

Danger Dudes

Processororienterad programmering (1DT049) våren 2012. Slutrapport för
grupp 15

Tommy Engstrom: 840306-1610

Mattias Kjetselberg: 790713-1457

Carl Carenvall: 840119-4975

25 maj 2012

Innehåll

1	Inledning	2
2	Danger Dudes	3
3	Programmeringsspråk	3
4	Systemarkitektur	3
4.1	Server	3
4.1.1	Python	3
4.1.2	Erlang	4
4.2	Klient	4
4.2.1	Python	4
4.2.2	Erlang	4
5	Samtidighet	5
6	Algoritmer och datastrukturer	5
6.1	Erlang	5
6.1.1	client	5
6.1.2	server	5
6.2	Python	6
6.2.1	Klient	6
7	Förslag på förbättringar	7
8	Reflektion	7
9	Installation och fortsatt utveckling	7

1 Inledning

En central del i vårt val av projekt var lärande. Vi ville snarare göra något där vi lärde oss mycket än något som skulle innebära att vi kunde producera mycket. En förändring som skedde tidigt var en skiftning av fokus för projektet. Ett zelda-liknande spel fanns fortfarande med som mål, men då snarare med syfte att visa upp tekniken som var det vi ville lägga mest energi på. Vår förhoppning var att kunna göra ett kommunikations/processhanteringslager som skulle tillåta en uppdelning av spelvärlden på mer än en server som var och en ansvarar för en yta av världen i fråga.

2 Danger Dudes

Från användarens perspektiv är dangerdudes bara ännu ett multiplayer spel sett ovanifrån med möjlighet till olika typer av interaktion med världen och andra spelare. När användaren flyttar runt i spelvärlden kommer hen inte märka om man flyttas från en server till en annan.

3 Programmeringsspråk

För själva spelet så valde vi att använda oss av python. Avsikten med detta var att vi skulle behöva lägga så lite tid som möjligt på att utveckla själva spelet och kunna fokusera på tekniken under ytan. Som en del i valet att använda python så har vi valt att göra den grafiska representationen med hjälp av pygame. Detta var återigen en effektivitetsfråga, då det innebar att det krävs väldigt lite kod för att visa enkel grafik och skriva simpel spellogik. Dessutom gav det oss möjlighet att läsa oss hur det kan fungera att blanda två olika språk. Inte minst var vi nyfikna på hur det skulle kunna fungera att blanda erlang med något annat språk, för att se hur man skulle kunna kombinera deras respektive styrkor.

För kommunikationen mellan erlang och python så har vi använt oss av Erlport. Detta är ett kodbibliotek för python som tillåter att meddelanden skickas mellan en pythonprocess och erlang enligt erlangs "External Term Format". Kodbiblioteket var visserligen inte speciellt stort, och ganska okomplicerat att använda. Men förutom några exempel som visar hur erlport kan användas så finns det väldigt lite dokumentation, något som till och från försvårat arbetet.

4 Systemarkitektur

Arkitekturen beskrivs enklast utifrån hur kommunikationen sköts, då detta är det centrala i projektet. All information skickas via erlang på ett eller annat sätt, men den mesta av den skickas och tas emot av python.

4.1 Server

När en server startar så har det dels en pythondel och dels en erlangdel.

4.1.1 Python

Pythondelen består av två processer:

- Själva spelet, som uppdateras kontinuerligt oavsett om meddelanden tas emot utifrån eller inte. Det är här själva spelvärlden existerar och körs. Meddelanden kan skickas ut till nätverket häifrån om och när så behövs.

- En lyssnarprocess som väntar på kommunikation utifrån. Denna process delar minne med spelprocessen och kan ändra i deras gemensamma data om den mottagna informationen kräver det. I python innebär det att all (eller väldigt mycket) data blir låst, och ur effektivitetssynpunkt finns det antagligen bättre sätt att göra det på. Denna process har också möjlighet att skicka data ut vid behov.

4.1.2 Erlang

Erlangdelen är något mer komplex. Den består av flera processer, men minst två:

- Först och främst så finns en process som lyssnar efter meddelanden från python, och som sedan ansvarar för att vidarebefordra dessa till rätt klient när sådana har kopplat upp sig mot servern. Det första processer gör är att skicka ett litet meddelande till python för att python ska veta vart den ska skicka meddelanden.
- Den andra processen som alltid är närvarande lyssnar efter försök att koppla upp sig mot servern. När en sådan förfrågan kommer in (och accepteras) så skickas ett meddelande till den första processen (som hanterar utgående trafik) så att denna ska känna till klienten. Slutligen så skapar den en kopia av den sista sortens process som finns hos servern;
- En process per klient ansvarar för att lyssna på inkommande trafik och ansvarar sedan för att skicka vidare den till python.

4.2 Klient

Precis som serverdelen så består klienten dels av python och dels av erlang.

4.2.1 Python

Precis som hos servern så har klienten en process för att lyssna efter inkommande trafik samt en process för logiken. Det som skiljer klientens pythondel från serverns är spellogiken. Hos klienten finns kod för den faktiska spelarinteraktionen, samt funktionalitet för att visa spelarens vy i ett grafiskt fönster. Spelaren behöver inte ha kännedom om hur hela kartan ser ut, utan bara de delar hen kan se och interagera med.

