Simulering av folkmassor

Datavetenskapliga institutionen, Umeå Universitet

24 mars 2005

Tobias Tjäder
dva99ttr@cs.umu.se
Rickard Winberg
ens04rwg@cs.umu.se

Sammanfattning

Detta dokument ingår som en beskrivning av ett subsystem i det stora Hultsfredsprojekt som har utvecklats i kursen TDBD21 vid Umeå Universitet VT05. Syftet är att ge en översikt av hur delsystemet som simulerar folkmassorna på Hultsfred är uppbyggt.

Dels ges en översikt av den övergripande arkitekturen av programvaran, med statiska strukturer och kommunikationsmönster, och dels ges små översikter över detaljerade delsystem såsom t.ex. vad som bestämmer hur personer rör sig.

Övergripande systembeskrivning

Systemet ur simuleringens synvinkel består av tre delar. Två delar ligger inom simuleringsprocessen, varav uppgiften för ena delen är att kontrollera agenternas vilja och räkna ut vart agenterna vill gå. Uppgiften för den andra delen är att ta agenterna från sin nuvarande position till den önskade positionen som första delen har räknat ut.

Förutom dessa två huvuddelar finns det en gömd logisk del som sköter all kommunikation med databasen, vilken är den tredje delen av systemet som simuleringen ser.

Hela systemet konfigureras mha. XML-filer som specificerar de detaljer systemet behöver veta för att kunna anpassa sin funktionalitet.

Viljesimulering och databashantering

I detta avsnitt ges en mer detaljerad beskrivning av modulen som styr agenternas vilja och kommunikation med databasen. Trots att beskrivningen är mer detaljerad än den övergripande systembeskrivningen har mycket information utelämnats. Det som redovisas här är en konceptuell modell, medan vissa saker ser annorlunda ut av optimeringsskäl när modellen har implementerats.

Statisk struktur

För att kunna avgöra var agenterna vill gå behöver denna modul innehålla en modell över Hultsfred. Denna modell innehåller framförallt:

1 Information om faciliteters positioner.

2 Spelschema för festivalen.

3 Information om artister, bl.a. vilken typ av musik de spelar.

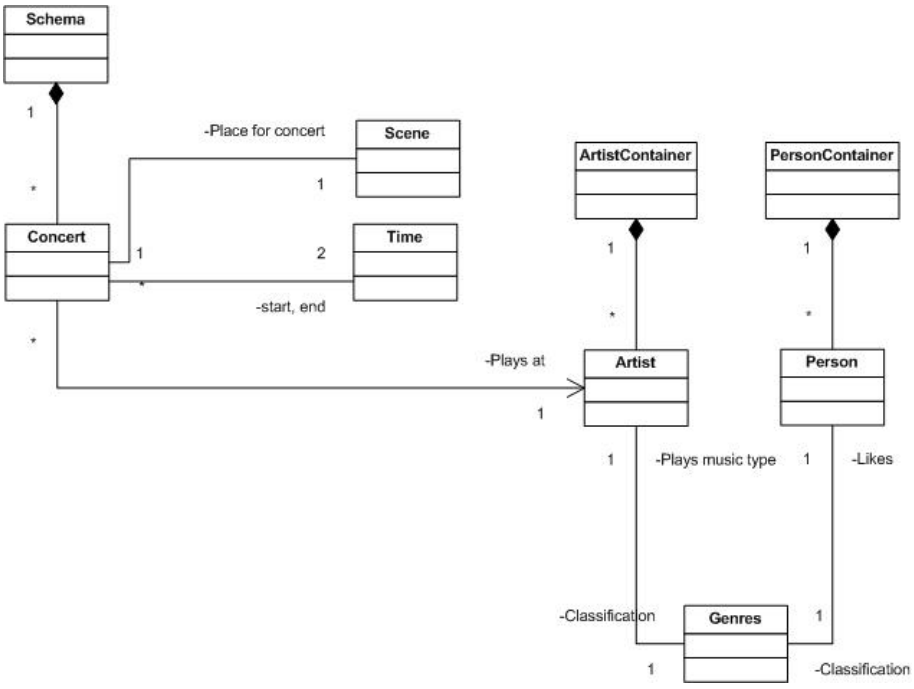
4 Information om personer på festivalen, bl.a. vilken typ av musik de tycker om Den statiska strukturen för denna modell visas i figur 1.

