Universität Stuttgart

Institut für Visualisierung
und Interaktive Systeme

Übungsblatt 5

Datenstrukturen und Algorithmen (SS 2016)

Abgabe: Mittwoch, 25.05.2016, 23:59 Uhr — Besprechung: ab Montag, 30.05.2016

Bitte lösen Sie die Übungsaufgabe in **Gruppen von 3 Studenten** und wählen EINEN Studenten aus, welcher die Lösung im ILIAS als **PDF** als **Gruppenabgabe** (unter Angabe aller Gruppenmitglieder) einstellt. Bitte erstellen Sie dazu ein **Titelblatt**, welches die Namen der Studenten, die Matrikelnummern, und die E-Mail-Adressen enthält.

Die Aufgaben mit Implementierung sind mit Impl gekennzeichnet. Das entsprechende Eclipse-Projekt kann im ILIAS heruntergeladen werden. Bitte beachten Sie die Hinweise zu den Implementierungsaufgaben, die im ILIAS verfügbar sind.[1]

Dieses Übungsblatt beinhaltet 3 Aufgaben mit einer Gesamtzahl von 30 Punkten.

Aufgabe 1 Impl Quadtree *[Punkte: 10]*

Quadtrees entsprechen einer weit verbreiteten und oft eingesetzten räumlichen Datenstruktur, welche eine effiziente Speicherung sowie das Auffinden von Objekten unterstützen. Die Generierung eines Quadtrees sowie die Implementierung einer Suchanfrage ist daher Bestandteil dieser Aufgabe. Vervollständigen Sie den Quelltext in den Dateien *Quadtree.java* und *AQuadtree.java*. Die jeweiligen Stellen sind mit *TODO Insert code for Assignment x.x [a,b,...]* markiert.

(a) *(6 Punkte)* In dieser ersten Teilaufgabe soll der Quadtree gemäß der in der Vorlesung vorgestellten Methode erzeugt werden. Ausgehend von einer Liste an Objekten, werden die Objekte der Liste auf vier Kindknoten verteilt, falls sich mehr als k Objekte in der Liste befinden. Daher hat jeder Knoten entweder genau vier Kinder (interner Baumknoten) oder keine (Blattknoten). Gehen Sie davon aus, dass die Objekte der Liste eindeutige Positionen besitzen und maximal einmal vorkommen. Die Aufteilung der Objekte zu den Kindknoten geschieht anhand von Positionsinformationen. Jeder Knoten enthält nur Objekte die sich in dem ihm zugeordneten Bereich befinden. Jeder Kindknoten überwacht einen Teil des Bereichs seines Elternknotens. Der Bereich des Elternknotens wird daher in vier gleichgroße Teile aufgeteilt. Es wird zuerst versucht ein Objekt in den Bereich oben links, dann oberen rechts, gefolgt von unten links und dann rechts zuzuordnen. Um zu prüfen ob sich ein Punkt in einem Rechteck befindet, können Sie die *contains()* Funktion der Klasse *Rect* verwenden.

(b) *(4 Punkte)* Als nächstes soll nun eine räumliche Suchanfrage (range query) implementiert werden. Dabei wird ein rechteckiger Bereich vorgegeben (Suchbereich), in dem nach Objekten gesucht werden soll. Suchanfragen werden nur dann an Kindknoten propagiert, wenn diese einen Teil des Suchbereichs abdecken.

Aufgabe 2 Impl Kollisionsabfrage mithilfe einer Gitterstruktur *[Punkte: 12]*

In dieser Aufgabe soll eine Gitterstruktur genutzt werden, um Objekte zu verwalten und Kollisionen zu ermitteln. Als Objekte werden hier ausschließlich Rechtecke genutzt. Die ersten beiden Teilaufgaben beziehen sich auf die Implementierung einiger Funktionen der Klasse *Rect*. Diese Teile sind unabhängig von der Gitterstruktur. Diese implementierten Funktionen werden dann in Teilaufgabe 3 und 4 genutzt. Vervollständigen Sie daher den Quelltext in den Dateien *Rect.java* und *CollisionMap.java*. Die jeweiligen Stellen sind mit *TODO Insert code for Assignment x.x [a,b,...]* markiert.

(a) *(2 Punkte)* Implementieren Sie in der Klasse *Rect* die Methode *coveredArea()*. Diese Methode soll eine Liste an ganzzahligen Gitterzellen zurückliefern, welche durch das Rechteck ganz oder teilweise belegt werden.

[1] `https://ilias3.uni-stuttgart.de/goto_Uni_Stuttgart_fold_997779.html`

(b) *(2 Punkte)* Implementieren Sie in der Klasse *Rect* die Methode *collisionWith()*. Diese Funktion soll *true* zurückliefern, genau dann wenn das aktuelle Rechteck das Rechteck, das übergeben wurde, schneidet. Ansonsten soll der Rückgabewert *false* sein.

(c) *(3 Punkte)* Implementieren Sie in der Klasse *CollisionMap* die Methode *fillCollisionMap()*. Diese Funktion soll an der Stelle *(x,y)* in *gridCollisionMap* Referenzen auf Objekte einfügen, welche die Stelle *(x,y)* ebenfalls überdecken. Für das Einfügen einer Objektreferenz steht Ihnen die Funktion *insertCollisionObject()* zu Verfügung und für das Auslesen dieser ist die Funktion *getObstaclesAtPoint()* vorgesehen.

