Prosjektoppgave HiST

LN350D Applikasjonsutvikling for mobile enheter

Vår 2011

Innlevering for
Ronny H Andersen
Ronny Øvereng
Karl-Erik Moberg

Innledning

Dokumentet er en avsluttende rapport for kurset ”LN350D Applikasjonsutvikling for mobile enheter” ved HiST.

Dokumentet beskriver oppgaven, løsningen og en egenvurdering av resultatet samt forslag til forbedringer.

Generelt

Oppgaven er en eksamens-innlevering til HiST, og lyder som følger:

> *Prosjektoppgave: Lag et program for å spille bondesjakk.*
>
> *Dette spillet kan tenkes i flere versjoner:*
>
> *1. Uten nettverk*
> *To spillere bruker den samme mobile enheten*
>
> *2. To over nettverk (med socket - behandles i leksjon 9)*
> *To spillere bruker hver sin mobile enhet*
>
> *3. Flere spillere*
> *Flere spillere bruker hver sin mobile enhet og spiller alle-mot-alle*
>
> *Prosjektene vurderes etter følgende kriterier:*
> *En greit fungerende app som utmerker seg verken positivt eller negativt gir karakteren C hvis den*
> **for enkeltstudenter inneholder versjon 1*
> **for toergrupper inneholder versjon 1 og 2*
> **for grupper på tre inneholder alle tre versjonene.*
> *Karakteren kan bli bedre hvis appen*
> **inneholder flere versjoner enn forventet*
> **utmerker seg med hensyn på funskjonalitet og/eller utseende*
> **har god/ryddig kode*
> *Treer-grupper som aspirerer til A kan bytte ut bondesjakk med ludo.*

Vår gruppe består av tre personer, og vi valgte Ludo i nettverk.

Prosjekt og verktøyvalg

Prosjektet valgte helt stadard verktøy:

- Eclipse Indigo (grunnet Eclipse freeze bug i andre versjoner)
- Android SDK 3.0
- ADT Version: 10.0.1.v201103111512-110841.

Debug/sjekking av kode ble først gjort vha. emulator - senere via android telefon. Telefoner benyttet var HTC HD2 (opprinnelig Windows-telefon), og modifisert til Android. Diverse Samsung telefoner er også benyttet til testing.

Noe vi ikke var klar over før siste uken av prosjektet, var at den ROM vi benyttet på HTC hadde en bug og ikke fungerte med sockets som forventet. Dette har skapt forsinkelser og mange timer med fånyttes leting etter en løsning. Etter installasjon av ny ROM fungerer det som forventet - men desverre mange uker for sent.

Emulatorene hadde den svakhet at ved oppstart av 3 emulatorer (1 server, 2 klienter), så ble det noe feil i debug-serveren, og utviklingsmiljøet kollapset med påfølgende maskin restart som eneste mulighet for å fortsette. Dette er også en feil vi føler har vært med å stanse utviklingen.

Verktøyene har allikevel vært tilstrekkelig på sine områder, selv om enkelte ting måtte forsøkes på en telefon for å se resultatet (eks. noen layout, animasjoner+lyd).

Dokumentering/språk ble ikke avtalt på forhånd, slik at noe er på engelsk og noe er på norsk. For oss som arbeider med slikt, burde det allikevel ikke være problematik å få med seg hva som er intensjonen.

Spill

Selve spillet

I denne delen vil vi beskrive spillets gang. Bakgrunnen/bildene avviker litt fra innlevert oppgave pga. mindre endringer mellom dokumenteringstidspunkt og endelig løsning.

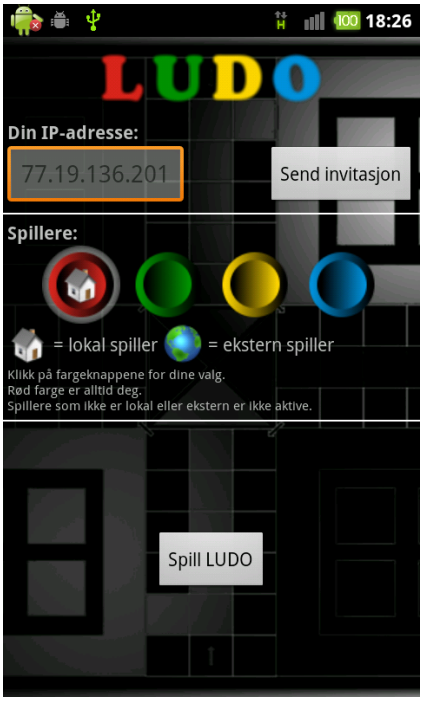
Utgangspunktet er at man spiller lokalt og/eller via nettverk.

