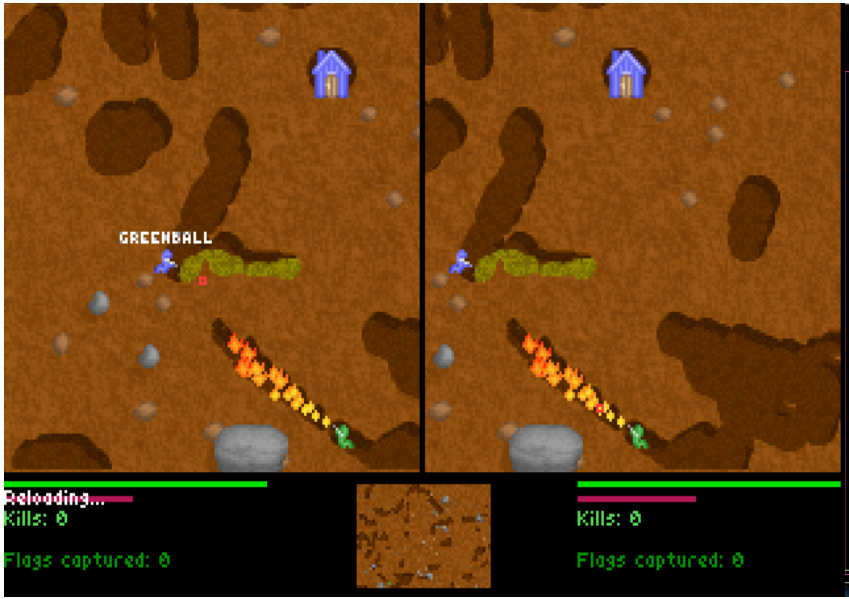
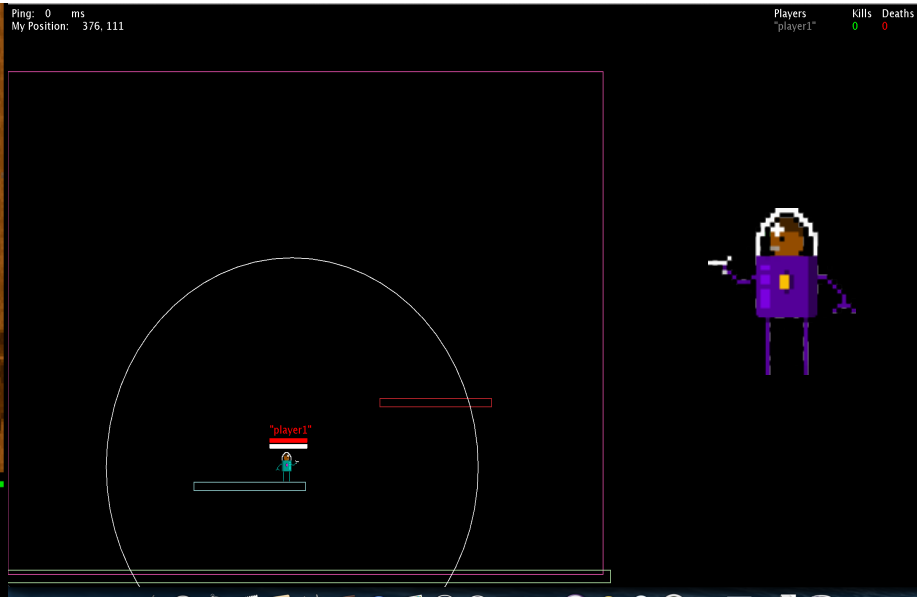

GEESE

GE.neric E.rlang SE.rver

- Namnet - GEESE
- Mål: multiplayerspel i realtid (inspiration: Liero)
- Spelserver
  - Hög responsivitet
  - Robusthet
  - Spellogik
- Valde att inledningsvis lägga fokus på servern
- GUI i JAVA

Liero vs Pixel Wars





Preview (demo)

Introduktion

- Servern är skriven i Erlang
- Fördelar med Erlang:
  - Skalbart
  - Passar väl där kravet på concurrency är stort
  - Felsäkert
  - OTP (Ramverk för server/klient-applikationer och liknande)

