Danger Dudes

Processorienterad programmering (1DT049) vÃĕren 2012. Slutrapport fÃűr grupp 15

Tommy Engstrom: 840306-1610
Mattias Kjetselberg: 790713-1457
Carl Carenvall: 840119-4975

May 24, 2012

Contents

1 Inledning

En central del i vÃěrat val av projekt var lÃďrande. Vi ville snarare gÃűra nÃěgot dÃďr vi lÃďrde oss mycket Ãďn nÃěgot som skulle innebÃďra att vi kunde producera mycket. En fÃűrÃďndring som skedde tidigt var en skiftning av fokus fÃűr projktet. Ett zeldaliknande spel fanns fortfarande med som mÃěl, men dÃě snarare med syfte att visa upp tekniken som var det vi ville lÃďgga mest energi pÃě. VÃěr fÃűrhoppning var att kunna gÃűra ett kommunikations/processhanteringslager som skulle tillÃěta en uppdelning av spelvÃďrlden pÃě mer Ãďn en server som var och en ansvarar fÃűr en yta av vÃďrlden i frÃěga.

2 Danger Dudes

FrÃěn anvÃďndarens perspektiv Ãďr dangerdudes bara Ãďnnu ett multiplayerspel sett ovanifrÃěn med mÃűjlighet till olika typer av interaktion med vÃďrlden och andra spelare. NÃďr anvÃďndaren flyttar runt i spelvÃďrlden kommer hen inte mÃďrka om man flyttas frÃěn en server till en annan.

3 ProgrammeringssprÃěk

FÃűr sjÃďlva spelet sÃě valde vi att anvÃďnda oss av python. Avsikten med detta var att vi skulle behÃűva lÃďgga sÃě lite tid som mÃűjligt pÃě att utveckla sjÃďlva spelet och kunna fokusera pÃě tekniken under ytan. Som en del i valet att anvÃďnda python sÃě har vi valt att gÃűra den grafiska representationen med hjÃďlp av pygame. Detta var Ãěterigen en effektivitetsfrÃěga, dÃě det innebar att det krÃďvs vÃďldigt lite kod fÃűr att visa enkel grafik och skriva simpel spellogik. Dessutom gav det oss mÃűjlighet att lÃďsa oss hur det kan fungera att blanda tvÃě olika sprÃěk. Inte minst var vi nyfikna pÃě hur det skulle kunna fungera att blanda erlang med nÃěgot annat sprÃěk, fÃűr att se hur man skulle kunna kombinera deras respektive styrkor.

FÃűr kommunikationen mellan erlang och python sÃě har vi anvÃďnt oss av Erlport. Detta Ãďr ett kodbibliotek fÃűr python som tillÃěter att meddelanden skickas mellan en pythonprocess och erlang enligt erlangs "External Term Format". Kodbiblioteket var visserligen inte speciellt stort, och ganska okomplicerat att anvÃďnda. Men fÃűrutom nÃěgra exempel som visar hur erlport kan anvÃďndas sÃě finns det vÃďldigt lite dokumentation, nÃěgot som till och frÃěn fÃűrsvÃěrat arbetet.

4 Systemarkitektur

Arkitekturen beskrivs enklast utifrÃěn hur kommunikationen skÃűts, dÃě detta Ãďr det centrala i projektet. All information skickas via erlang pÃě ett eller annat sÃďtt, men den mesta av den skickas och tas emot av python.

4.1 Server

NÃďr en server startar sÃě har det dels en pythondel och dels en erlangdel.

4.1.1 Python

Pythondelen bestÃěr av tvÃě processer:

âĂć SjÃďlva spelet, som uppdateras kontinuerligt oavsett om meddelanden tas emot utifrÃěn eller inte. Det Ãďr hÃďr sjÃďlva spelvÃďrlden existerar och kÃűrs. Meddelanden kan skickas ut till nÃďtverket hÃďifrÃěn om och nÃďr sÃě behÃűvs.

