Risk

Digital implementering av det klassiska brädspelet Risk

Daniel Hesslow
danhes@student.chalmers.se
Institutionen för Informationsteknik

Arvid Hast
hasta@student.chalmers.se
Institutionen för Informationsteknik

Niklas Jonsson
nikjonss@student.chalmers.se
Institutionen för Informationsteknik

Fredrik Lindevall
frelinde@student.chalmers.se
Institutionen för Informationsteknik

27 maj 2016
LSP310 V16
Kommunikation och ingenjörskompetens
Chalmers tekniska högskola

Innehåll

1 Inledning

Brädspelet Risk är en omtyckt klassiker men under lång tid har de som kunnat spela Risk begränsats till de som har tillgång till det fysiska brädspelet. Dessa personer kan dock inte spela risk med vännerna på spårvagnen eller i kön till festivalen. Även om personer sitter på olika spårvagnar borde de kunna spela med varandra. Det har med andra ord länge saknats en bra digital version av det populära brädspelet. För att det ska vara lätt att spela med andra krävs en applikation som är lätt att förstå och lätt att använda, till och med för de som aldrig spelat förut. Det behövs en anpassning av spelet som passar in i den vardag människan idag lever i med hjälp av de teknologiska framstegen som har skett under de senaste decennierna.

1.1 Bakgrund

Spelet Risk är ett klassiskt brädspel där två till sex spelare tävlar om världsdominans. Ett territorium är ett område och tillsammans bildar flera territorier världens kontinenter. Varje spelare börjar med lika många territorier var med en armé på varje territorium. Sedan turas spelarna om att placera ut ett antal arméer baserat på hur många territorier och kontinenter de äger (Hasbro, 1993). Spelare kan anfalla angränsande territorier som ägs av en annan spelare genom att slå ett antal tärningar. Dör alla arméer på fiendeterritoriet tas det över.

Det finns två stora mobila plattformar, IOS och Android. Dessa plattformar kräver olika sätt att kommunicera med operativsystemet. IOS använder sig av programmeringsspråket Objective-C och Android använder sig av Java. Applikationen är utvecklad för Android av den praktiska anledningen att projektets programmerare är mer vana vid Java. Implementeringsprocessen skulle dock vara snarlik för de båda plattformarna.

1.2 Syfte

Rapportens syfte är att beskriva processen att översätta konceptet bakom det klassiska brädspelet Risk till ett mobilspel. Slutmålet med projektet är att skapa en applikation som är enkel och intuitiv att använda, även för enheter med mindre skärmar, och som gör att spelet blir tillgängligt för fler.

Rapporten omfattar hela utvecklingsprocessen från idéstadie till en fungerande prototyp.

2 Metod

Utvecklingsprocessen började med att ta fram en domänmodell och ett antal användningsfall för att skapa en uppfattning om hur applikationen ska byggas. Applikationen utvecklades med syftet att vara lik brädspelet Risk men eftersom det finns många olika regler och sätt att spela Risk togs först beslut kring vilka regler som ska gälla. Till exempel beslutades att inte implementera några slags uppdrag eller andra tillägg som finns i senare versioner av Risk. I stället ligger fokus på enkelhet och de grundläggande reglerna som alla versioner av Risk har gemensamt. Därefter började implementeringsprocessen och de första körbara iterationerna tog form.

2.1 Grafik

När världskartan, som är huvuddelen av grafiken i Risk, ska visas uppkommer det några problem som vanliga applikationer inte behöver hantera. Eftersom världskartan är stor och detaljerad och en mobilskärm är liten måste användaren kunna zooma in och ut och dessutom panorera fram och tillbaka. Detta betyder att kartan måste kunna skalas upp och ner med hög kvalitet. I vanliga fall görs detta med hjälp av så kallad vektorgrafik där bilden inte är definierad i form av pixlar utan istället kurvor och linjer. Dessutom behöver applikationen kunna byta färg på regioner dynamiskt och ta reda på vilka regioner som gränsar till vilka.