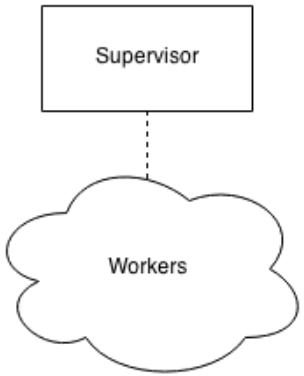
- Klienten är skriven i Java
- Fördelar med Java:
  - Plattformsoberoende
  - Stort bibliotek
  - Mycket information på nätet
  - Jinterface

Introduktion

- All kommunikation sker genom TCP (transmission control protocol)
- Garanterar att rätt paket kommer fram i rätt ordning
- Valde TCP över UDP

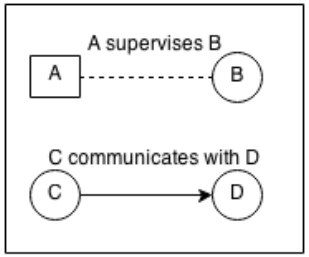
Supervisor

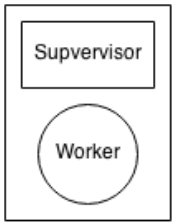
- Erlang OTP designprinciper
- Våran supervisor startar tre workers

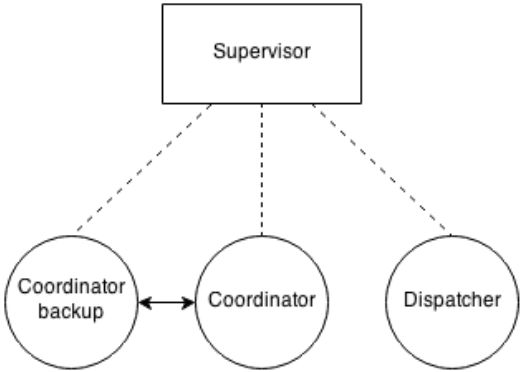


Server

- Dispatcher tar hand om alla anslutningar
- Coordinator har ett tillstånd som innehåller information om alla spelinstanser samt spelare
- Varje gång coordinatorns tillstånd ändras sparas detta i backup
- Crashar coordinatorn initieras den med tillstånd som fanns lagrat i backup (om det fanns något sparat)

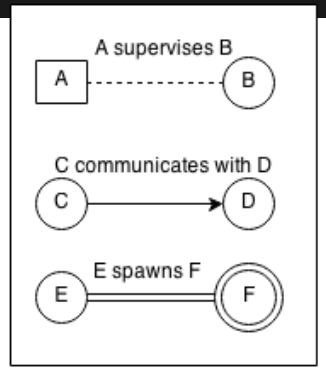


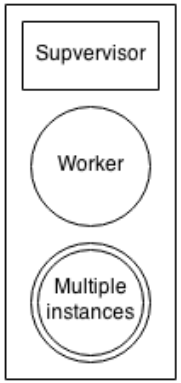


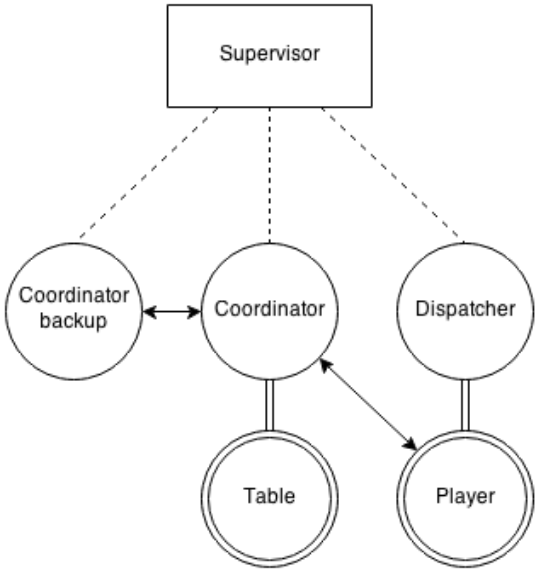


Server

- När en anslutning accepteras skapas en *Player*-process.
- Player-processen komminucerar med klienten.
- Om klienten vill skapa ett nytt spelbord, *Table* görs detta via coordinator
- Varje spelbord innehåller en instans av ett spel samt allmän info om spelet







Server

Klienten kommunicerar med servern via sin *Player*-process.

Denna process kommer vara i olika sates.