Dynamiskt uppförande

Modulen är uppdelad i tre delar som körs parallellt. Anledningen till detta är dels att databaskommunikationen inte ska störa övriga systemet när nätverkskommunikationen är långsam, och dels att det ska vara enkelt att kontrollera simuleringen med hjälp av en terminal.

Den första tråden hanterar kommandon som ges genom en terminal och kan användas för att t.ex. ändra hur fort tiden i simuleringen går, och för att hoppa i tiden.

Den andra tråden hämtar information över en TCP-förbindelse om vilka personer som inte ska simuleras, utan som kontrolleras av någon annan del i systemet

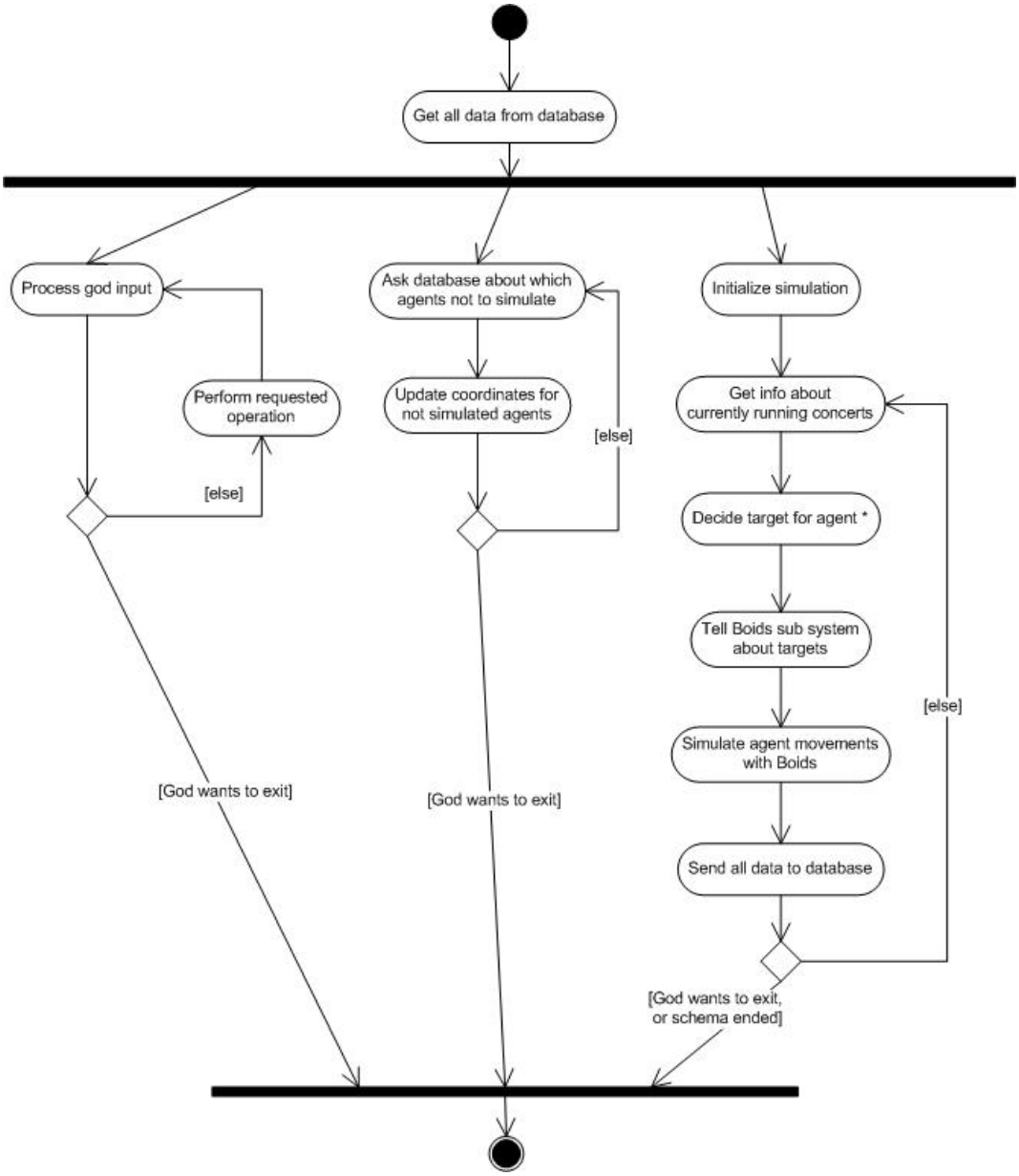


Figur 1: UML

utanför detta systems kontroll. Detta är viktigt eftersom de som simuleras måste ta hänsyn till alla personer, även de som inte kontrolleras internt.

