

# Laborationsrapport

## D0041D Objektorienterad Programmering Laboration 2

### Handledare:

Gustav Sternbrant

### Hans Granlund

**hangra-4@student.ltu.se**

Luleå Tekniska Universitet

Skellefteå

2016-02-05

### Pontus Stenlund

**ponste-5@student.ltu.se**

Luleå Tekniska Universitet

Skellefteå

2016-02-05

## Innehållsförteckning

<b>Problemspecifikation .....</b>	<b>3</b>
<b>Användarhandledning.....</b>	<b>4</b>
<b>Algoritmbeskrivning .....</b>	<b>5</b>
<b>Systembeskrivning .....</b>	<b>6</b>
<b>Lösningens begränsningar.....</b>	<b>8</b>
<b>Problem och Reflektioner .....</b>	<b>9</b>
<b>Diskussioner .....</b>	<b>10</b>
<b>Testkörningar .....</b>	<b>11</b>

## Problemspecifikation

I denna laboration så skulle vi implementera ett QuadTree som en template klass. Denna datastruktur fungerar likt ett binärt träd där varje nod har barn. Skillnaden är att QuadTree har fyra barn och ett vanligt binärt träd har två. Varje barn representerar en indelning av en kvadrat i fyra mindre kvadrater.

För att hålla reda på data vi lagrar i QuadTree så måste varje element (nod) ha en X,Y koordinat.

När vi stoppar in ett element i QuadTree så avgörs vilken cell elementet hamnar i och om denna cell har uppnått sin fulla kapacitet så delas denna cell upp i fyra mindre celler och elementet stoppas in i ett av dem.

Vi skulle även implementera en klass för elementen, *QuadTreeElement*.

Denna klass skulle även vara en template klass.

I den andra delen skulle vi bara implementera ett par stycken makron som gjorde att vi kunde använda oss av Gustavs grafiska gränssnitt för att få en visuell representation på våra träd.

## Användarhandledning

För vår lösning så skapar vi ett objekt av QuadTree genom att skriva:  
`QuadTree<QuadTreeElement<"typ">> "namn"`.

Sedan måste man anropa `QuadTree.Setup(float width, float height)`. Här sätter vi dimensionerna på kvadraten som QuadTree kommer att vara i.

Sedan kan man skapa noder genom att skriva `QuadTree<"typ"> "namn"` (float x, float y, value) och sedan anropa `QuadTree.InsertElement(QuadTreeElement)`.

Eller så skriver anropar man `QuadTree.InsertElement(QuadTreeElement<"typ">(float x, float y, value))` för att lägga till ett element i trädet.

För att hitta data som är lagrat i trädet så anropar man `QuadTree.FindElements(float x, float y, float width, float height, std::vector<"typ">& output)`. Argumenten som man anger i denna funktion blir en sökruta som förhoppningsvis överlappar celler i trädet. Om sökrutan överlappar trädet någonstans så letar vi reda på element och lagrar noderna i vektorn som vi skickar in som argument.

## Algoritmbeskrivning

Vi bygger upp ett träd likt ett binärt träd. Men istället för att varje nod kan ha två barn så kan QuadTree ha fyra barn. Detta innebär att djupet på trädet blir mycket mindre på QuadTree. Så vi behöver inte gå igenom lika många noder när vi letar efter värde.

I trädet så sparas noderna i en vektor upp till en maxstorlek som är satt till fyra. När vektorn är full så skapar vi fyra nya objekt av QuadTree och sätter pekare till de nya objekten. Och varje objekt kan ha fyra noder.

Och sedan när något av de nya objekten är fulla så skapar vi nya träd och sätter pekare.

## Systembeskrivning

Vi har även en template klass QuadTree och en struct QuadTreeElement som innehåller följande metoder:

### QuadTree:

- QuadTree(), default konstruktör som sätter startpunkten på trädet till 0,0 och trädet har inga dimensioner
- QuadTree(float x, float y), konstruktör som sätter startpunkten på trädet.
- Setup(float width, float height), metod som sätter dimensionerna på trädet.
- Split(), delar upp trädet i fyra mindre träd.
- InsertElement(const T& element), metoden lägger till QuadTreeElement till trädet. Om trädet redan har fyra element så anropas Split() för att dela upp trädet och sedan hittar metoden vilket av de fyra nya träden elementet skall stoppas in.
- FindElements(float x, float y, float width, float height, std::vector<T>& output). Man anger ett område som man skall söka av. Metoden kollar vilka delar av trädet som sökområdet omfattar och lägger till de element till vektorn som man skickar in som argument.

Klassen innehåller även variablerna float x, float y, float width, float height. Pekarna nw, ne, sw, se som pekar på dess barn. Och vektorn <T> points som lagrar elementen som tillhör trädet.

Vi har även en static const int MaxElements som är satt till fyra för att kunna ändra hur många noder det krävs innan vi delar upp trädet.

Structen QuadTreeElement innehåller följande metoder:

- QuadTreeElement(), default konstruktor
- QuadTreeElement(float x, float y, T value), konstruktor som sätter vilken plats elementet ska ha och vilken data den ska innehålla.
- ~QuadTreeElement, destruktör.

QuadTreeElement innehåller även variablerna float x, float y, T data. Dessa variabler sparar position och data på elementet.

## Lösningens begränsningar

En begränsning som vi har är ifall vi ändrar storleken på trädet så kan det hända att noder hamnar utanför.

Om man har ändrat storleken tar vi inte hänsyn till de element som var i trädet från början så då kan vi ha element som från början låg innanför trädet men efter storleksändringen är utanför men som fortfarande tillhör trädet.