Start opp Ludo-spillet, og du får opp valg om du enten vil starte et nytt spill eller koble deg mot en som er spill-master (for nettverks-spill).

Hvis du velger å starte et nytt spill, får du også muligheten til å bestemme noen regler og hvilket brett som det skal spilles på.

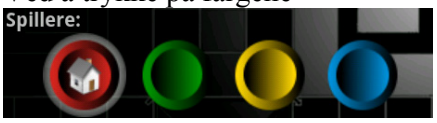
Her er to av brett-typene som kommer med første versjon av spillet.

Når du nå har satt reglene, er du klar for å starte spillet.



Her kan du velge å sende invitasjon til deltakere om du ønsker. Det er kun ip-adresse og informasjon som kommer opp til mottaker. (P.t. er det ikke lagt inn automatisk start av spillet fra en motatt sms).

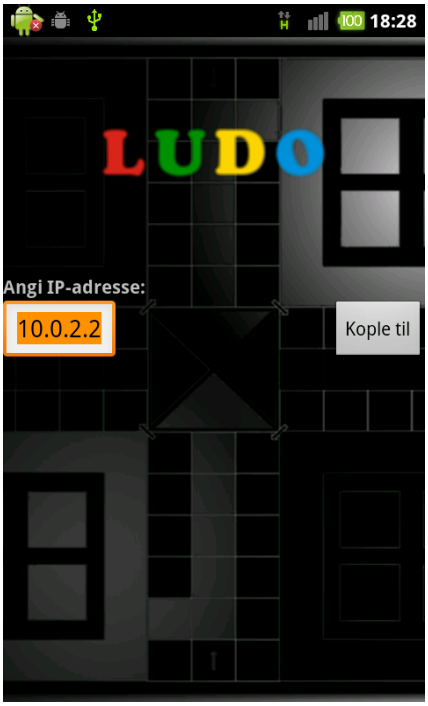
Ved å trykke på fargene



vil du kunne legge til lokale spillere, samt se hvem som kobler seg på utenfra.

Når alle er klare til spill, trykkes 'Spill LUDO' og spillet starter.

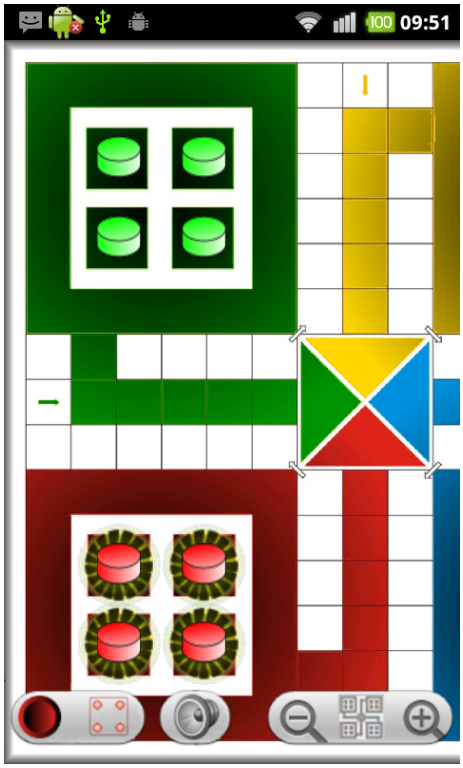
Hvis du har valgt å koble opp mot et eksisterende spill, setter du inn ip-adresse for oppkobling.



Ved tilkobling, vil farge bli tildelt, og meldingen

vil være på skjermen inntil master-spiller velger å starte spillet. Hvis det skulle vise seg at det ikke er flere tilgjengelige farger (dvs. det er for mange spillere i forhold til farger), så blir du med i spillet, men får bare se på andre som spiller.

Vi har nå startet spillet, og alle spillere ser det samme brettet.



Bildet nede til venstre angir hvem som kaster.
Deretter er det terningen du trykker på (eller rister telefonen) når det er din tur, lyd av/på og skalering.

Det er mulig å zoome inn på spillet (hvis det eks. er store brett), og du bruker fingeren til å flytte selve brettet på skjermen.

Brettet kan skaleres til ønsket størrelse med knappene til høyre. Midterste knapp skalerer til full skjerm.

Hvis en spiller forlater spillet underveis,
vil denne meldingen dukke opp – og brikkene fjernes fra spillet. De andre fortsetter som normalt.

Melding vil også vises på alle tilkoblede klienter hvis master-spiller avslutter spillet før det er ferdig.