De existerande biblioteken för vektorgrafik på Android uppfyllde inte dessa krav, därför implementerades ett på egen hand.

Från början var planen att mjukvarurendrera allting genom att gå igenom varje pixel och se om den är innanför eller utanför en given form, men även om detta skulle vara möjligt på en modern dator är det ingenting som en mobilprocessor klarar av. Därför blev utvecklarna tvungna att använda mobilens grafikprocessor (GPU) och därmed den enda applikationsprogrammeringsgränssnittet för grafikkortet på mobiler, *OpenGL ES*. En SVG-fil (Scalable Vector Graphics) över världskartan skapades. OpenGL använder trianglar för att rita ut former på skärmen, vilket betyder att vektorgrafik-formerna som fanns i SVG-filen behövde konverteras till trianglar.

2.1.1 Inläsning av indata

Huvudsakligen består filen av XML (Extensible Markup Language). Varje form i XML-dokumentet har ett dataelement som enkelt kan läsas in med hjälp av en *Recursive Descent Parser*. När kartan blev mer och mer detaljerad (se fig. 1) växte filen till närmare 150KB vilket tar flertalet sekunder att läsa in varje gång en användare vill starta ett nytt spel. Detta är inte acceptabelt men genom att läsa in filen när applikationen startade, ersätta Javas klass, *Scanner*, och buffra indatan löstes detta problem.

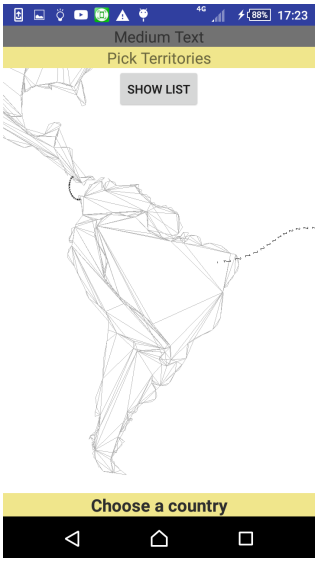
Figur 1: Indatan för Applikationen

2.1.2 Konvertering till GPU-data

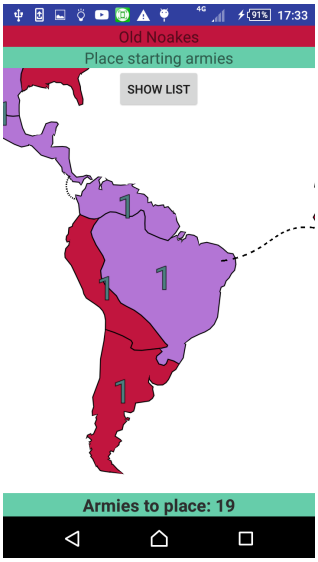
I SVG-filen finns även linjer som delar upp kontinenter i mindre regioner, visar vilka regioner på olika kontinenter som arméer kan färdas mellan samt linjer som slår samman kontinenter och regioner . Med hjälp av *de Casteljau's algoritm* kan de former som behöver delas på separeras. I det här skedet finns listor med punkter som alla representerar flera kurvor som tillsammans utgör en region (se fig 2).

Varje region approximeras med en polygon som sedan trianguleras. En naiv approximering genom att evaluera varje kurva som formerna är gjorda av med jämna mellanrum visade sig skapa ojämn precision. De kurvor som var långa blev synligt hackiga och de kurvor som var korta använde många trianglar i onödan. Istället delades varje kurva upp i två mindre kurvor tills dess att varje kurva var liten nog. Detta gav ett bättre resultat med jämnare precision. För att triangluera polygonen (se fig 3) används en

Figur 2: Inläst data



Figur 3: Triangulerad



Figur 4: Slutgiltigt resultat

metod som kallas *Ear Clipping*. Därefter skickas informationen till OpenGL och en projektionsmatris skapas utifrån zoom- och panoreringsvärderna för att få fram den önskade bilden på rätt plats i rätt skala samt måla i önskad färg (se fig 4).

