Spelmotor för Rundradions Pidro-projekt

Krister Bäckman

Examensarbete

Informationsteknik

2010

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	2780
Författare:	Krister Bäckman
Arbetets namn:	Spelmotor för Rundradions Pidro-projekt
Handledare (Arcada):	Jonny Karlsson, Hanne Karlsson
Uppdragsgivare:	Rundradion Ab - Svenska YLE

Sammandrag:

Projektets mål var att utveckla en prototyp av ett Pidro-kortspel i Adobe Flash. Rundradion Ab - Svenska YLE fungerade som uppdragsgivare. Spelet består av en server- och en klientapplikation. Serverapplikationen består av en spelmotor och en datorspelare med artificiell intelligens. Klientapplikationen används av människospelare. Examensarbetet handlar om spelmotorn för spelet. Spelmotorn är utvecklad i ServerSide ActionScript och körs på Flash Media Server. Servern och klienterna kommunicerar med varandra över RTMP-protokollet. Spelmotorn sköter kommunikationen mellan de tre delarna av spelet. Spelmotorn klarar av att hantera flera samtidiga spel med olika kombinationer av människo- och datorspelare, ändå så att det alttid finns minst en människospelare i ett spel.

I den första delen av examensarbetet presenteras utvecklingsverktygen, programmeringsspråket (SSAS) och Adobe Flash. Den andra delen förklarar hur spelmotorn är uppbyggd, hur klasserna fungerar och vilka designval som påverkade projektet mest. Datasäkerhetsaspekter behandlas och förslag ges på hur spelmotorn kan utvecklas vidare så att det skall vara svårare att genom fusk få en fördel i spelsituationer.

Examensarbetet ger förslag till hur spelmotorn kan vidareutvecklas och gör en slutsats att Adobe Flash är lämplig för spelutveckling

Nyckelord:	Spelmotor, Rundradion – Svenska YLE, Flash, ServerSide ActionScript, Flash Media Server, delade objekt
Sidantal:	48
Språk:	Svenska
Datum för godkännande:	29.4.2010

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	2780
Author:	Krister Bäckman
Title:	Game engine for Rundradio's Pidro-project
Supervisor (Arcada):	Jonny Karlsson, Hanne Karlsson
Commissioned by:	Rundradion Ab - Swedish YLE

Abstract:

The overall objective was to develop a prototype of a Pidro-card game in Adobe Flash. The project was commissioned by Rundradion Ab – Swedish YLE. The game consists of both a server and a client application. The server application on the other hand consists of the game engine and a computer player with artificial intelligence. The client applications are used by human players. This degree thesis deals with the game engine for the game. The engine is developed in Server Side ActionScript and runs on Flash Media Server. The server and clients communicate with each other over the RTMP protocol. The engine handles the communication between the three parts of the game. The game engine can handle multiple simultaneous games with various combinations of human and computer players; however there must always be at least one human player in a game.

The first part of the thesis presents the development tools, the programming language and Adobe Flash. The second part explains how the game engine is designed, what classes are used and what design decisions affected the project most. Security is discussed and proposals are put forward for how the game engine must be designed so that it will be harder to get an advantage in game by cheating.

The thesis provides suggestions to how the game engine can be developed further and make a conclusion that Adobe Flash is a suitable tool for game development

Keywords:	Game engine, Rundradion – Swedish YLE, Flash, Server-Side ActionScript, Flash Media Server, Shared objects
Number of pages:	48
Language:	Swedish
Date of acceptance:	2010-04-29

INNEHÅLL

Figurer

Förkortningar

AI	Artificial Intelligence
DLL	Dynamically Linked Library
DoS	Denial of Service
DDoS	Distributed Denial of Service
FMS	Flash Media Server
FMIS	Flash Media Interactive Server
FMDS	Flash Media Development Server
FMSS	Flash Media Streaming Server
IP	Internet Protocol
MIT	Massachusetts Institute of Technology
RPC	Remote Procedure Call
RTMP	Real-Time Messaging Protocol
RTMPE	Encrypted Real-Time Messaging Protocol
RTMPS	Real-Time Messaging Protocol with SSL
SSAS	Server-Side ActionScript
SSL	Secure Socket Layer
SVN	Subversion

FÖRORD

Skrivandet av detta examensarbete har varit en lång process och det drog ut på tiden p.g.a. jobb. Själva prototypen för spelet blev klart för över ett halvt år sedan.

Jag vill tacka mina handledare Jonny Karlsson och Hanne Karlsson som var till stor hjälp när jag skrev denna rapport. Jag vill också tacka de andra i utvecklingsteamet, Ron Holmström och Peter Saloviin för ett lyckat projekt.