Regler

Regler implementeres av en egen klasse, og påvirkes av settings. Noen initielle regler ble diskutert, og følgende ble satt opp:

- 2-4 personer kan delta i spillet. Hver spiller får 4 brikker av samme farge og anbringer disse i et firkantet felt (gården) av samme farge som brikkene.
- Det kastes med en terning som viser 1-6. Det spilles på brettet fra høyre til venstre. Den som har de røde brikkene begynner, nestemann til venstre fortsetter.

- Ingen brikke kommer ut av gården før eieren kaster 6. (Her kan vi kanskje legge inn valg for flere for å få fortgang i spillet (f.eks. 1 og 6)) Seinere flyttes brikkene til venstre etter terningkastene.
- Kastes på nytt 6, kan en etter ønske enten føre en brikke ut av gården eller bringe en brikke videre på feltet. Alle kast på 6 gir rett til et nytt kast. Er en ikke i stand til å flytte en brikke, mister en kastet.
- Setter et kast en spiller i stand til å besette en plass hvor en motstanders brikke står, blir denne slått ut og må begynne forfra. Altså ikke mulig å stille seg ved siden av.
- Er plassen opptatt av en av ens egne brikker, blir den nye brikken også anbrakt her. To brikker av samme farge sperrer vegen (danner port) for brikker av andre farger. Så lenge porten står, kan ingen av disse brikkene slås ut. (Denne regelen utgikk)
- Kan en motspiller ikke flytte andre av sine brikker, mister han kastet.
- Når en brikke den midtlinjen som fører til dens "hjem" (det sted på midtfeltet som har brikkens farge), føres den framover denne midtlinjen. Hjemmet nås bare ved å kaste det nøyaktige antall øyne. Kastes det for mange, må brikken bli stående der den er.
- Et tårn kan flyttes det antall plasser som man før øyne på terningen, delt på to. Får man for eksempel terningkast 4, kan man flytte et tårn 2 plasser fremover.
  Dersom man får et oddetall over 1, kan man også flytte tårnet i sin helhet på samme måte, men man er da tvunget til å demontere det for å "bruke opp" det siste øyet. Får man terningkast 5, kan man altså flytte øverste brikke 5 felter eller flytte tårnet 2 felter og øverste brikke et felt. Det er ikke mulig å lage et tårn på første felt utenfor basen. En brikke må flyttes fra dette feltet før neste brikke kan flyttes ut.
  (Her brukte vi det antall øyne som ble kastet).

Overordnet løsning

Målsetninger

Oppgaven er i og for seg en relativ enkel konseptuell oppgave – de fleste har jo spilt ludo en eller annen gang. Det å få til en klient/server-løsning har også vært en utfordring på åpent nett - ikke alt har fungert så strømlinjeformet som i teorien.

Vår løsning hadde til hensikt å dekke følgende funksjoner:
- Registrere (klient)spillere mot en (hoved) spiller.
- La spillere automatisk få tildelt farge.
- Bestemme hvem som kaster terning og vise resultatet til alle.
- Slå inn brikker/spille likt for alle.
- Mest mulig likhet for klient-/tjener-del.
- Lite nettverkstrafikk.
- Overvåkning av eget rammeverk.
- Enkel re-definering av spill – flere brett-typer.

