# Toteutusdokumentaatio

Ohjelman luokat on pyritty refaktoroimaan mahdollisimman lyhyiksi ja vähentämään toisteisuutta yhteisten luokkien ja metodien alla. Huomasin toteutuksen aikana, että A\* ja Dijkstra ovat hyvin samankaltaiset.

**Algoritmit**

Dijkstra

goal.distance = 0

heap.add goal

while heap not empty

Node a = heap.poll

for each kaari from a

neighbor = kaari.target

weight = kaari.weight

distance = a.shortest + weight

if distance > a.shortest

neighbor.shortest = distance

neighbor.previous = a

heap.add a

A\*

goal.distance = 0

heap.add goal

while goal not in closed list

node a = heap.poll

closed.add a

for each kaari from a

Node b = kaari.target

int cost = a.shortest + kaari.weight

if b not in closed

if b.shortest > cost

b.shortest = cost

b.previous = a

heap.add b

**Tietorakenteet**

Heap

Tapaukset, joissa tyhjään kekoon lisätään alkio, tyhjästä keosta yritetään poistaa alkio tai jos keosta poistetaan sen ainut alkio eivät ole sisällytetty seuraavaan pseudokoodiin sillä toiminnot ovat hyvin suoraviivaiset ja vakioaikaiset.

heap.insert(node)

heap.size +1

int k = heap.size -1

while k > 0 and node < parent

swap k, parent

k = (k-1)/2

heap.poll

node small = heap.first

heap.first = heap.last

heap.size-1

heapify(root)

return small

heapify(i)

left = 2i +1

right = 2i +2

if r < heap.size

smaller child = compare left, right

if heap.i > smaller

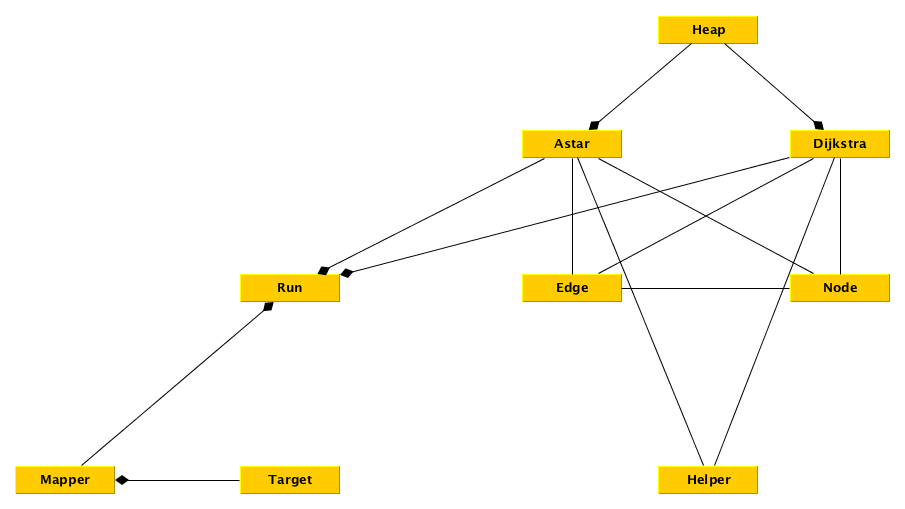
swap i, smaller

heapify(smaller)

else if left = heap.size and heap.i > left

swap left, i

**Luokkakaavio ja ohjelman yleisrakenne**

****

Kuva 1. Luokkakaavio (keskeneräinen, lopullinen versio palautukseen)

**Kehitettävää**

TBA.