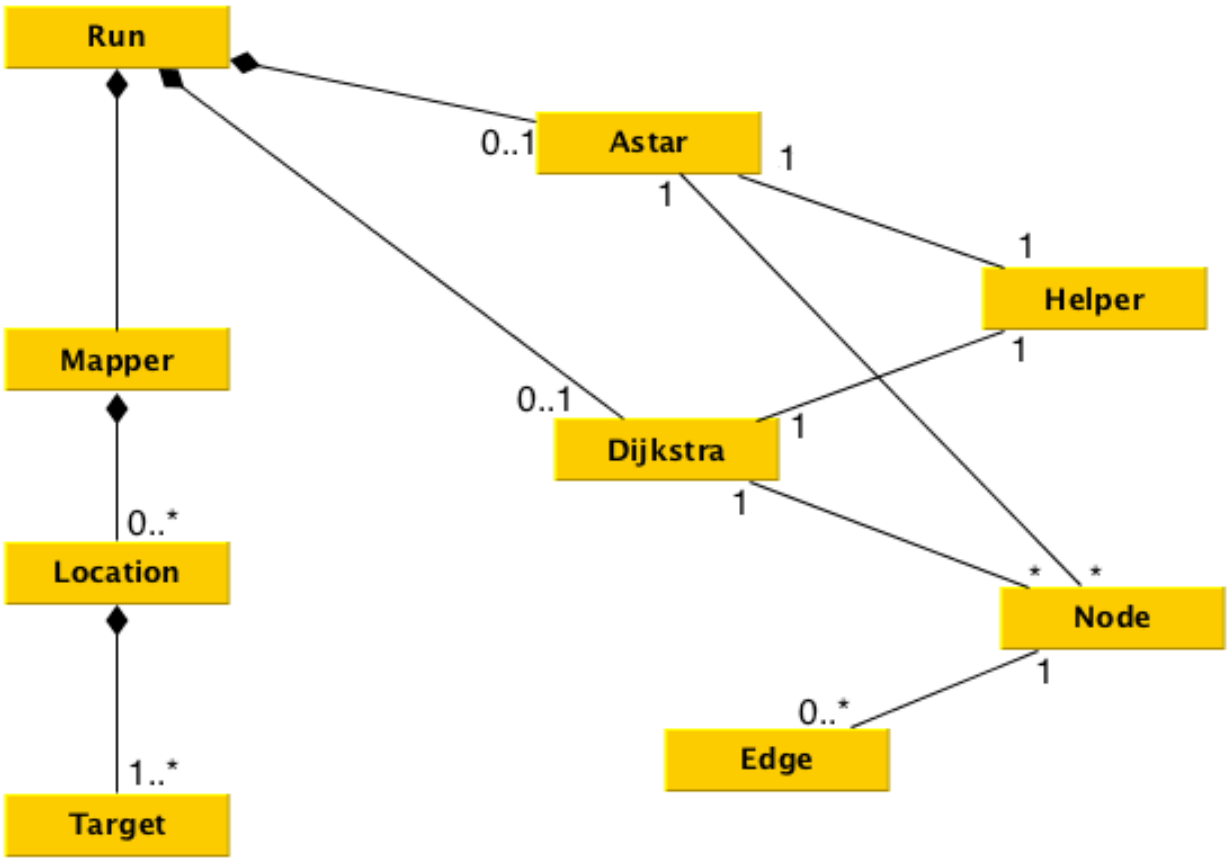
Toteutusdokumentaatio

Ohjelmaa toteuttaessani huomasin, että määrittelydokumenttiin kirjaamani suunnitelmat hieman muuttuivat. Jätin kuitenkin dokumentin alkuperäiseksi, jotta sen kanssa on helppo verratan esimerkiksi tietorakenteiden suunniteltua ja toteutunutta toteutusta. Huomasin toteutuksen aikana myös, että A* ja Dijkstra ovat hyvin samankaltaiset.

Luokkakaavio ja ohjelman yleisrakenne

Ohjelman rakenne on melko suoraviivainen. Pääohjelma ohjailee muuta toimintaa ja luo tarvittavia olioita. Tekstitiedoston lukeminen on toteutettu sillä oletuksella, että tiedosto on halutun mallinen. En ole keskittynyt sen sisällön testaamiseen. Sen lisäksi oletetaan, että kaaripainot eivät ole negatiivisia ja A* toiminnan kannalta yksikään kaari kahden pisteen välillä ei saa olla lyhyempi kuin heuristiikan antama arvio tälle etäisyydelle.