Numren som visar hur många arméer som finns i varje territorium renderas i processorn på en bild som sedan skickas till grafikenheten och där renderas på så kallade 'Textured quads'. Eftersom dessa bilder bara behöver skapas en gång och sedan kan sparas är detta mycket effektivt.

2.2 Model-View-Controller

När grafiken fungerade började arbetet med att ta fram en modell för spelet och en tillhörande vy och kontroller för att interagera med gränssnittet och uppdatera modellen. En klasstruktur började ta form med klasser för spelare, territorier, tärningar och en huvudklass. Tanken var att huvudklassen, helt enkelt kallad Risk, skulle ha referenser till spelar- territorie- och tärningsobjekt och ansvara för både spelflödet och för att uppdatera egenskaperna hos resterande objekt som utgör modellen.

2.2.1 Problem

Efter ett tag uppstod problem då det inte fanns mycket planering och klassernas referenser till varandra blev ganska röriga. Ska spelarna ha

referenser till ägda territorier eller ska territorierna ha en referens till sin ägare? Ska tärningarna vara en egen klass eller helt enkelt en funktion i spelflödet? Framförallt blev det uppenbart att modellen och kontrollern inte separerades på något meningsfullt sätt då huvudklassen Risk både hanterade modellen och spelflödet och därmed anropades direkt när spelaren interagerade med gränssnittet.

Problemen med modellen löstes genom att gå igenom ett antal användningsfall för att bilda en bra uppfattning om vilken information som behövs när och i vilka sammanhang. Till exempel gavs territorieobjekten varsin referens till ett spelarobjekt som ägare, och spelarobjekten ett heltal för antalet ägda territorier då det är det enda som egentligen är relevant för spelarobjektet och som inte är lättillgängligt genom att kontrollera enskilda territorieobjekt. Problemet med kontrollerns separation från modellen löstes genom att skapa en separat kontrollerklass (kallad Controller) där hela spelflödet sköts och interaktion med gränssnittet hanteras. Klassen Risk används nu endast för egenskaper hos flödet, som till exempel vems tur det är, och för att ha referenser till och hantera övriga delar av modellen. Därmed beskriver modellen spelet men är helt oberoende av vyn och grafikimplementeringen.

2.2.2 Ansvarsområden

När modellen var någorlunda komplett och kontrollern fungerade som den skulle skapades en vy-klass med syftet att observera modellen och baserat på tillstånd hos denna uppdatera grafiken. Allteftersom vy-klassen implementerades och mer av logiken bakom spelet visades på skärmen visa det sig att många av kontrollerns ansvarsområden, till exempel nuvarande spelare och nuvarande fas i spelet, också ska visas av vyn och därför borde vara egenskaper hos modellen.