För att behålla strukturen i trädet kan vi innan storleksändringen spara alla noder i en lista och efter ändringen lägga tillbaka elementen på deras position. Då skulle de element som är utanför trädet försvinna. Detta skulle kräva en ny funktion typ `Resize()`.

Eftersom vi använder oss av floats för bredd och höjd så kan vi dela upp trädet väldigt många gånger. Plus att de tar upp lite mer minne.

Vi har inte något argument till konstruktorn till `QuadTree` för att sätta `MaxElements` för att ändra hur många noder som krävs innan trädet delas upp så måste man in i källkoden.

Vi har allt public så användaren kommer åt att ändra storleken och startpunkten. Detta kan innebära problem om någon användare ändrar dessa variabler så kan strukturen på trädet förstöras.



## Problem och reflektioner

Under laborationens gång har vi stött på ett par problem.

Vårt största problem som vi har fått är att vi fick unresolved externals p.ga vi hade deklarerat vår dekonstruktor i QuadTree men inte definierat den.

Ett av våra andra problem är att vi inte fått FindElement till att fungera som den ska. Den har antingen gett oss för få element eller för många. Detta hände på grund av vi hämtade alla värden från noderna som sökområdet omfattade. När vi insåg problemet så var det enkelt att fixa till. Vi satte bara en koll till.

I vår InsertElement så hade vi ett tag att den alltid delade upp trädet när vi stoppade in element.

## Diskussion

I den här labben så känner vi att vi har fått mer förståelse för hur QuadTree fungerar. Och genom lite efterforskningar så har vi fått konkreta exempel när vi kan använda oss av dem.

Och genom att arbeta i grupp så har det varit givande att få diskutera hur man kan implementera de olika funktionerna.

Vid felsökningen så har det även varit givande att ha haft två par ögon som har letat efter fel.

## Testkörningar

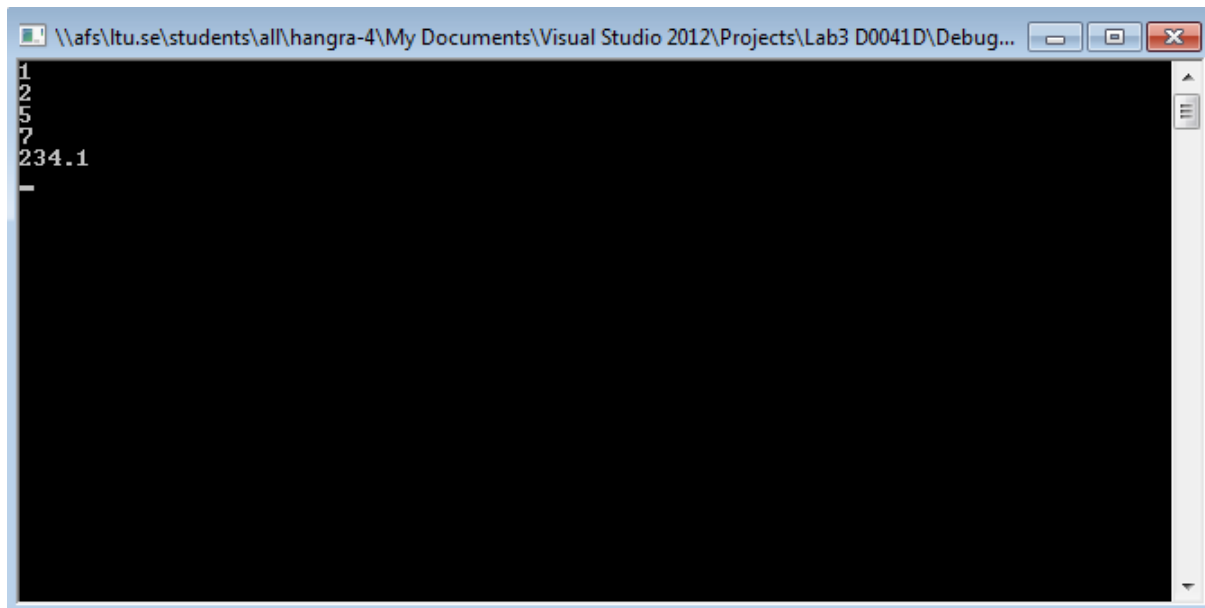
För att testa vår kod så skrev i följande rader i main() med att sätta ett par element både innanför och utanför trädets för att se ifall de hamnade rätt

```

1  #pragma once
2  #include "Quadtree.h"
3  #include <iostream>
4
5
6  void main()
7  {
8      std::vector<QuadTreeElement<float>> tjabba;
9      QuadTree<QuadTreeElement<float>> hej;
10     hej.Setup(15,15);
11     QuadTreeElement<float> x1(4,4,1);
12     QuadTreeElement<float> x2(5,2,2);
13     QuadTreeElement<float> x3(5,1,4);
14     QuadTreeElement<float> x4(8,3,5);
15     QuadTreeElement<float> x5(17,1,6);
16     QuadTreeElement<float> x6(0,20,7);
17     hej.InsertElement(x1);
18     hej.InsertElement(x2);
19     hej.InsertElement(x3);
20     hej.InsertElement(x4);
21     hej.InsertElement(x5);
22     hej.Setup(8,8);
23     hej.InsertElement(QuadTreeElement<float>(1,1,14.5));
24     hej.InsertElement(QuadTreeElement<float>(7,3,234.1));
25     hej.InsertElement(QuadTreeElement<float>(2,2,7));
26     hej.InsertElement(QuadTreeElement<float>(9,9,8));
27     hej.FindElement(2,2,15,15,tjabba);
28
29     for(int i = 0; i<tjabba.size();i++)
30     {
31         std::cout << tjabba[i].data << std::endl;
32     }
33
34

```

Denna bild visar de värden som finns inom sökområdet.



```
1
2
5
7
234.1
-
```