Till sist vill jag också tacka uppdragsgivaren för chansen att delta i Pidro-projektet.

Helsingfors, april 2010
Krister Bäckman

1 INLEDNING

1.1 Målsättning och syfte

Detta examensarbete behandlar ett programutvecklingsprojekt beställt av Rundradion Ab - Svenska YLE (härefter Rundradion). Syftet med programutvecklingsprojektet var att utveckla en prototyp för spelet Pidro som går att spela over nätet. Prototypen motsvarar kravspecifikationen i Bilaga 1.

Den färdiga prototypen kan av Rundradion vidareutvecklas och publiceras på Internet (se kapitel 1.4). Eftersom spelet kommer att spelas över Internet kommer stora krav att ställas på datasäkerhetsaspekter.

Rapporten består av två delar. Den första delen presenterar utvecklingsverktygen och utvecklingsmiljön. Den andra delen visar klassdiagram, hur spelmotorn är utvecklad, vilka problem som löstes under utvecklingen och hur datasäkerheten i spelmotorn är uppbyggd.

Det sista kapitlet kommer att svara på följande forskningsfrågor.

Är Adobe Flash en bra plattform för spelutveckling?
Vilken design skall väljas för spelmotorn?
Var skall spellogiken placeras?
Vilka utvecklingsverktyg skall användas?
Vilken design motverkar fusk?
Påverkar den prestandan?
Hur kan fusk förhindras?
Hur påverkar plattformen datasäkerheten?
Hur försäkrar man sig om att bara autentiserade användare får spela?

1.2 Utförande

Arbetet utfördes som ett gemensamt projekt med två andra studeranden från Arcada. Detta examensarbete behandlar utvecklingen av spelmotorn.

Spelet utvecklades i Adobe Flash (se kapitel 2.1). I utvecklingen av spelmotorn för Pidro-spelet användes FlashDevelop för programmeringen, Mantis för buggspårandet (eng. bugtracking) och Subversion för versionshanteringen av koden (se kapitel 2.4).

1.3 Begränsningar

Det antas att läsaren är bekant med begreppet “streaming media” och kortspelet Pidro. Referenser till litteratur och bilagor som behandlar dessa ämnen finns dock angivna i texten.

Det blev överenskommet med uppdragsgivaren att de sköter om testningen men spelet har ”ad-hoc”-testats under utvecklingen så att den motsvarar en alfaversion av den färdiga produkten.

1.4 Uppdragsgivaren

Rundradion är ett aktiebolag som ägs till 99,9% av den finska staten. (Rundradion 2009a) Rundradions uppgift som ett kommunikationsbolag med allmännyttig verksamhet är att erbjuda TV- och radioprogram för alla finländare på lika villkor. (Rundradion 2009b)

Radio X3M är en finlandssvensk radiokanal under Rundradion som spelar modern musik och följer aktivt med betydande aktuella händelser. Radio Vega är en radiokanal med “äldre vuxna” som målgrupp, där samhällsfrågor, livsstil, kultur och underhållning hör till det primära innehållet (Rundradion 2009c)

Enligt Rundradion kommer Pidro-spelet till en början att kunna spelas på både X3M-communityn som är Radio X3M:s hemsida och på Radio Vegas hemsidor.

1.5 Definitioner

Spelmotor

Spelmotorn sköter om spelets gång, lagring av speldata, kommunikationen mellan klinterna och servern och AI.

AI

Datorspelare med artificiell intelligens.

Flash

Flash är Adobes plattform för multimedia. Den används främst till att lägga till animationer och interaktivitet till webbsidor. Flash kan också användas för att strömma ljud och video över webben. Flash innehåller ett scriptspråk som kallas för ActionScript.

Pidro

Ett kortspel likt bridge med speciellt stor popularitet i svenska Österbotten.

Subversion

En versionshanteringsprogramvara.

Plattform

Med plattform menas den programvara som möjliggör en funktion inom informationsteknik.

Alfatestning

Med alfatestning menas att mjukvaran har testats av utvecklarna under utvecklingen men inte av en större publik så som i öppen betatestning.

2 UTVECKLINGSVERKTYG

2.1 Adobe Flash

Uppdragsgivaren ville att spelet skulle utvecklas på Adobe Flash (härefter Flash). I Flash installeras värddatorerna med Flash Media Server-programvaran (härefter FMS). Utvecklingen av spelmotorn gjordes med Server-Side ActionScript (härefter SSAS).