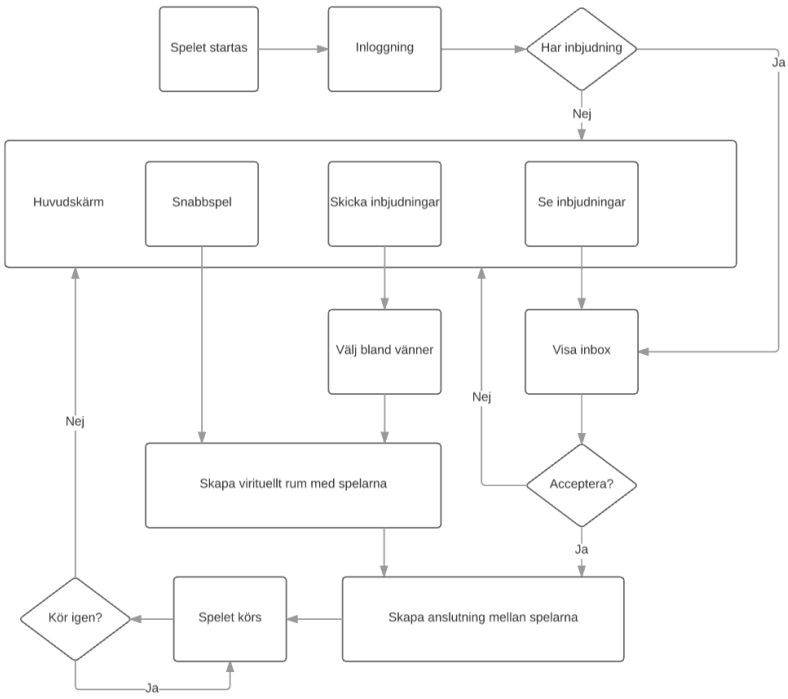
Förhållandet mellan modellen och kontrollern skiftade nu. Förut lagrade modellen endast spelares och spelobjekts egenskaper och kontrollern skötte all logik och uppdaterade modellen beroende på nuvarande förhållanden. Nu frågar i stället kontrollern modellen efter nuvarande förhållanden och evaluerar efter det vad i modellen som ska uppdateras. Detta är nödvändigt om kontrollern ska behålla sin separation från vyn. Vyn ändras beroende på modellen och kontrollern ändrar på modellen men kontrollern kan inte säga till vyn att ändras, alltså måste alla egenskaper som ska visas i vyn vara egenskaper hos modellen.

2.3 Utveckling av flerspelarläge på olika enheter

För att göra spelet roligare och mer flexibelt fanns flerspelarfunktionalitet som mål för projektet, alltså att göra det möjligt att spela med personer på andra enheter över internet. Istället för att sätta upp en egen server används en mer färdig lösning, till stor del på grund av projektets tidsbegränsning.

2.3.1 Google Play Services

Lösningen som används är en del av Google Play Services, som är tillgängligt om avgiften för ett utvecklarkonto hos Google är betald (25 USD). Spelarna ansluts till en av Googles servrar då de loggar in med sitt Google-konto. Efter att en spelare är inloggad kan den välja mellan tre alternativ för att spela med andra; Snabbspel, bjud in vänner eller se inbjudningar. Efter att spelarna är valda för en spelomgång kopplas spelarna ihop enhet till enhet (engelska peer to peer), om möjligt. Om detta inte kan göras skickas nätverkssignalerna via en av Googles servrar istället. Logiska flödet för exakt hur en spelomgång skapas och körs visas i figur 5.



Figur 5: Flöde för flerspelaromgång mellan olika (Google, 2013)

2.3.2 Integration med modellen

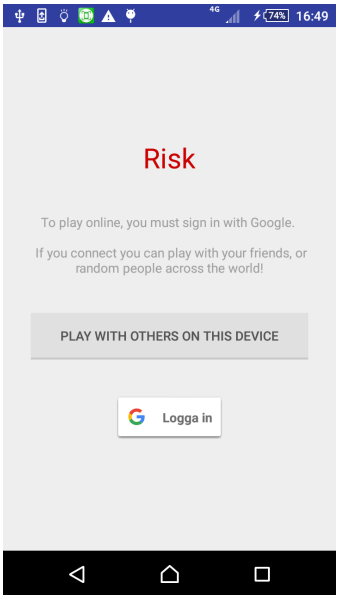
När spelarna väl är sammankopplade synkroniseras spelarnas modeller genom att låta en klass för nätverkshantering observera modellen precis som vyn. När något relevant i modellen ändras skickar denna ett meddelande till de andra spelarnas applikationer. Dessa anpassas för att beskriva en av tre händelser; antalet arméer på ett territorium har ändrats, ett territorium har en ny ägare eller turen har ändrats till en ny spelare.

Meddelandet tas emot av resterande spelare, tolkas, och ändrar sedan genom kontrollern den egna modellen på samma sätt. Dessa tre händelser är det enda som faktiskt behöver synkroniseras mellan de olika modellerna. Resterande egenskaper är bara relevanta för att uppdatera den egna vyn eller för dina egna handlingar, till exempel vilka territorier som är valda för att kriga eller flytta arméer och vilken spelfas spelaren befinner sig i.