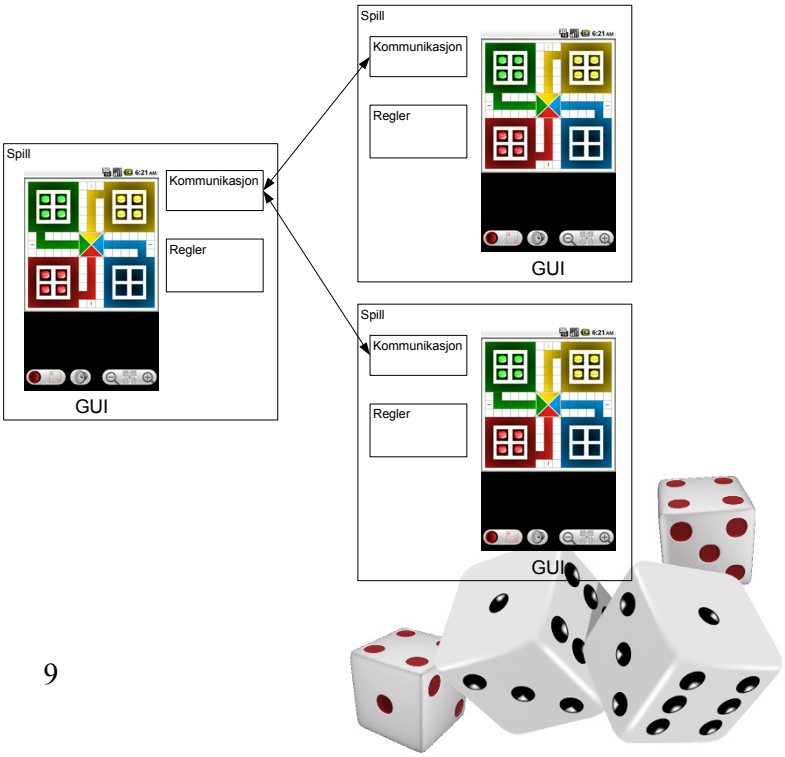
I tillegg til en løsning av oppgaven isolert sett, ønsket vi å abstrahere funksjonalitet for framtidig utvidelse og mulighet for å være litt mer fleksibel. Kommunikasjons-modulen er en motor som kan benyttes i andre sammenhenger hvor det er mange-til-mange kommunikasjon. Tilpasninger for hver type klient muliggjøres via lyttere mot kommunikasjons-delen.
Noe funksjonalitet kan i en første versjon være litt mer problematisk å legge ut som en generisk modell - dvs. hvis man ser på selve spillet (ludo, bondesjakk), regler, brett-typer (enkle/komplekse), brikker (enkle/komplekse type-basert flytting) mm. – men vi har forsøkt å legge til rette for andre typer spill så langt det lar seg gjøre - innenfor rammen av den tiden vi hadde til rådighet. Det har hele tiden vært meningen å tilrettelegge systemet for utvidelser og endringer for andre typer brettspill – og da benytte samme kodebase og konsept.

Løsningen i et falkeblikk.

En spiller opptrer som tjener, og klienter kobler seg opp mot denne.
Tjeneren bestemmer hva som skal spilles (brett), evt regelendringer etc.

Klienter kobler seg opp når tjener tillater dette. Etter en tid vil alle som skal spille være oppkoblet, og spillet kan starte.

Hver klient har sin egen lokale versjon av spillet, så det er kun kontroll-informasjon som går mellom enhetene – dvs. flyttinger og andre administrative meldinger.



Når en klient kobler seg til, vil den stå og vente inntil spillet starter. Når spillet startes, vil informasjon om andre spillere og regler bli overført[1].

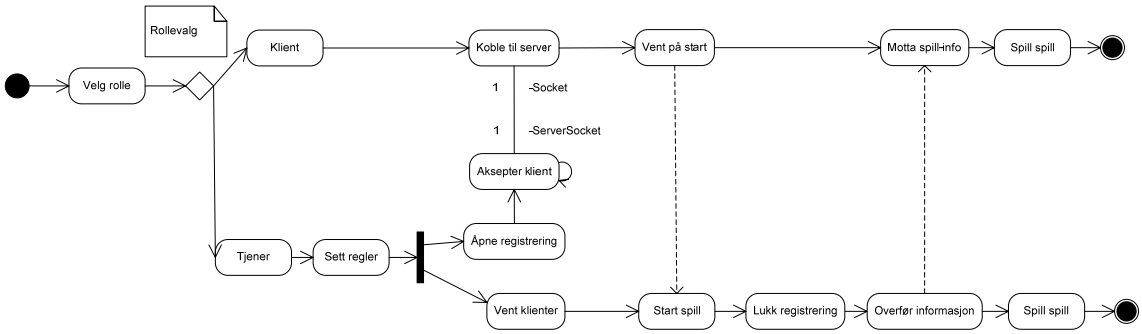
Spillet gjennomføres inntil reglene sier man har en vinner.

[1] For overføringer i spillet generelt, er det benyttet Serializable som overføres mellom klientene. For regler er det benyttet JSON bare for å vise at det er mulig med forskjellige formater.

Program state

Forenklet state diagram

Vi har ikke tatt med bruk av tilbake-knapp – dette behandles som avslutning av spill (activity).
Helt forenklet vil da en skisse over state se slik ut:



Denne forenklede state-modellen tar som sagt ikke høyde for transisjoner med user input - eks. tilbake-tast, brudd i kommunikasjon etc. Dette blir sikkert klarere når man ser på rollene/samspill mellom klassene.

Modellen benytter tråder og singletons, og en detaljert state-modell må sees ut fra flere use-case og hvordan enkelte transisjoner i modellen setter andre deler av modellen i en endret state. Detaljering av dette mener vi ikke vil tilføre oppgaven noe av merverdi i forhold til merarbeidet det medfører.