Talk state:

- Hantera kommandon och förfrågningar om olika spel. Lista/Skapa/Gå med

Game state:

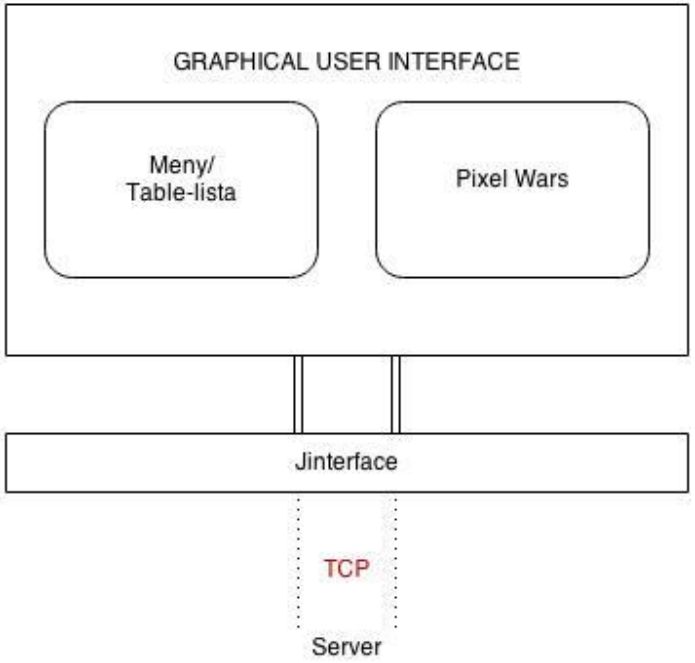
- Skicka kommandon för spelet. Move/Fire/Jump
- Ta emot spelinformation. *Game state*

Spel-logiken:

- All logik på servern (För- och nack-delar)
- Tar in kommandon och itererar spelet
- Skickar relevant spelinformation vidare

Java klient

- GUI
- Ingen spel-logik
- All data kommer från servern
- Kommunikation sker via TCP
- Jinterface-modulen fungerar som en tolk mellan klient och server



Kommunikation mellan klient och server: Jinterface och TCP

- All data som skickas mellan klient och server sker över TCP
- Datan består av binära representationer av Erlang-typer
- För att tolka och hantera datan på klientsidan används ramverket Jinterface som kan omvandla Erlangtyper till java-objekt och vice verca

Organisation och planering

- Startade en trello-board men det var inget vi använde oss av
- Träffats varje dag.
- Liten uppdelning i början, senare en strikt fördelning av arbetet efter olika programdelar som gen_server/java-client/spellogik

Diskussion av concurrency (serversidan)

- På serversidan har vi använt oss av Erlang och därmed actor-modellen
- Tack vare actor-modellen har vi undvikt behovet av synkronisering

Diskussion av concurrency (klientsidan)

- Fokusen i projeket låg på att skapa en generell spelserver
- Ej behov av concurrency på klientsidan
- Ville inte “tvinga” fram concurrency om det inte var nödvändigt