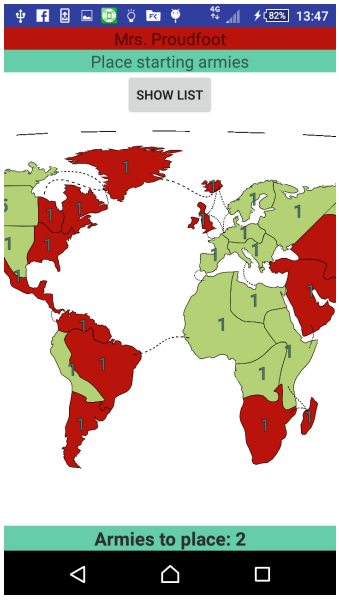
3 Resultat

Den slutliga prototypen fungerar bra och uppfyller alla krav som ställdes i början av projektet. Den fungerar på alla Android-enheter förutom ett försumbart litet antal enheter som kör en Android-version äldre än 4.0.3 som släpptes 2011. Applikationen är lättanvänd och det är möjligt att spela både på en delad enhet eller på varsin med hjälp av Google Play Services.

När applikationen först startas visas en startskärm med en knapp för att starta ett spel för flera spelare på enheten och en knapp för att logga in på Google Play (se fig 6). Loggar spelaren in visas nya knappar för att starta ett snabbspel, bjuda in vänner att spela, se spelinbjudningar och för att logga ut. När spelaren loggat in kan den spela med andra över internet. När ett spel startas fylls hela skärmen av världskartan och en dynamisk överläggsgrafiken (engelska UI-Overlay) med diverse information och knappar. Båda dessa uppdateras av vy-klassen när relevanta egenskaper hos modellen ändras och båda används dessutom för att genom kontrollern manipulera modellen när knappar trycks på eller territorier väljs. Panorering och skalning av världskartan sköts helt av OpenGL-implementeringen och berör inte på något sätt modellen eller kontrollern.



Figur 6: Första skärmen



Figur 7: Spelskärmen

3.1 Spelets gång

Spelarna turas om att slutföra varsin tur. I början av spelet får varje spelare ett antal arméer och turas sedan om att välja varsitt territorium att ockupera med en armé genom att trycka på territoriet. När alla territorier är ockuperade fortsätter spelaren på samma sätt i tur och ordning att förstärka de ägda territorierna genom att placera ytterligare arméer tills inga arméer finns kvar att placera.

Därefter består varje tur av tre faser; *placera arméer, anfall, flytta arméer.*

I början av turen får den nuvarande spelaren ett antal arméer att placera. I fasen *placera arméer* placeras de nya arméerna ut på ett eller flera ägda territorier genom att välja territoriet, välja antal arméer att placera med reglaget (engelska *Slider*) som kommer fram och bekräfta med knappen *Place*. När alla arméer är placerade börjar anfallsfasen.

Under fasen *anfall* kan spelaren välja ett eget territorium med mer än en armé att anfalla ifrån och sedan ett angränsande fientligt territorium att anfalla. För att bekräfta anfallet trycker spelaren på knappen *Fight!* som kommer fram när både ett anfallsland och ett försvarsland är valda. Anfallets utfall består i att ett litet antal anfalls- eller försvarsarméer dör och utfallet bestäms av ett antal tärningsslag som simuleras av

applikationen. Dör samtliga försvarsarméer ockuperas territoriet av anfallaren. Spelaren får anfalla ett obegränsat antal gånger och väljer när den är klar med denna fasen genom att trycka på knappen *Next Phase*.