Det finns tre versioner av FMS, Flash Media Streaming Server (härefter FMSS), Flash Media Interactive Server (FMIS) och Flash Media Development Server (härefter FMDS). FMS kommer härefter att användas då det menas alla tre versioner. FMDS är utvecklingsversionen av FMIS. Den behöver ingen licens. FMDS har en begränsning på maximalt 10 anslutningar. (Adobe 2009a) FMDS har använts i detta examensarbete som grund för spelmotorn.

Eftersom Adobe inte har publicerat källkoden för FMS är det den enda programvaran för tillfället som man kan utveckla SSAS för. Licensen för FMS förbjuder förändringar och alla försök att få reda på källkoden för programvaran. (Adobe 2010b)

Spelet utvecklades på FMDS p.g.a. att den var den enda tillgängliga programvaran för att utveckla serverapplikationer för Flash.

FMSS fungerar som en plattform för ”streaming media”. Denna version av FMS erbjuder två olika sorters tjänster. Direktsändning av video över nätet och en s.k. vod (video on demand) tjänst där klienten kan koppla upp till servern och be om att få se en videosnutt som är sparad på servern. (Adobe 2009a)

För en mer ingående förklarning på ”streaming media” se Michael Topics bok Streaming Media Demystified. (Topic 2002:10)

FMIS och FMDS innehåller samma funktioner som FMSS och utökar funktionaliteten med SSAS och Client-Side ActionScript (härefter CSAS). SSAS och CSAS kan användas t.ex. för att utveckla mer avancerade videoapplikationer eller spel. FMIS och FMDS kan utökas med kopplingar till C++. (Adobe 2009a)

FMS stöder Microsoft Windows Server 2003 med Service Pack 2, Microsoft Windows Server 2008, RedHat 4 och 5.2. I utvecklingen av spelet installerades FMDS på Microsoft Windows Vista Home Basic. FMS kan även installeras på 64-bit operativsystem men körs då i 32-bit läge. Adobe erbjuder FMS endast på engelska. (Adobe 2010a)

För administration av FMS används Flash Media Administration Console (härefter FMAC). FMAC behandlas i kapitel 2.3.

På Windows maskiner installeras FMS som en ”service” och i Linux miljö kan man starta FMS med ”initscripts”.

På FMS kan man starta upp flera instanser av samma serverapplikation. Instanserna körs i var sin virtuella omgivning så att de är helt isolerade från varandra.

Klientapplikationerna för Flash kan köras på Adobe Flash Player, Adobe AIR och Adobe Flash Lite. (Adobe 2009a)

När man utvecklar Flash-komponenter, som skall köras på någon klientapplikation för Flash, sparas filerna med ”.fla” ändelsen. När applikationen är färdig att överföras till webben komprimeras ”.fla”-filerna till ”.swf”-filer. För att kunna spela ”.swf”-filer måste en Flash-spelare vara installerad på datorn. (Prayaga, Suri, 2007:11)

Kommunikationen mellan klient och server i Flash sker via RTMP-protokollet (Real-Time Messaging Protocol). (Adobe 2009a)

2.2 Server-Side ActionScript

SSAS är Adobes namn för JavaScript 1.5 eller ActionScript 1. SSAS baserar sig på ECMAScript Language Specification Edition 3 och körs på Mozillas SpiderMonkey-motor. SpiderMonkey är inbäddad på FMIS och FMDS. SpiderMonkey kompilerar och exekverar SSAS-script. (Adobe 2009a:8) (Wikipedia 2010a)

SSAS har färdiga klasser för bl.a. applikationen, klienterna, delade objekt och nätförbindelser. Klasserna sköter funktioner som inloggning, nätverkskommunikation, åtkomstkontroll och informationslagring. Dessa klasser har använts i olika lösningar i examensarbetet. Förutom de klasser som är specifika för Flash Media Server kan man också använda JavaScripts egna klasser. (Adobe 2009b)

Fastän SSAS stöder klassiska arv är ”prototype chaining” eller ”prototype based inheritance” den metod som ursprungligen var menad för att skapa arv i ECMAScript. (Zakas 2005:109)

För att skapa en klass i SSAS gör man på följande sätt:

```
function Game() {
}
```

Metoder och variabler deklareras enligt följande:

```
function Game(host) {
     // privat metod
     function initPlayersSo() {
     …
     };

     // privilegierad metod
     this.initGameDataSo = function() {
     …
```

```
    };
```

Privilegierade metoder har tillgång till privata variabler och metoder och är tillgängliga från publika metoder och andra ställen.

```
    …
    // medlemsvariabel
    var m_host = host.loginName;
    …
};
```

Publika metoder deklareras med ”prototype”:

```
function Game() {
    …
};
…
Game.prototype.initPlayerIds = function() {
    …
};
```

ECMAScript skiljer sig från traditionella programmeringsspråk på så vis att klasser byggs upp med ”prototype”-funktionen och språket har ingen separat ”class”-instruktion.