Deadlocks

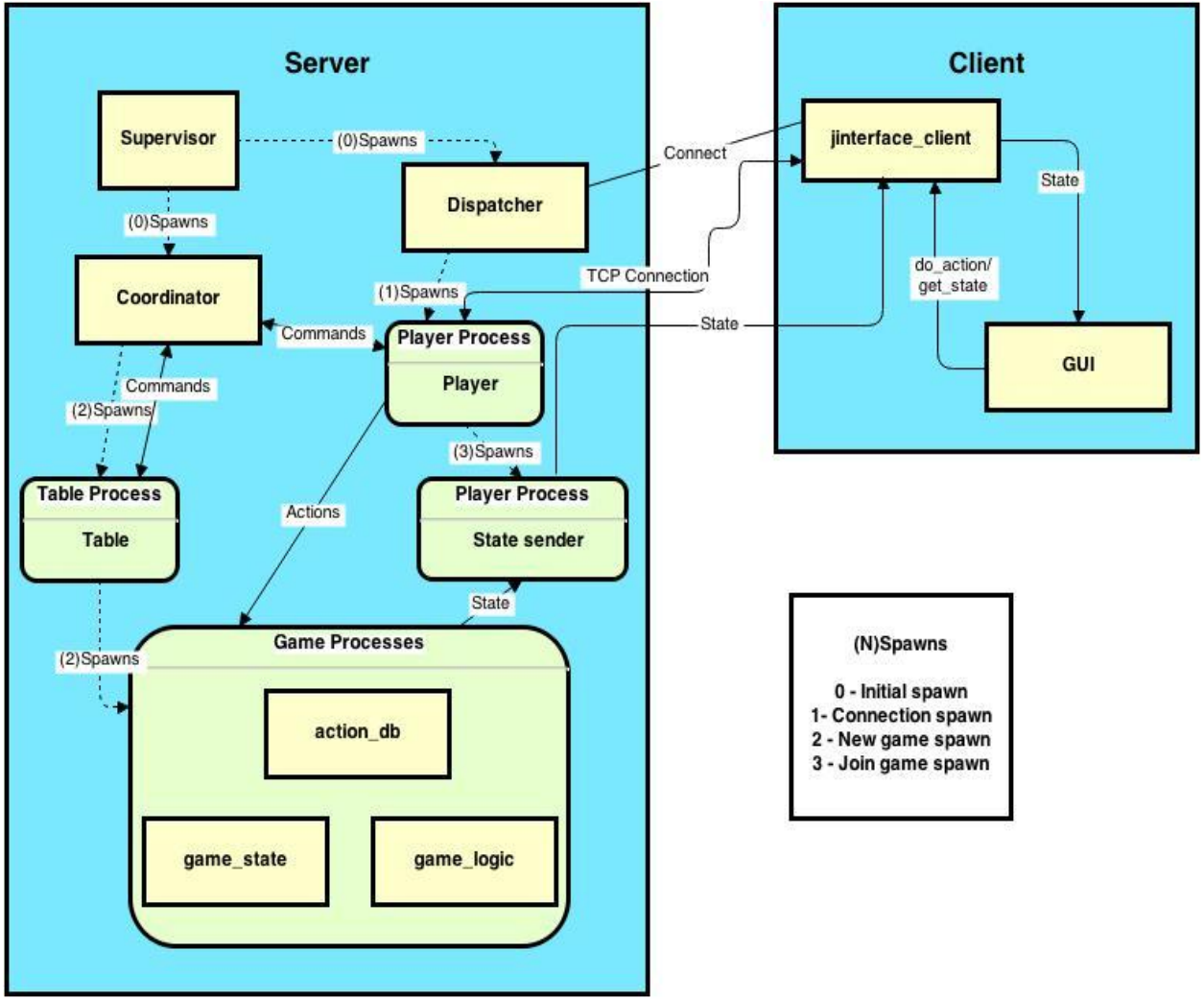
- Deadlocks kan inte uppstå
- Vi använder oss av actor-modellen
- De synkrona meddelanden vi skickar riskerar aldrig att orsaka deadlocks
- Operationer för mnesia är antigen atomiska eller lockless

Teknisk demonstration

Fire

```
private void fire(Point mousePosition) {
    OtpErlangList argList;
    OtpErlangInt type = new OtpErlangInt(25);
    OtpErlangInt x = new OtpErlangInt(mousePosition.x);
    OtpErlangInt y = new OtpErlangInt(Game.height - mousePosition.y);
    OtpErlangObject[] posArray = {x,  y};
    OtpErlangTuple posTuple = new OtpErlangTuple(posArray);
    OtpErlangObject[] argArray = {type, posTuple};
    argList = new OtpErlangList(argArray);
    Main.client.doAction("fire", argList);

}
```


Server
Supervisor
(0)Spawns
Dispatcher
(0)Spawns
Coordinator
(1)Spawns
Commands
Player Process
Player
Commands
(2)Spawns
(3)Spawns
Table Process
Table
Actions
Player Process
State sender
State
(2)Spawns
Game Processes
action_db
game_state
game_logic
Client
jinterface_client
Connect
State
TCP Connection
do_action/
get_state
State
GUI
(N)Spawns
0 - Initial spawn
1- Connection spawn
2 - New game spawn
3 - Join game spawn

Guldkorn

- Supervising av coordinator med en backup-modul

Ruttna ägg

- Alla *actions* lagras i en databas som ej är avsedd för mycket temporär data
- Lag spikes uppkommer ibland när flera bord är uppe. Detta problem kan bero på Erlangs inbyggda garbage collection
- Spellogiken, man kan gå genom hörn

Frågor och kommentarer

-