Universität Leipzig
Institut für Informatik

WS 19/20

Modellierung und Programmierung 1 – Übungsserie 5

Abgabetermin: 15.01.2020, 23:59 Uhr
Abgabeformat: 1 ZIP-Datei
Max. Punkte: 32

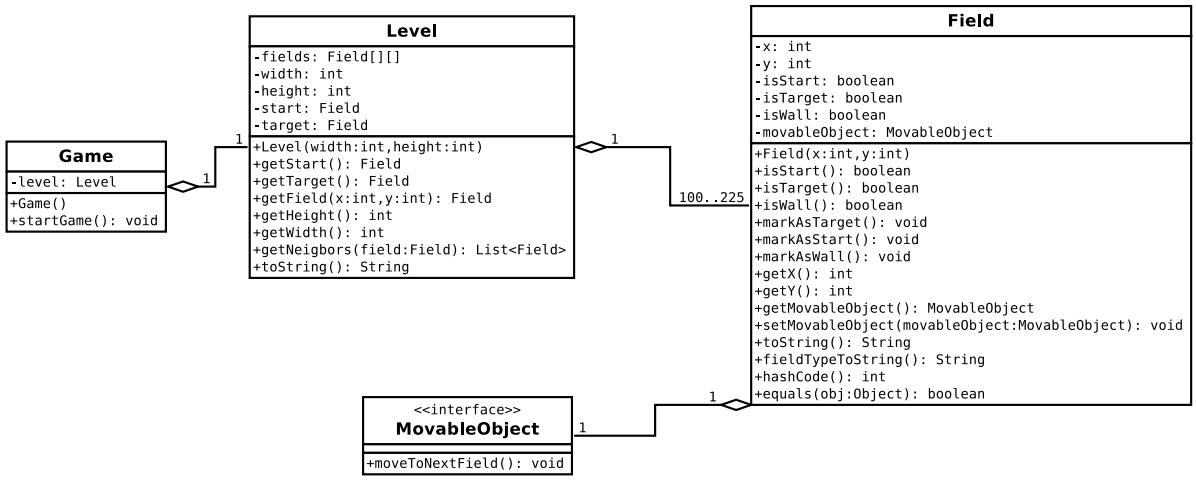
Java Programmierung - Labyrinthsimulation (Rekursion, Interfaces)

Ein kleines Entwicklungsstudio möchte ein neues 2D Top-Down Hack-And-Slay Spiel entwickeln.

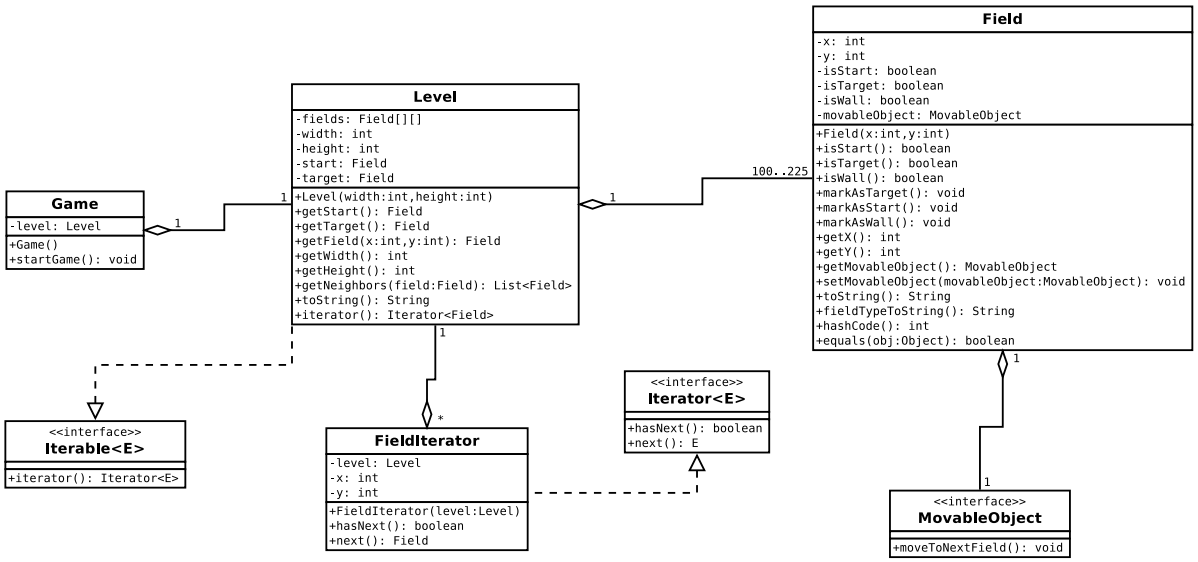
```
Step 11
WWWWWWWWWWW
WW  W   W W
W  W    WWW
W W    W  W
WWW WW    W
W W     WEW
WP      W W
W   W W   W
WWWWW    WW
W W   W   W
W   E   W W
W  W S   WW
WW W      W
W    W    W
WWWWWWWWWWW

Player has won the game!
```

Das Studio steht aber noch am Anfang der Entwicklung. Als selbständiger Entwickler, sollen Sie es dabei unterstützen. Im Moment sind folgende Klassen als UML und Java-Quellcode (von einem vorherigen ähnlichen Softwareprojekt) gegeben:



1. Iterable `(8 Punkte)` Zu Beginn soll die Klasse `Level` um einen Iterator erweitert werden, mit welchem man über alle Felder iterieren kann. `Level` speichert ein zweidimensionales Array von Objekten der Klasse `Field`.