(d) *(3 Punkte)* Implementieren Sie in der Klasse *CollisionMap* die Methode *getCollisionCandidates()*. Diese Funktion soll eine Menge an Objekten zurücklieferen, mit denen das übergebene Objekt möglicherweise kollidieren könnte.

(e) *(2 Punkte)* Welche Vor- und Nachteile ergeben sich bezüglich des Speicherplatzes und Suchanfragen bei der Verwendung von Quadtrees im Vergleich zu einem Gitter-basierten Ansatz?

Aufgabe 3 BSP-Baum *[Punkte: 8]*

In der dargestellten Szene (Abbildung 1) befinden sich acht Objekte (1-8), die durch einen BSP-Baum unterteilt werden sollen. Hierbei sollen achsen-orientierte Hyperebenen verwendet werden, um den Raum aufzuteilen, bis sich jedes Objekt in einem einzelnen Blatt befindet. Die Ebenen werden jeweils mit einem Kleinbuchstaben versehen. Ein Richtungspfeil legt die linke Seite der Ebene fest. Jede Ebene soll die Anzahl der Objekte in dem entsprechenden Bereich halbieren. Hierbei gilt es folgendes Schema einzuhalten:

- Horizontale Ebenen haben einen Richtungspfeil nach oben.
- Vertikale Ebenen haben einen Richtungspfeil nach links.

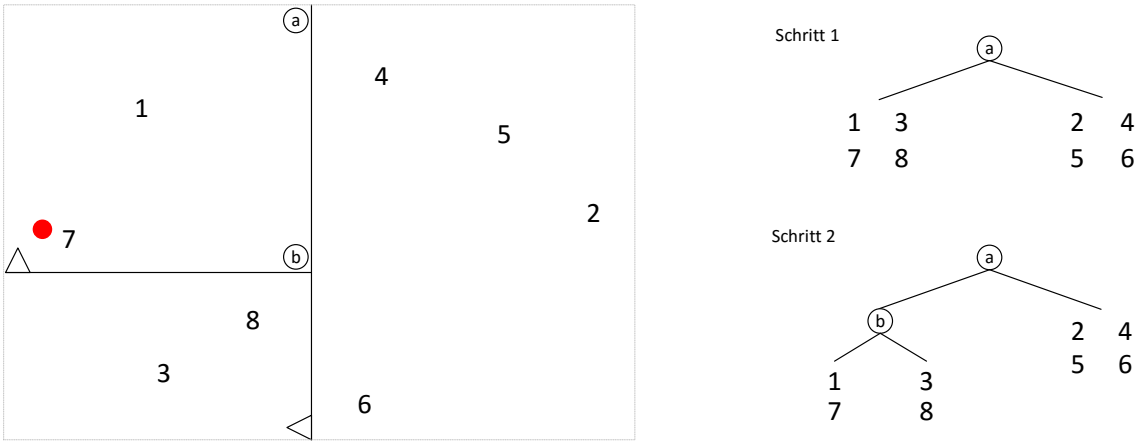

Abbildung 1: Szene mit 8 Objekten für die Erstellung eines BSP-Baumes.

(a) *(2 Punkte)* Fügen Sie in das linke Bild weitere Ebenen mit dem vorgegebenen Beschriftungsschema ein, bis sich jedes Objekt in einem separaten Bereich befindet.

(b) *(2 Punkte)* Das rechte Bild zeigt zwei Schritte für die Erstellung des BSP-Baums. In jedem Schritt wird eine neue Ebene in den Baum eingefügt und die Objekte entsprechend aufgeteilt. Zeichnen Sie alle Schritte auf, die durch das Einfügen der neuen Ebenen den Baum verändern (pro Schritt ein Baum). Die eingefügten Ebenen sollten bezüglich Struktur und Reihenfolge Ihrer Lösung aus Aufgabenteil a) entsprechen.

(c) *(2 Punkte)* Der BSP-Baum kann beispielsweise zur effizienten Schnittberechnung von Objekten verwendet werden. Geben Sie für den roten Kreis die Traversierung des Baumes an, bis das potentielle Schnittobjekt gefunden wurde. Wie viele Vergleichsoperationen sind nötig um das Objekt zu finden? Geben Sie in O-Notation an, wie viele Schritte im *worst case* und im *best case* nötig sind, um ein beliebiges Schnittobjekt in einem nach den in der Aufgabe gegebenen Konstruktionsregeln erzeugten BSP-Baum zu finden, wobei n die Anzahl der Objekte ist. Begründen Sie Ihre Antwort.

(d) *(2 Punkte)* In dieser Aufgabe wurden einige Konstruktionsregeln zur Erzeugung des BSP-Baums vorgegeben. Wie verändern sich die *worst case*- und *best case*-Abschätzung, wenn anstatt der Regel, dass jede Ebene die Anzahl der Elemente halbiert, nur vorgegeben ist, dass jeder Knoten mindestens ein Element enthalten muss? Geben Sie Ihre Abschätzung in O-Notation an, wobei n erneut die Anzahl der Objekte ist und begründen Sie Ihre Antwort.