Design

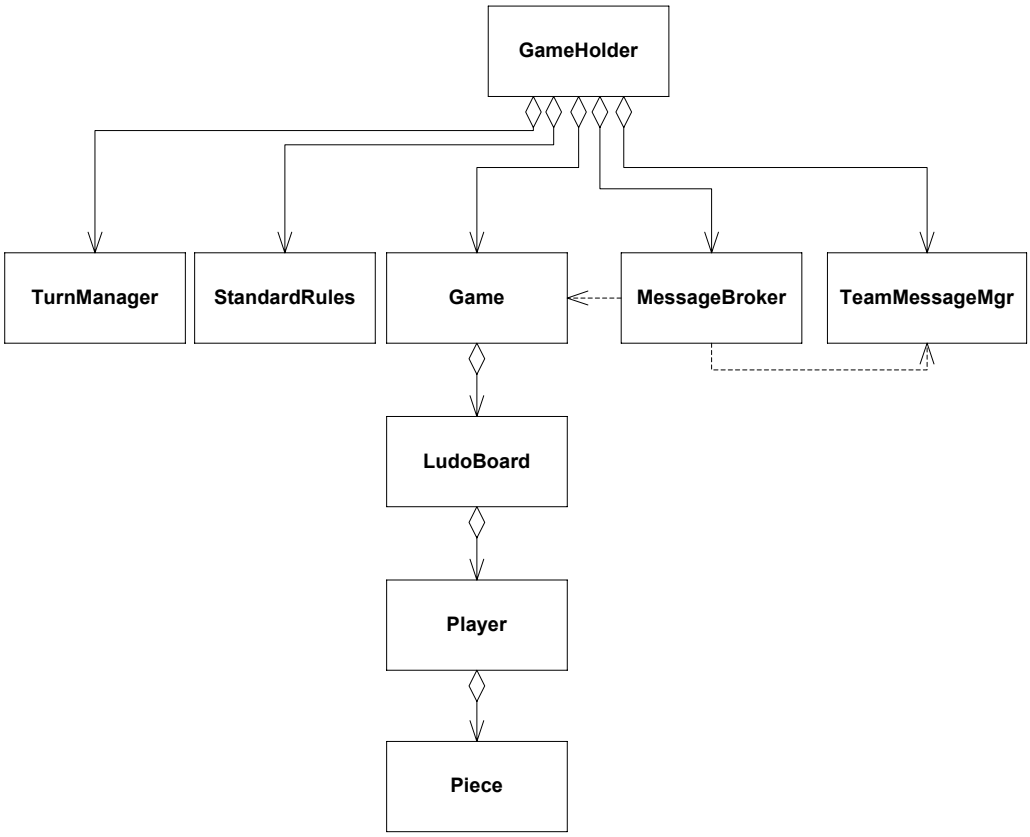
Innledning

Ett av delmålene for design var å frigjøre deler av koden for enklere implementasjon av andre typer brett, andre regler og annen kommunikasjon. Det å bygge ut en ferdig applikasjon med 'mange muligheter' tar tid og må gjennomføres for mange av de mulighetene man ser for seg, før man sitter igjen med et api eller konsept som gjør akkurat det man ønsker. Vår tanke var å lage en første versjon av kode som vi kunne bygge videre på – og gjerne med komponenter som var gjenbrukbare.

Selve kommunikasjons-delen hadde vi jobbet med fra starten av – meningen var å lage en 'mange-til-mange' kommunikasjons-modell som mer eller mindre administrte seg selv. Overordnet burde man kunne motta og sende meldinger, samt overvåke og kontrollere rammeverket fra et 'fritt' ståsted. All kommunikasjon mellom deltakere skjer 'automagisk', og spillets gang er hendelsespåvirket gjennom samme modell.

Overordnet modellperspektiv

For å gjøre konseptet klarere, er definisjoner via interface tatt vekk. Modellen er ment for å gi en forståelse for hvordan koden er satt sammen.



Selve spillet holdes sammen av **GameHolder** (singleton) siden vi skifter på hvilken Activity som har rollen som fører av spillets framgang. **Game** er representasjonen av selve spillet. Her skjer flyttingen av brikker. Det sjekkes mot regler (**StandardRules**) om brikker skal slås ut eller settes i tårn.

TeamMessageMgr tar seg av kommunikasjon med klienter. MessageBroker tolker meldinger fra klienter og handler etter dette. StandardRules er regler i spillet. **TurnManager** holder kontroll på hvem det er sin tur.

Samspillet mellom disse går via Message(handlers), slik at meldingene blir håndtert når trådene tillater dette.