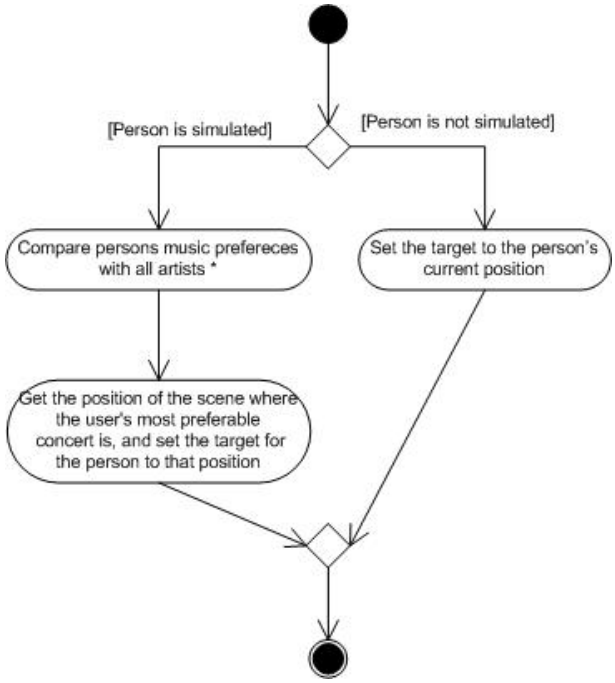
När information om vilka personer som inte ska simuleras har hämtas uppdateras deras positioner med den data som databasen UDP-strömmar till detta system. Denna process upprepas ända tills simuleringen avslutas. Denna tråd blockerar bara den tredje tråden under ett väldigt kort ögonblick när data förs över mellan trådarna, så om nätverkskommunikationen går dåligt stör inte denna tråd. Den tredje tråden fortsätter istället att iterera med gammal information.

Den tredje tråden har till uppgift att simulera tidsskeendet på festivalen och bestämma vart agenter vill gå. Den ansvar också för att anropa boidssystemet som utför själva förflyttningarna. Hur denna tråd fungerar mer detaljerat illustreras i figur 2.



Figur 2: Programflöde

Aktiviteten för att bestämma agentens mål har i nuvarande implementering en mycket enkel funktion. Målet styrs i nuläget enbart av konserter, men denna algoritm skulle kunna byggas ut till att ta hänsyn till en mängd olika faktorer, såsom toalettbehov, hungrighet, trötthet etc. Den nuvarande beslutsgången illustreras i figur 3 nedan.



Figur 3: Beslutflöde

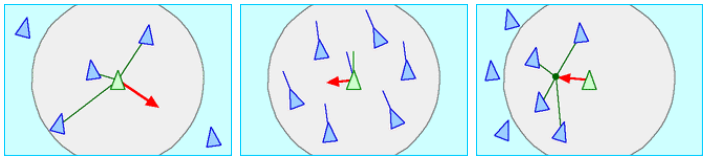
Agentförflyttning

När målet är bestämt för varje specifik agent skall själva simuleringen av förflyttningen börja. När en agent rör sig inom ett område måste denne interagera med andra agenter samtidigt som den rör sig. Detta har lösts genom en generell imlementation av BOIDS-modellen som konstruerats av Craig Reynolds [1]. Modellen baseras på tre st regler. Varje agent har dessa i åtanke inför varje cykel av förflyttningar:

1 *Separation*, undvik att tränga ihop sig med grannarna.

2 *Alignment*, efterlikna grannarnas hastighet och riktning.

3 *Coheison*, eftersträva att ha den genomsnittliga centrala positionen.



