F', 'W'},
        {'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W'},
};
```

a) Spielfeldentwurf mit UML und Sequenzdiagramm

UML-Klassendiagramms



Sequenzdiagramm



b) Code

Siehe beigelegte files.

Testfälle:
siehe test package.