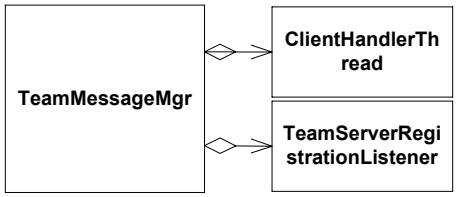
Kommunikasjon

Protokoll

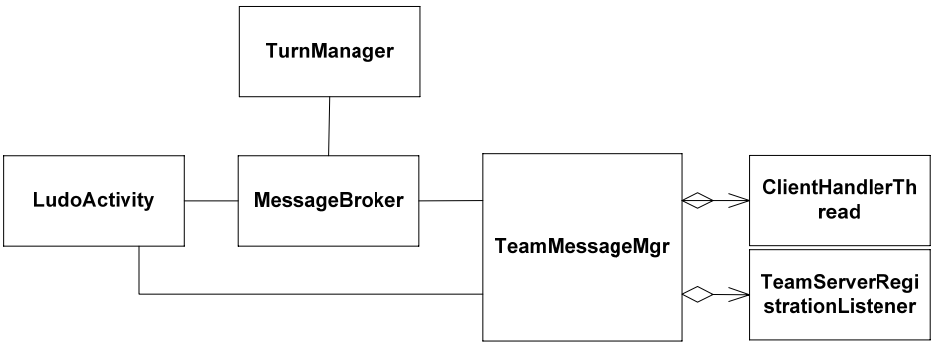
Protokollen for spillet er enkel. Vi har valgt en enkel String-type protokoll med separerte verdier.
Meldingene er prefixet med G for spill-relaterte meldinger, A for administrative meldinger. Videre er det parametre for farge, antall flytt, brikker som slås inn, vinner med mer.

Generelt

TeamMessageMgr instansierer en privat klasse - ClientHandlerThread – for hver klient-forbindelse som opprettes. Mekanismene er de samme for klient eller server.
TeamServerRegistrationListener er en klasse som starter en server-port for tilkobling av klienter. Når en klient blir connected, vil det opprettes en ClientHandlerThread for denne forbindelsen, og server er klar til ny registrering.



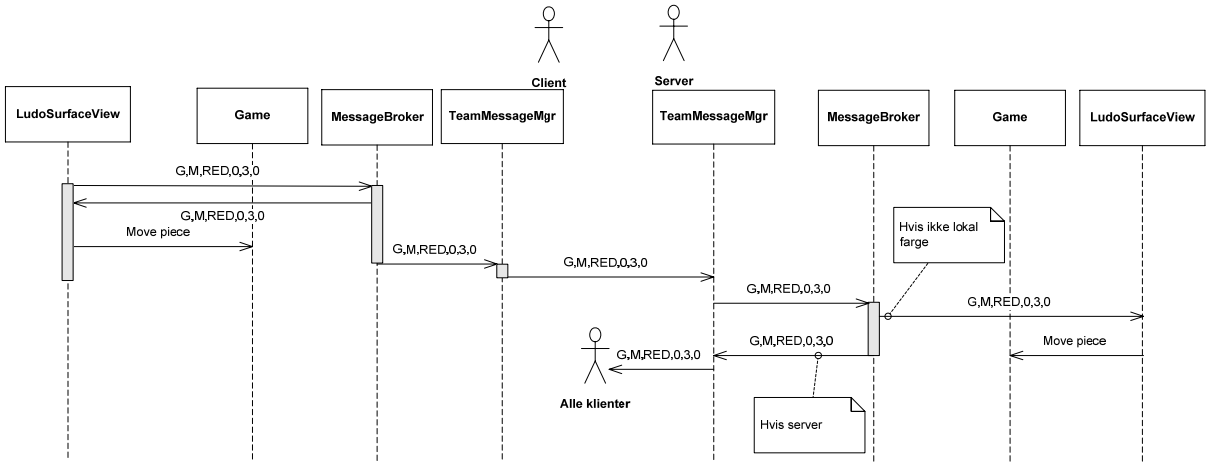
TeamMessageMgr videresender to typer meldinger – administrative eller 'content'-type (det som klientene sender seg imellom). Alle meldinger blir sendt ut til registrerte listeners. Man kan altså velge å legge inn en listener for kun det ene eller begge.



LudoMessageBroker tar som sagt i mot meldingene fra klientene og parser informasjon. (Broker kan skiftes ut til en annen klasse hvis det er en annen protokoll - informasjons-struktur - som benyttes for å kontrollere spillet.) Informasjon om flytt går videre til alle klienter i nettverket avhengig av om du er server eller klient. Rammeverket for utveksling av informasjon kjenner ikke til innhold i meldingene – dette er det broker som gjør.