Figur 4: Sepearion, Alignment och Cohesion

Vektorerna som används vid varje beräkning har en viss styrka. Om till exempel en agent befinner sig långt från gruppens genomsnittliga position (regel 3) medför det att tillhörande vektor är stor. Varje agent har även en vektor som representerar agentens hastighet och riktning. Denna används för att bestämma agentens position vid nästa simuleringscykel. Sammanräknat har alltså systemet tre st vektorer att använda för att uppdatera en agents position. Detta görs enligt ekvationen:

$$V_{new} = \alpha(Separation) + \beta(Alignment) + \gamma(Coheison)$$

Där V_{new} är riktningsvektorn åt vilket håll agenten kommer röra sig nästa cykel. Storleken på denna vektor representerar också vilken hastighet som agenten kommer röra sig. Notera faktorerna α, β och γ. Dessa är vikter på hur mycket respektive vektor skall påverka det slutliga resultatet.

För att få ett fungerande BOIDS-system måste alla beräkningar göras på vektorer i föregående cykel innan uppdatering görs på nuvarande cykel. Det vill säga, systemet måste vara *serialiserat*.

När agenten skall uppdatera sin position görs en addition mellan dess gamla riktningsvektor och nya riktningsvektor:

$$V = \nu_1(V_{old}) + \nu_2(V_{new})$$

Detta för att förhindra att agenten skall kunna göra orealistiska vändningar. Faktorerna ν_1 och ν_2 är vikter som bestämmer hur mycke gamla respektive nya vektorn skall påverka den slutgiltiga riktningsvektorn. Till slut uppdateras agentens slutgiltiga position genom att addera riktningsvektorn till den nuvarande positionen.

$$Pos_{new} = Pos_{old} + V$$

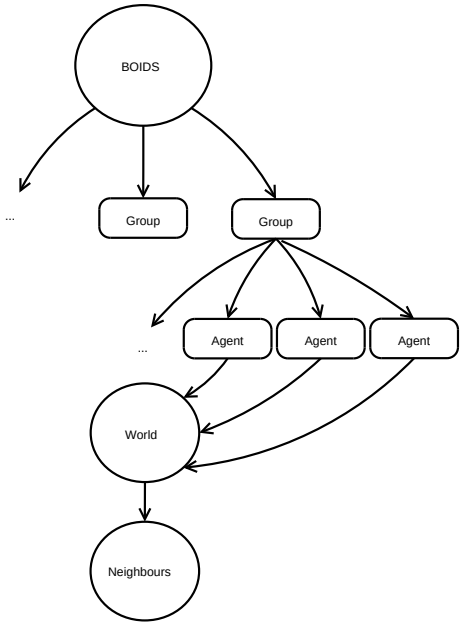
Det som hittils redogjorts är i princip vanliga BOIDS. En agents rörelse beror enbart på dess grannar. För att kunna bestämma vart en agent skal gå läggs helt enkelt ytterliggare en vektor till i beräkningen av V_{new}.

$V_{new} = V_{new} + \theta(Target)$

Detta medför att agenten också kommer att rörs sig mot ett specifikt mål.

Applicering av BOIDS-modellen

Systemet som grundar sina beräkningar enligt den angivna modellen, beskrivs i figur 5. Varje agent instansieras av en tillhörande gruppklass. Alla förflyttnings-anrop till agenter sker genom denna klass. När en agent skall göra en förflyttning anropar den world-klassen för att få veta om den har några grannar. Beroende på befintliga grannar och det egna målet anpassas beräkningar av nästa steg enligt BOIDS-modellen. När varje individuell agent har beräknat sin nästa position rapporterar denne detta till world-klassen.



Figur 5: BOIDS systemet

Resultat

Det beskrivna systemet kommunicerar alltså mot en given databas som den kontinuerligt uppdaterar med agenters positioner. Positionerna beräknas med hjälp av en implementation av Graig Reynolds BOIDS-modell som simulerar agenters rörselse från en plats till en annan. Vilken plats de rör sig mot simuleras efter varje agents preferenser som påverkar vart den "vill" gå vid just denna tidpunkt. Preferenserna kan vara musiksmak, hunger, trötthet etc. Dessa preferenser beräknas mot agentens behova samt också efter evenemang på Hultsfredfestivalen så som: koncerter, tävlingar m.m.

Referenser

[1] Reynolds, C. W. (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34.y