âĂć En lyssnarprocess som vÃďntar pÃě kommunikation utifrÃěn. Denna process delar minne med spelprocessen och kan Ãďndra i deras gemensamma data om den mottagna informationen krÃďver det. I python innebÃďr det att all (eller vÃďldigt mycket) data blir lÃěst, och ur effektivitetssynpunkt finns det antagligen bÃďttre sÃďtt att gÃűra det pÃě. Denna process har ocksÃě mÃűjlighet att skicka data ut vid behov.

4.1.2 Erlang

Erlangdelen Ãďr nÃěgot mer komplex. Den bestÃěr av flera processer, men minst tvÃě:

âĂć FÃűrst och frÃďmst sÃě finns en process som lyssnar efter meddelanden frÃěn python, och som sedan ansvarar fÃűr att vidarebefordra dessa till rÃďtt klient nÃďr sÃědana har kopplat upp sig mot servern. Det fÃűrsta processer gÃűr Ãďr att skicka ett litet meddeleande till python fÃűr att python ska veta vart den ska skicka meddelanden.

âĂć Den andra processen som alltid Ãďr nÃďrvarande lyssnar efter fÃűrsÃűk att koppla upp sig mot servern. NÃďr en sÃědan fÃűrfrÃěgan kommer in (och accepteras) sÃě skickas ett meddelande till den fÃűrsta processen (som hanterar utgÃěende trafik) sÃě att denna ska kÃďnna till klienten. Slutligen sÃě skapar den en kopia av den sista sortens process som finns hos servern;

âĂć En process per klient ansvarar fÃűr att lyssna pÃě inkommande trafik och ansvarar sedan fÃűr att skicka vidare den till python.

4.2 Klient

Precis som serverdelen sÃě bestÃěr klienten dels av python och dels av erlang.

4.2.1 Python

Precis som hos servern sÃě har klienten en process fÃűr att lyssna efter inkommande trafik samt en process fÃűr logiken. Det som skiljer klientens pythondel frÃěn serverns Ãďr spellogiken. Hos klienten finns kod fÃűr den faktiska spelarinteraktionen, samt funktionalitet fÃűr att visa spelarens vy i ett grafiskt fÃűnster. Spelaren behÃűver inte ha kÃďnnedom om hur hela kartan ser ut, utan bara de delar hen kan se och interagera med.

4.2.2 Erlang

Hos klienten Ãďr erlangdelen betydligt mycket enklare. Den bestÃěr endast av tvÃě processer.

âĂć FÃűrst och frÃďmst behÃűvs, precis som hos servern, en process som lyssnar efter meddelanden frÃěn python och som skickar den vidare till servern. Eftersom en klient bara Ãďr kopplad till en server vid varje givet tillfÃďlle behÃűver man inte hÃělla koll pÃě vart trafiken skall, utan den kan vidarebefordras rakt av.

âĂć En process som lyssnar efter inkommande trafik frÃěn nÃďtverket och sedan helt enkelt skickar den vidare till python. DÃě denna data inte behÃűver tolkas av erlang kan den helt enkelt skickas vidare in till python direkt. Denna process fungerar precis som de processer servern har fÃűr varje klient.

5 Samtidighet

NÃďr en spelare vill flytta sig i spelvÃďrlden skickas ett meddelande om detta till servern, dÃďr sjÃďlva fÃűrflyttningen ska ske. Iom att spelet sker i realtid och flera spelare samtidigt ska kunna fÃűrflytta sig (och potentiellt fÃűrsÃűka flytta sig till samma plats pÃě kartan) kan varje spelare pÃěverka servern i princip samtidigt, och i vissa fall eventuellt vilja pÃěverka samma data samtidigt. Krockar bÃűr, vad vi vet, inte kunna ske som det ser ut. Det lÃďge dÃďr det skulle kunna bli en krock Ãďr mellan erlang och python, men utifrÃěn den information vi fÃětt kommer meddelandena in till python behandlas ett och ett allteftersom de anlÃďnder.