Eksempel på kommunikasjon

Flytt av brikke (Rød, brikke 0, 3 flytt)



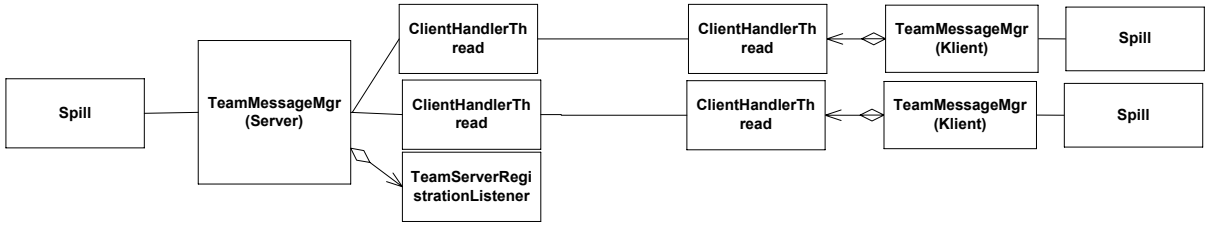
Melding om flytt av brikke sendes broker, som sender den til sine interne lyttere – og til alle eksterne. Server mottar og videresender til alle involverte. Her er det slik at hvis dette ikke er en 'lokal' farge, så flyttes brikken – ellers er den flyttet når vi opptrer som klient.

Start av spill – litt mer teknisk.

Oppstart

Spillet startes likt av alle. Ved valg av tilkoblingstype (starte spill eller koble seg til et spill), så velges også hvilken rolle du spiller i nettverket – tjener eller klient. Som tjener starter man også en TeamServerRegistrationListener som fungerer som accept på socket-forbindelser.

Det opprettes en TeamMessageMgr på alle enheter og mekanismene for tilkobling er nesten lik for alle. Resultatet av oppkoblingene er at server har flere ClientHandlerThread-klasser som leser kommunikasjonen fra klienter – klientene har kun en.



Klienter har nå LudoJoinGameActivity kjørende, progressboks for å vente på server. Når server har bestemt at selve spillet starter, sendes det ut informasjon om hvilke spillere som er med, samt settings for spillet. Spillet lastes og initieres.

Alle starter opp LudoActivity og brett-informasjon parses fra lokal xml (se litt lengre ned). Det opprettes en handler i LudoActivity for meldinger som kommer inn – men denne benyttes kun for administrative meldinger. Meldinger som omhandler selve spillet er lagt i LudoSurfaceView.

Det er nå LudoMessageBroker som 'styrer butikke'. Siden vi vet at TeamMessageManager har kjennskap til at vi har rolle som server eller klient, vil enkelte av meldingene behandles spesielt der – eks. hvem er neste spiller.

Brett/definisjoner

Selve definisjonen av spillet er abstrakt – det er ingen implementasjon av visning i klassene for spill eller brett. Implementeringen er gjort via xml og bilder. Noen refler foreligger – eks. brikkes visuelle representasjon er navngitt med antallet brikker i høyden (getId()). Vi har lagt med oste-brikker bare for å vise et eksempel på at definisjonen av brikker bare skjer i xml. (**Se brettet Psyko-ludo – gule brikker.**)

Selve definisjonen lastes i Game-klassen fra LudoActivity. Det startes opp en parser av xml-definisjonen: ParseBoardDefinitionHelper ph = new ParseBoardDefinitionHelper(); - som setter verdier i Game-klassen og LudoBoard-klassen.

Definisjoner er laget for

- Grafisk brett (informasjon om opprinnelige dimensjoner)
- Koordinater til brikkebane.
- Startposisjon for brikker
- 'Hjem til mål'-bane for hver spiller.
- Definisjon av brikke-navn (en mal) – en mapping mot grafisk representasjon.

Brikkene vet bare hvor langt de har gått på brettet – brettet (LudoBoard) vet hvor på brettet de befinner seg.

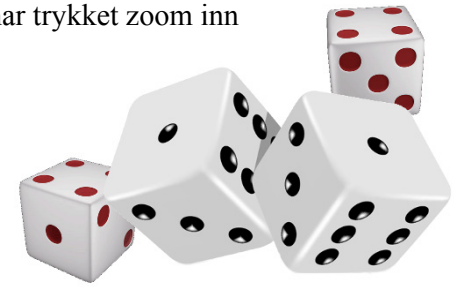
GUI

Vi har valgt SurfaceView som utgangspunkt for visningen. Her ble vi møtt med flere utfordringer. Det første var at definisjonen for den grafiske utformingen av brettet ikke samstemte med det som ble lastet opp i systemet – og det var forskjellig mellom flere typer mobiler. Det viste seg at density på display spilte en større rolle enn vi hadde forutsett. Da ble det nødvendig med re-kalkulering av alle brikke-posisjoner i forhold til ny skalering.