Kuva 1. Luokkakaavio

Luokkakaavioon ei ole laitettu tietorakenteita. Kaavio selvittää karkeasti miten luokat liittyvät toisiinsa.

Algoritmit

Dijkstra

```
initialize
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*

```
initialize
set heuristics
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
```

Molempien algoritmien initialize on hieman hitaampi metodi, sillä siinä luodaan algoritmien tarvitsemat Node ja Edge oliot. Aikavaativuus tälle on O(n^2). Sen lisäksi A*:rin set heuristic on aikavaativuudeltaan O(n) sillä se käy kertaalleen Nodet läpi ja laskee niiden etäisyysarvion maalista. Arvioin algorimit kuitenkin myös ilman initialize operaatiota:

Dijkstra: O((nodejen + edgejen lukumäärä)*log nodejen määrä)). Eli keko-operaatiot sekä jokaiselle solmulle sen jokaisen kaaren tarkistus ja mahdollinen relaksointi.

A*: Aikavaativuus kuten Dijkstrassa mutta käytännössä nopeampi, sillä ei tutkita kaikkia solmuja vaan lähdetään heuristiikan avulla oikeaan suuntaan verkossa.

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) aikavaativuus korkeintaan O(log n). Tilavaativuus O(1).

```
heap.size +1
int k = heap.size -1

while k > 0 and node < parent
  swap k, parent
  k = (k-1)/2
```

heap.poll aikavaativuus korkeintaan O(log n). Tilavaativuus O(log n) rekursion takia.

```
node small = heap.first
heap.first = heap.last
heap.size-1
heapify(root)
return small
```

heapify(i) aikavaativuus korkeintaan O(log n). Tilavaativuus O(log n) rekursion takia.

```
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
```

LinkedList

Listattu algoritmien vaatimat operaatiot. Listaan toteutettu myös muutama muu ArrayListin toiminnan kaltainen operaatio.

Add (olio o) aikavaativuus O(1). Tilavaativuus O(1).

```
if empty
  Head = o
  Tail = o
else
  o.next = head
  head.prev = o
  head = o
```

Search (olio s) aikavaativuus O(n), käydään kerran lista läpi. Tilavaativuus O(1)

```
if empty
  return null
for each olio
  if olio = s
    return s
return null
```

Contains (olio c) aikavaativuus O(n), käydään kerran lista läpi. Tilavaativuus O(1)

```
if empty
  return false
for each olio
  if olio = c
    return true
return false
```

Puutteet ja parannusehdotukset

-Koodissa paljo varaa refaktorointiin esimerkiksi Dijkstra ja Astar yhteisen rajapinnan / staattisen yliluokan alle.
-Tiedoston luvun testaus sisällön suhteen, sekä solmujen ja nodejen luonti ei ole suoraviivaisin. Nyt oletetaan, että tekstitiedosto on oikeanlainen eikä varauduta virheisiin.
-Myös LinkedListassa on varaa siistiä koodia lyhyemmäksi.
-Kaaripainoja / heuristiikkaa ei tarkisteta. Oletetaan karttatiedoston olevan sopiva.
-Tehdään koko Dijkstra, ei lopeta kun maali löytyy. A* lopettaa kun haluttu solmu on käsitelty.
-Erittäin paljon viime hetkien koodausta ja ratkaisuja viimeisellä viikolla -> toisteista koodia.
-Testit ovat melko tyhmiä mutta auttoivat havaitsemaan jos jokin muutos rikkoi jotakin erityisesti tietorakenteiden kanssa.
-Keon taulukon kokoa ei kasvateta, voi aiheuttaa ongelmia jos yritetään tallettaa liikaa alkioita kekoon.
-Testidata algoritmeille on melko hölmö ja yksinkertainen.
-Lopputulos ei siis mitään production level tavaraa mutta olen tyytyväinen että omat tietorakenteet tuntuvat toimivan ja algoritmien toiminta-ajatus selkeytyi kurssin aikana.

Lähteet:
Tiran luentomoniste.