När anfallsfasen avslutas börjar den slutliga fasen: *flytta arméer*. Under denna fasen får spelaren flytta dina arméer till valfria angränsande territorier genom att välja ursprungsland, land att flytta till, och sedan använda reglaget och knappen *Place* på samma sätt som i placeringsfasen. När spelaren är klar med denna fasen trycker den på knappen *Next Turn* och turen ges till nästa spelare. Spelet fortsätter tills en spelare uppnår världsdominans genom att ockupera samtliga territorier på kartan.

4 Diskussion

Den slutgiltiga applikationenen är intuitivt och enkel att spela. Layouten är överskådlig trots de små skärmarna som används och spelarens möjligheter presenteras tydligt. Fokus har dock legat mycket på bra tekniska lösningar och en ren och enkel design. För att göra spelet mer tilltalande är det möjligt att ett större fokus borde ligga på effekter och grafik som inte bara presenterar data på ett överskådligt sätt utan också ger spelet en speciellt känsla. Spel som lyckas bra på Play Store (Androids portal för applikationsnerladdning) är det ofta enkla spelkoncept där huvudfokus snarare ligger på något visuellt än på spelfunktionaliteten.

4.1 MVC

Programmets MVC-struktur blev inte lika självklar som tänkt då visuell information gavs till användaren både genom OpenGL och en överläggsgrafik med information och knappar och eftersom det inte var en självklarhet vad som var information specifik för kontrollern och vad som borde vara egenskaper hos modellen. Detta löstes genom att flytta data från kontrollern till modellen och med en separat vy-klass som ändrar på berörd grafik och överläggsgrafiken när modellen förändras.

4.2 Grafik

Valet att implementera vektorgrafikshanteringen själva istället för att använda existerande bibliotek har visat sig influera resultatet positivt. Den initiala tid det tog att lära sig OpenGL kan mer eller mindre likställas med den tid det skulle ta att lära sig det bibliotek som skulle använts istället.

Tiden det tar att implementera grafiken kan således jämföras med den tid det skulle ta att integrera det potentiella biblioteket med applikationsspecifik kod. Den lilla tid som potentiellt kunnat sparats om det hade används ett existerande bibliotek skulle inneburit en oproportionellt stor förlust i minskad flexibilitet. I slutskedet av projektet implementerades till exempel en funktion som visar delarna på kartan längst västerut strax till höger om det som är längst österut eftersom jorden är rund. Detta är en funktion som ett existerande bibliotek med all sannolikhet inte kunnat genomföra. På samma vis kan den implementerade grafiken gradvis få ett territorium att byta färg vilket ett bibliotek inte skulle klarat av. Spelet ger således ett mer professionellt och polerat intryck än vad det hade gjort med ett existerande bibliotek.
Eftersom grafiken är implementerad med hjälp av OpenGL finns det stora möjligheter att vidareutveckla grafiken för att den ska bli mer tilltalande.

4.3 Nätverk

Google Play Services (Google, 2016) erbjöd ett bra sätt att möjliggöra spel mellan olika enheter men det ställde också en hel del krav på implementeringen. Eftersom spelet är turn-based var responstid inget problem, så länge den håller sig inom rimliga gränser. Däremot blev det en utmaning vilken data som ska skickas och på vilket sätt. Att spelet spelas på mobila enheter med en mobil internetanslutning ställer krav på hur mycket data det är rimligt att skicka och hur ofta, ett problem som stöts på lika fort på en desktop-applikation, speciellt inte om datan bara skickas över lokala nätverk. Implementeringen fungerar bra och är framförallt väldigt snål med hur mycket data som skickas.

Referenser

Google. (2013). *Mobile multiplayer made manageable.* Hämtad 2016-05-9, från `https://youtu.be/MGkHszLrV7g`

Google. (2016). *Starting a real-time multiplayer game.* Hämtad 2016-05-18, från `https://developers.google.com/games/services/android/realtimeMultiplayer#starting_a_real-time_multiplayer_game`

Hasbro. (1993). *Risk rules.* Hämtad 2016-05-18, från `http://www.hasbro.com/common/instruct/risk.pdf`