Department of Computer Science
Umeå University 2 maj 2017

Objektorienterad Programmering (5DV133)
Design Obligatorisk Uppgift 3

Marko Nygård, oi12mnd
Ludvig Paju, c16lpu
Max Holmberg, dv16mhg
Gustav Wallgren, c16gwn

2 maj 2017

Kursansvarig: Anders Broberg
Handledare: Adam Dahlgren
Jakob Vesterlind
Sebastian Sandberg
Didrik Lindqvist
Daniel Harr

Innehåll

1 Systembeskrivning

1.1 Klasser och Klassansvar

Detta program är indelat i sex klasser, indelade med ansvar enligt UML-diagrammet i 1.2. För att initiera programmet finns en main-metod, vad denna gör och vilket ansvar den har är beskrivet i 1.3.

1.1.1 Grid

Klassen *Grid* representerar ett område som innehåller noder som kan kommunicera med varandra. *Grid* tar som inparameter en lista med noder. I ett visst tidssteg kan den med en viss sannolikhet generera en händelse per nod. Om en händelse inträffar i en viss nod kan denna nod med en viss sannolikhet skapa ett agentmeddelande. *Grid* har även koll på det maximala kommunikationsavståndet för en nod, samt bestämma nodernas grannar. Utöver händelser kan den även generera *Request* - objekt, dvs. en nods begäran om information om en händelse. Den håller koll på dessa i en lista, och tar bort *Request* - objekt som antingen har överskridit maximalt antal steg eller kommit tillbaka till sin startnod. Slutligen uppdaterar den samtliga *Agent*- och *Request* - objekts positioner i varje tidssteg.

1.1.2 Request

Klassen *Request* symboliserar en begäran om information som en nod sänder ut. Den ska ta sig till den nod som lagrar informationen om händelsen, dvs. Den nod där händelsen inträffade. Den kan göra ett visst antal hopp innan det anses att den inte kommer att nå sitt mål. Den håller även reda på vilken väg den har tagit i en Stack av noder. Om den når sitt mål sparar den ner informationen i en sträng. Dess metoder är:

- Move:
  Denna metod har som syfte att flytta en *Request* ett steg. Den prioriterar att hoppa till noder som har information om den begärda händelsen, finns ingen sådan hoppar den till en slumpvald granne.

- GetMessage:
  Denna metod returnerar strängen med meddelandet.

- hasReturned:
  Avgör om en begäran kommit tillbaka till startnoden i ett visst tidssteg.

1.1.3 Agent

Klassen *Agent* används som informationsspridning i omgivningen. En agent består av följande: En konstant för max antal hopp som kan göras mellan noder.
En hashmap med ett händelse-id som nyckel och en lista med riktning och steg till händelsen som värde.

En startnod av typen *Node*.
En integer som håller koll på hur många hopp agenten har gjort. En lista över besökta noder.
En nod av typen *Node* som håller koll på var agenten ska ta vägen i nästa tidssteg.

- move:
  Denna metod anropas för att få en agent att förflytta sig till nästa utvalda nod.

- update:
  Denna metod anropas efter varje hopp som agenten gör för att uppdatera den information som agenten bär på. Informationen som uppdateras är antalet hopp som har gjort och informationen om avstånd till händelser i hashmapen som agenten har med sig. Varje nod som agenten besöker läggs in i listan över besökta noder i denna metod.

1.1.4 Event

Event definierar en händelse i omgivningen. En händelse består av händelse-id, tid och position. Klassen har metoder för att skapa en händelse, hämta ett händelse-id och även att hämta tiden för när en händelse skapades.

1.1.5 Node

Node definierar en nod i omgivningen. En nod existerar på en position, och består av en routing-tabell som håller koll på events och riktning/antal steg till dem, en lista med dess grannar samt en lista med de händelser som inträffat på den positonen. Den har attribut för att hålla koll på en förfrågan, samt en kö för att hantera de agenter som passerar.

routingTable är alltså en exakt tabell som Agents routingTable. Riktningen till en annan nod definieras av den andra nodens plats i neighbourList.

Metoder:

- addEvent:
  När en händelse inträffar på dess position anropar Grid metoden addEvent som lägger till händelsen i eventsHere-listan samt routingTable.

- compareTable:
  När en agent passerar noden måste den kunna jämföra sin routingTable med nodens. Det gör den via compareTable, som jämför och uppdaterar tabellen med den mest effektiva vägen till samtliga event.

- createRequest:
  Skapar en förfrågan från noden och lägger till den i currentRequest

- checkRequest:
  Denna metod anropas varje tidssteg från Grid. Den kollar om en förfrågan är startad från noden, och om så är fallet anropar den dess hasReturned för att kolla om den

är tillbaka. Om den inte är det ökas antalet tidssteg sen förfrågan skickades med ett. Om den är tillbaka hämtas ett meddelande som beskrivet i Request ovan. Om en förfrågan inte återkommit efter 8*max antal steg för förfrågan så ska checkRequest skapa en ny förfrågan. Det ska den göra endast en gång.