4.2.2 Erlang

Hos klienten är erlangdelen betydligt mycket enklare. Den består endast av två processer.

- Först och främst behövs, precis som hos servern, en process som lyssnar efter meddelanden från python och som skickar den vidare till servern. Eftersom en klient bara är kopplad till en server vid varje givet tillfälle behöver man inte hålla koll på vart trafiken skall, utan den kan vidarebefordras rakt av.

- En process som lyssnar efter inkommande trafik från nätverket och sedan helt enkelt skickar den vidare till python. Då denna data inte behöver tolkas av erlang kan den helt enkelt skickas vidare in till python direkt. Denna process fungerar precis som de processer servern har för varje klient.

5 Samtidighet

När en spelare vill flytta sig i spelvärlden skickas ett meddelande om detta till servern, där själva förflyttningen ska ske. Iom att spelet sker i realtid och flera spelare samtidigt ska kunna förflytta sig (och potentiellt försöka flytta sig till samma plats på kartan) kan varje spelare påverka servern i princip samtidigt, och i vissa fall eventuellt vilja påverka samma data samtidigt. Krockar bör, vad vi vet, inte kunna ske som det ser ut. Det läge där det skulle kunna bli en krock är mellan erlang och python, men utifrån den information vi fått kommer meddelandena in till python behandlas ett och ett allteftersom de anländer.

En annan punkt vid vilken samtidighet är aktuellt är när en klient ansluter till servern. Klientens socket kommer då sparas i en lista i den process i erlang som ansvarar för att skicka data från servern till klienterna. Detta kan i princip ske samtidigt som servern försöker skicka data till klienterna.

6 Algoritmer och datastrukturer

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

6.1 Erlang

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

6.1.1 client

Klientsidan i erlang är ganska simpel: en process som väntar på att python skall skicka något och vidarebefordrar det, samt en process som väntar på att servern skall skicka något och vidarebefordrar det. När klientens processer väl är igång behöver ingen data i erlang uppdateras (om man inte ska byta server). När datan kommer från python är den packeterad enligt erlangs "External Term Format", och binär patternmatching används för att plocka ut den data som faktiskt ska skickas. Detta är måhända onödigt, men vi har valt att göra så under utvecklingen för att få bättre koll på vad som faktiskt skickas.

6.1.2 server

Serversidan beskrivs i ganska ingående detalj i sektionen om systemarkitektur. Den skiljer sig från klienten på tre punkter:

1. En process lyssnar hela tiden efter inkommande tcp trafik på en fördefinierad port (vi använder port 2233). När en uppkoppling uppträttas så skapas en ny process kopplad till den socket som hör till uppkopplingen, som lyssnar efter inkommande trafik. Inge i dessa processer uppdateras, utan de flyttar bara data från nätverket till servern.
2. Hos servern finns flera processer som lyssnar i stället för bara en. Detta innebär att data kan komma in till servern från flera källor samtidigt.
3. När en ny uppkoppling skapas så måste den process i erlang som skickar data från servern till klienterna uppdateras. Då data väldigt sällan skickas till mer än en klient i taget måste denna process kunna välja vilken socket som skall få trafiken. Detta har vi löst genom en lista med sockets och ett tillhörande id (endast ett nummer som börjar på 0 och räknas upp för varje ny klient). När en ny klient kopplar upp sig mot servern skickas ett meddelande till denna process om händelsen, tillsammans med klientens socket. Denna socket läggs då till i listan genom att (som brukligt är i erlang) funktionen körs igen, där listan över klienter har uppdaterats med den nya informationen.

6.2 Python

Även i python finns endast de två modulerna klient och server.

6.2.1 Klient

Python-klienten kör 2 trådar:

1. En tråd som lyssar efter meddelanden från Erlang och de meddelandena tas emot i 2 olika funktioner.
 - a) En init-funktion som Erlang-klienten anropar i början för att tala om vilken port Python ska använda för att skicka meddelanden genom Erlang till servern.
 - b) En funktion som tar emot meddelanden från servern om hur världen ser ut där klienten för tillfället befinner sig.
2. Den andra tråden kör Pygame för klientens GUI. Tråden ritas upp det som den andra tråden fått genom meddelanden från servern och tar också emot input från användaren. Input (till exempel förflyttning) från användaren skickas till servern genom den port den andra tråden fått meddelande om att använda och ritas inte upp för användaren förrän servern svarat tillbaka.

7 Förslag på förbättringar

8 Reflektion

9 Installation och fortsatt utveckling

För att kunna köra programmet behövs följande separata delar:

- Erlang, av självförklarande skäl.
- Python, av lika självförklarande skäl.
- Erlport, för kommunikationen mellan erlang och python.
- Pygame, för grafiken och viss logik.

Servern och klienterna startas separat från varandra, men servern **MÅSTE** i nuläget startas först. Därtill behöver klienterna startas med det ip som servern har. Både servern och klienterna startas genom erlang, som sedan automatiskt kör sina respektive pythonscript.