En annen utfordring er at vårt SurfaceView også benyttes til valg av brikke, samt flytting av selve brettet på skjermen. Vi valgte å legge til side zoom-funksjonalitet (multi-touch) i denne versjonen, og heller legge disse i layout.

Flytt av brikker

Flytt av en brikke starter i LudoSurfaceView når spiller berører brikken. Når fingeren forlater skjermen vil dette trigge en event i LudoSurfaceView med opplysninger om at en berøring har skjedd. Koordinatene til berøringspunktet blir beregnet ift om bruker har trykket zoom inn eller zoom ut på skjermen.



Dersom det er flere brikker i nærheten av berøringspunktet vil Klassen Game velge den brikken som har kortest avstand fra punktet til senter av koordinat for brikken. For å akseptere et gyldig trykk på brikke tillates det trykk på en større flate som er beregnet til et relativt området rundt senteret til brikken. Brikken er enten i ”home” posisjon, i spill rundt brettet på fellesområdet, eller på vei mot mål i eget målområde. Klassen Piece vet selv hvor på brettet en brikke befinner seg og i hvilken posisjon, samt hvilke andre brikker den eventuelt er i tårn med.

Klassen LudoBoard hjelper til med å holde en vektor med koordinater som representerer veien rundt bordet og er definert med et løpenummer og en index (x,y) for senter på feltet. Opplysningene er definert i en xml fil for hver brett-type som kan spilles. Tilsvarende data er definert for klassen Player som har felt som representerer spillerens ”home” område, samt ”way-home”-område som er målområdet til en spiller.

Når en brikke er identifisert fra berøringspunktet vil klassen StandardRules fortelle om brikken kan flyttes det antall posisjoner som terningen viser eller ikke. Hvis brikken kan flyttes sørger klassen LudoMessageBroker (beskrevet tidligere) for at brikken flyttes på lokal enhet, samt distribuerer melding til andre lokale spillere og nettverksspillere. Klassen LodoBoard organiserer så flytt av brikke til rett sted på brettet. Klassen Piece flytter brikken og finner selv ut om den er kommet i mål eller har startet mot mål i eget målområde, sammen med opplysninger om ny posisjon.

Når en brikke er flyttet sjekker Game om det finnes andre brikker på den nye posisjonen. En brikke kan flyttes til en ledig plass, slår ut en annen spiller sin brikke eller tårn, eller den kan lage tårn med egne brikker. Game sjekker med StandardRules hvilke regler som gjelder for de andre brikkene som står i samme posisjon. Datamodellen skiller mellom definisjon av regler og implementering av hvordan brikkene skal håndteres som en følge av gjeldende regler. På denne måten vil reglene lett utvides hvis andre eller tilleggsregler skal tas i bruk.

Avsluttende kommentar – arbeid til V2

Til en (eventuell) neste release har vi noen ting vi ser vi ønsker å gjennomføre foruten design-endringer som er nevnt tidligere.

- Når en spiller mottar en sms, kan spillet startes automatisk – eller at det plukkes opp en sms for oppkoblingsinformasjon i bakgrunnen når spillet startes.
- Kommunikasjon via wifi kompletteres med samme api via bluetooth.

Konklusjoner vedr. design

Når vi har kjørt spillet en stund, og ser hvordan det oppfører seg, ser man gjerne ting som kanskje kunne vært endret – både funksjonelt eller pga. stabilitet og/eller performance.

I utgangspunktet fungerer spillet/modellen etter hensikten, og målet er oppnådd. Når det ikke er optimale forhold (eks. nettverk), så er det elementer i modellen som bør revurderes.

- Kommunikasjon klient/server bør analyseres mer og settes opp med flere konkrete use-case for drop i kommunikasjon.

- Threading og TeamMessageMgr kan gjøres mer stabil (eks retry i nettverk) – problemer pga. kommunikasjon mot emulator/telefon gjorde at denne delen ikke ble så gjennomtestet som vi selv hadde ønsket.

- Et mulig re-design eller vurdering av algoritmen for kommunikasjon mellom server/klienter bør sees opp mot en mer rendyrket MVC-modell. Dette må jo også sees i sammenheng med det totale bildet mhp. hva som sendes over nettverket. Vi har bestrebet oss på lite trafikk i nettverket, men vi ser samtidig at med litt mer nettverkstrafikk er det mulig med en mer optimalisert styring av hva som er på brettet fra den sentrale spilleren (server).

(Denne siden er med hensikt uten annet innhold enn logo…
This page is intentionally left … with a logo)