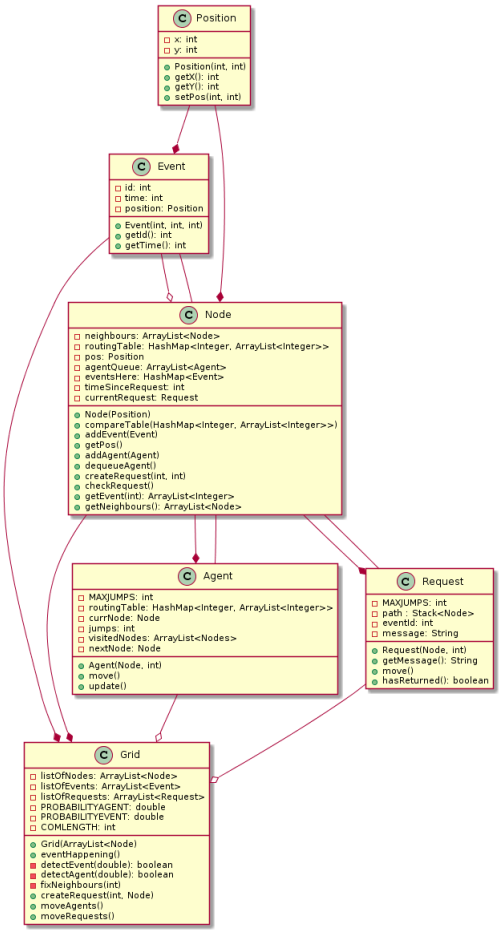
- getEvent :
  getEvent kollar om ett event finns i nodens routingtabell, och om så är fallet returneras dess egenskaper(steg och riktning). Denna anropas av request när den navigerar.

- getNeighbours:
  Anropas av Agenter och förfrågningar i deras navigation. Ger listan med nodens grannar.

1.1.6 Position

Position är en klass som håller koll på positionen för noder och händelser i omgivningen. Den lagrar inget annat än x och y-koordinat av typen integer. Den har metoder för att hämta dessa koordinater och en metod för att ändra positionens läge genom att uppdatera x och y.

1.2 UML-diagram



Figur 1 – UML-diagram över klasserna

1.3 Initiering och Uppdatering av Systemet

Omgivningen (*Grid*) initieras med hjälp av en lista av noder. För ett rektangulärt område kan noderna skapas t.ex. med en dubbel for-loop. *Grid* hittar även alla noders grannar (exempelvis alla noder som är inom en 15 längdenheters radie för en viss nod).

I varje tidssteg är det en viss sannolikhet i varje nod att en händelse skapas. Detta sker med *Grid*s metod eventHappening. Händelserna sparas i en lista. När en händelse inträffar finns det en viss sannolikhet att en *Agent* skapas i den nod där händelsen inträffade.

När en *Agent* skapas, får den information om händelsen (id och position). I varje följande tidssteg tar den ett steg i en viss riktning, med prioritering av obesökta noder, via sin metod move. För varje nod den besöker, kan den växla sin information med

den noden med sin metod update. Om noden inte har någon information om agentens händelse eller om informationen som agenten har om händelsen är bättre (avståndet till händelsen är kortare), uppdateras nodens routingtabell med agentens information. Agenten kan endast ta ett givet antal steg innan den försvinner.

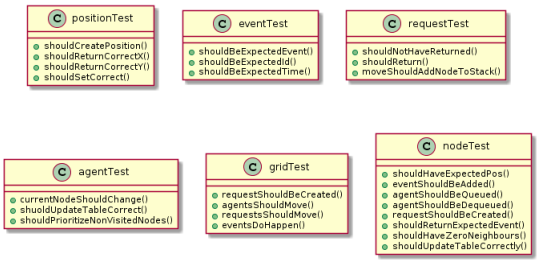
I ett visst tidssteg kan *Grid* skicka ut en begäran (dvs. ett *Request*-objekt ska skapas) om information om en händelse som finns i listan av sparade händelser i *Grid*. Begäran skapas i noden med dess metod createRequest, och den letar i varje tidssteg efter den granne som har den bästa informationen om den efterfrågade händelsen och hoppar till den grannen med metoden move. Om ingen granne har information om händelsen, går den till en slumpvald granne. Den sparar alla besökta noder i en stack. När den hittat noden med händelsen går den tillbaka längs samma väg den kom. Begäran kan endast ta ett visst antal steg, och om den inte hunnit tillbaka, skapas en ny begäran i startnoden. Detta kan ske endast en gång.

2 Tester

Programmet ska testas genom att de grundläggande funktionerna i varje klass testas. För *Position* kommer klassen att testas genom att kolla att en instans kan skapas, att rätt värden returneras vid anrop till get-metoderna, samt att setPosition ska fungera korrekt. Vidare ska *Event* testas genom att kolla att ett skapat event har rätt attribut. *Request* testas genom att kolla att ett skapat event inte är markerat som returned förrän det har kommit tillbaka, samt att det i ett område med få noder ska komma tillbaka. Det ska också kontrolleras att när ett request går ett steg ska den lägga till noden den kommer till i sin path. Klassen *Agent* ska testas genom att kontrollera att update och move-metoderna fungerar som det ska (alltså att agenten rör på sig och uppdaterar sin routingtabell korrekt), samt att agenten prioriterar att gå till noder den ej besökt förr.

Grid är en viktig klass då den håller reda på alla noder och hanterar alla agenter och förfrågningar. Eftersom den är väldigt beroende av övriga klasser behöver dess metoder endast testas översiktligt, då varje annan klass testas för sig själv. Det ska dock testas att createRequest skapar ett *Request*, att moveAgents förflyttar agenterna som förväntat, att moveRequests flyttar förfrågningarna samt att, om events är satta till att hända med 100-procentig sannolikhet så skapar eventHappening-metoden *Event*.

Klassen *Node* ska vid anrop till konstruktorn skapa en instans av en nod, så det måste kollas. Att denna instans har förväntad position ska sedan kontrolleras, samt att den har noll grannar inlagt som inititeringsläge. Det ska även kontrolleras att vid begäran ska *Event* läggas till enligt funktion, att hantering av agenter fungerar som det ska, samt att getEvent returnerar korrekt data. Det måste även kontrolleras att compareTable fungerar som den ska.



Figur 2 – UML-diagram över testerna.