Erstellen Sie die Klasse `FieldIterator`, welche sich intern die Position des gegenwärtigen Feldes speichert. Die Klasse muss dabei das Interface `Iterator` implementieren. Implementieren Sie die `hasNext` und die `next` Methoden so, dass `FieldIterator` über jedes Feld des Levels genau einmal iteriert. Klasse `Level` soll anschließend das Interface `Iterable` implementieren und dabei den neuen `FieldIterator` verwenden. *(8 Punkte)*

2. Levelverfeinerung `(16 Punkte)` Gegenwärtig sind alle Felder, welche am Rand liegen immer als Mauerfelder markiert, so dass ein Spieler niemals aus dem Level laufen kann. Als nächsten Punkt wollen die Entwickler dem Level zufällig Mauern hinzufügen, um die Wegsuche zu erschweren. Dann soll mittels eines rekursiven Algorithmus überprüft werden, ob ein Level überhaupt „lösbar" ist, d.h. ob ein Spieler auf einem Pfad vom Start zum Ziel kommen kann oder nicht. Die ToDo-Liste für die Abarbeitung dieser Aufgabe sieht wie folgt aus:

   a) Implementieren Sie in der Klasse `Game`, die Methode `addWalls`, welches dem Level weitere Mauern hinzufügt. Dabei soll über alle Felder des Levels iteriert werden, z.B. mittels des in der vorherigen Aufgabe gebauten Iterators. Jedes Feld, welches weder Start-, Ziel- noch Mauerfeld ist, soll mit Wahrscheinlichkeit $\frac{1}{3}$ ein Mauerfeld werden. *(3 Punkte)*

   b) Erweitern Sie das `Level` um folgende Methoden:

      - `public boolean existsPathFromAToB(Field a, Field b)`
      - `public List<Field> findPathFromAToB(Field a, Field b)`
      - `private List<Field> findPathFromAToB(Field a, Field b, Collection<Field> markedFields)`

      Die Methode `findPathFromAToB` soll `null` zurückgeben, falls es keinen Weg von a nach b gibt, ansonsten soll sie eine Liste von Feldern vom a nach b (inklusive) zurückgeben. Der Weg darf natürliche keine Mauern enthalten. Die Implementierung soll für alle Nachbarfelder n von b (die mit `getNeighbors` ermittelt werden können) rekursiv einen Weg von a nach n ermitteln (falls ein solcher existiert), und in diesem Fall b zu dem Weg am Ende hinzufügen. Bereits besuchte Felder sollen markiert werden, damit Felder nicht mehrfach durchsucht werden. Methode `existsPathFromAToB` soll `true` genau dann zurückgeben, wenn einen Weg von a nach b existiert *(10 Punkte)*.

      Hinweis: Die öffentliche `findPathFromAToB` startet nur die eigentliche Rekursion (in der privaten `findPathFromAToB`) mit geeigneter Initialisierung von `markedFields`. Zudem stellt im Allgemeinen der Weg von a nach b nur einen Teil der markierten Felder dar, daher sollte für Markierung und Ergebnis (Weg) verschiedene Objekte verwendet werden.

   c) Ändern Sie den Konstruktor der Klasse `Game` so ab, dass das Level verfeinert wird, solange kein Pfad zwischen Start und Ziel existiert *(3 Punkte)*.

3. Levelbevölkerung `(8 Punkte)` Damit das Level nicht nur aus Mauern besteht, wollen die Entwickler das Level mit Lebewesen bevölkern. Lebewesen unterscheiden sich dabei in Spieler und Gegner. Ein Spieler will immer vom Start zum Ziel gelangen. Trifft er dabei auf einen Gegner, stirbt er und fängt wieder vom Start aus an. Die Gegner bewegen sich zufällig umher. Helfen Sie ihnen dabei auf folgende Art und Weise:

   a) Die Klasse `Player` soll den Pfad des Spielers in einem Attribut speichern. Zu Beginn soll der abgespeicherte Pfad der Weg vom Start zum Ziel sein. *(2 Punkte)*.

   b) Implementieren Sie dann die Methode `respawnPlayer` der Klasse `Player`, welche den Spieler zurück auf den Start setzen soll und seinen aktuellen Fortschritt auf dem Pfad vom Start zum Ziel zurücksetzt *(2 Punkte)*.

   c) Implementieren Sie in der Klasse `Player` die Methode `moveToNextField`. Diese soll, solange der Spieler noch nicht beim Ziel angekommen ist, pro Aufruf der Methode immer das nächste Feld aus dem gespeicherten Pfad wählen. Sollte ein Gegner auf diesem Feld stehen, dann muss der Spieler vom Start erneut beginnen. Sonst geht er auf das neue Feld *(4 Punkte)*.

Abschließend kann das Spiel nun simuliert werden. Das Entwicklungsstudio ist Ihnen sehr dankbar für Ihre Hilfe und überlässt Ihnen den Quellcode.