En annan punkt vid vilken samtidighet Ãďr aktuellt Ãďr nÃďr en klient ansluter till servern. Klientens socket kommer dÃě sparas i en lista i den process i erlang som ansvarar fÃűr att skicka data frÃěn servern till klienterna. Detta kan i princip ske samtidigt som servern fÃűrsÃűker skicka data till klienterna.

6 Algoritmer och datastrukturer

Koden kan delas in i tvÃě tydligt separata delar: erlang och python.

6.1 Erlang

I erlang finns det tvÃě aktiva moduler: client och server.

6.1.1 client

Klientsidan i erlang Ãďr ganska simpel: en process som vÃďntar pÃě att python skall skicka nÃěgot och vidarebefordrar det, samt en process som vÃďntar pÃě att servern skall skicka nÃěgot och vidarebefordar det. NÃďr klientens processer vÃďl Ãďr igÃěng behÃűver ingen data i erlang uppdateras (om man inte ska byta server). NÃďr datan kommer frÃěn python Ãďr den packeterad enligt erlangs "External Term Format", och binÃďr patternmatching anvÃďnds fÃűr att plocka ut den data som faktiskt ska skickas. Detta Ãďr mÃěhÃďnda onÃűdigt, men vi har valt att gÃűra sÃě under utvecklingen fÃűr att fÃě bÃďttre koll pÃě vad som faktiskt skickas.

6.1.2 server

Serversidan beskrivs i ganska ingÃěende detalj i sektionen om systemarkitektur. Den skiljer sig frÃěn klienten pÃě tre punkter:

1. En process lyssnar hela tiden efter inkommande tcp trafik pÃě en fÃűrdefinierad port (vi anvÃďnder port 2233). NÃďr en uppkoppling upptrÃďttas sÃě skapas en ny process kopplad till den socket som hÃűr till uppkopplingen, som lyssnar efter inkommande trafik. Inge i dessa processr uppdateras, utan de flyttar bara data frÃěn nÃďtverket till servern.

2. Hos servern finns flera processer som lyssnar i stÃďllet fÃűr bara en. Detta innebÃďr att data kan komma in till servern frÃěn flera kÃďllor samtidigt.

3. NÃďr en ny uppkoppling skapas sÃě mÃěste den process i erlang som skickar data frÃěn servern till klienterna uppdateras. DÃě data vÃďldigt sÃďllan skickas till mer Ãďn en klient i taget mÃěste denna process kunna vÃďlja vilken socket som skall fÃě trafiken.
   Detta har vi lÃűst genom en lista med sockets och ett tillhÃűrande id (endast ett nummer som bÃűrjar pÃě 0 och rÃďknas upp fÃűr varje ny klient). NÃďr en ny klient kopplar upp sig mot servern skickas ett meddelande till denna process om hÃďndelsen, tillsammans med klientens socket. Denna socket lÃďggs dÃě till i listan genom att (som brukligt Ãďr i erlang) funktionen kÃűrs igen, dÃďr listan Ãűver klienter har uppdaterats med den nya informationen.

6.2 Python

ÃĎven i python finns endast de tvÃě modulerna klient och server.

7 FÃűrslag pÃě fÃűrbÃďttringar

8 Reflektion

9 Installation och fortsatt utveckling

FÃűr att kunna kÃűra programmet behÃűvs fÃűljande separata delar:

- âĂć Erlang, av sjÃďlvfÃűrklarande skÃďl.
- âĂć Python, av lika sjÃďlvfÃűrklarande skÃďl.
- âĂć Erlport, fÃűr kommunikationen mellan erlang och python.
- âĂć Pygame, fÃűr grafiken och viss logik.

Servern och klienterna startas separat frÃěn varandra, men servern MÃĚSTE i nulÃďget startas fÃűrst. DÃďrtill behÃűver klienterna startas med det ip som servern har. BÃěde servern och klienterna startas genom erlang, som sedan automatiskt kÃűr sina respektive pythonscript.