De SSAS klasser som använts i utvecklingen av spelmotorn gås igenom i kapitel 3.2.

2.3 Flash Media Administration Console

FMAC är administrationsgränssnittet för FMS. Med FMAC kan man administrera användare, användarrättigheter och applikationer på FMS.

För att komma in på FMAC måste man gå via inloggningsgränssnittet. Gränssnittet består av en inloggningslucka och har länkar till dokumentationen. FMAC är en

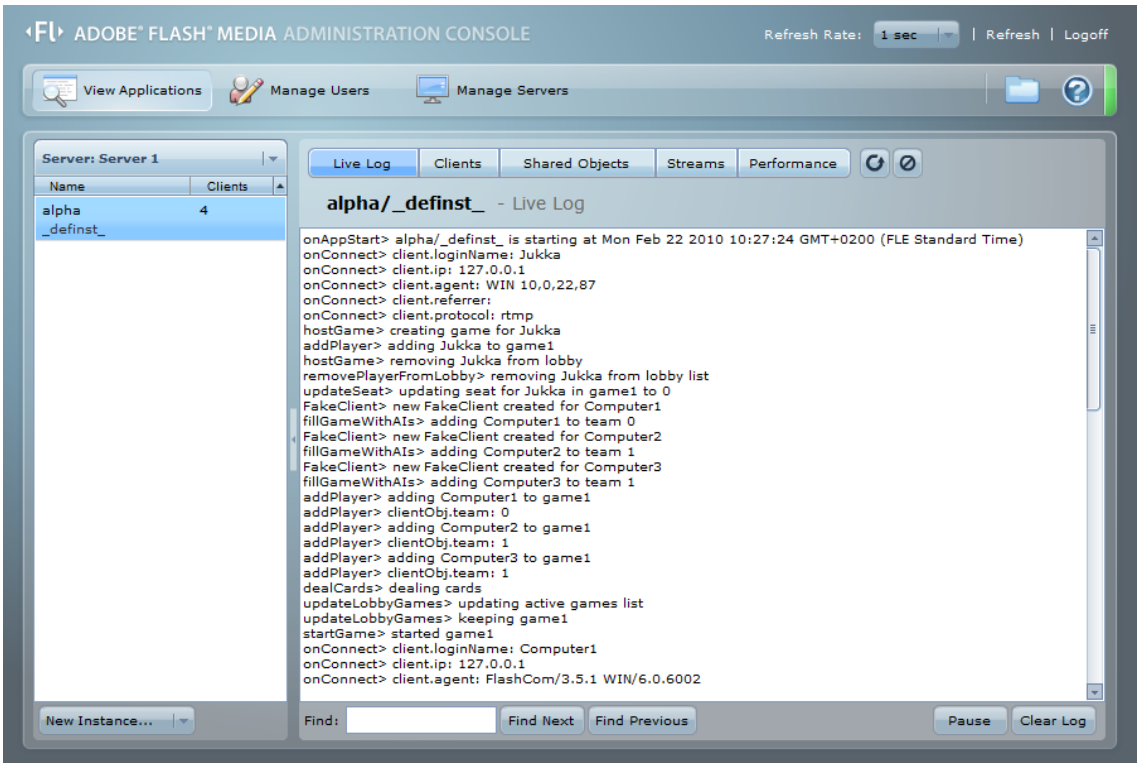
applikation i Flash som används via webbläsaren. Med FMAC kan man logga in på en lokal FMS eller på en annan FMS via ett nätverk.

Verktyget används för felsökning och för att komma åt att se loggen för de applikationer som körs, och deras delade objekt. Man kan också se grafer på resursanvändning och på antalet anslutningar till applikationerna.

I ”Live Log” (härefter applikationsloggen) i FMAC kan man välja för vilken applikation man vill se loggen och sedan följa med loggen i realtid. Förutom applikationsloggen skapar FMS en loggfil för varje applikation under sin installationskatalog.

Applikationsloggen var under utvecklingen ett av de viktigaste verktygen tillsammans med panelen för delade objekt. Genom att använda dessa två verktyg fick man all felsökningsinformation som behövdes för utvecklingen.

I Figur 1 visas applikationsloggen för en nyuppstartad instans av spelmotorn. Figuren visar en del av applikationsloggen då en användare har anslutit sig till servern och startat ett spel med tre datorspelare. Namnet framför ”>”-tecknet är namnet på den metod som har skrivit i loggen. Programmet heter ”alpha” och instansen heter ”_definst_”. Beteckningen ”_definst_” är grundinställningen för applikationsnamn i FMS.



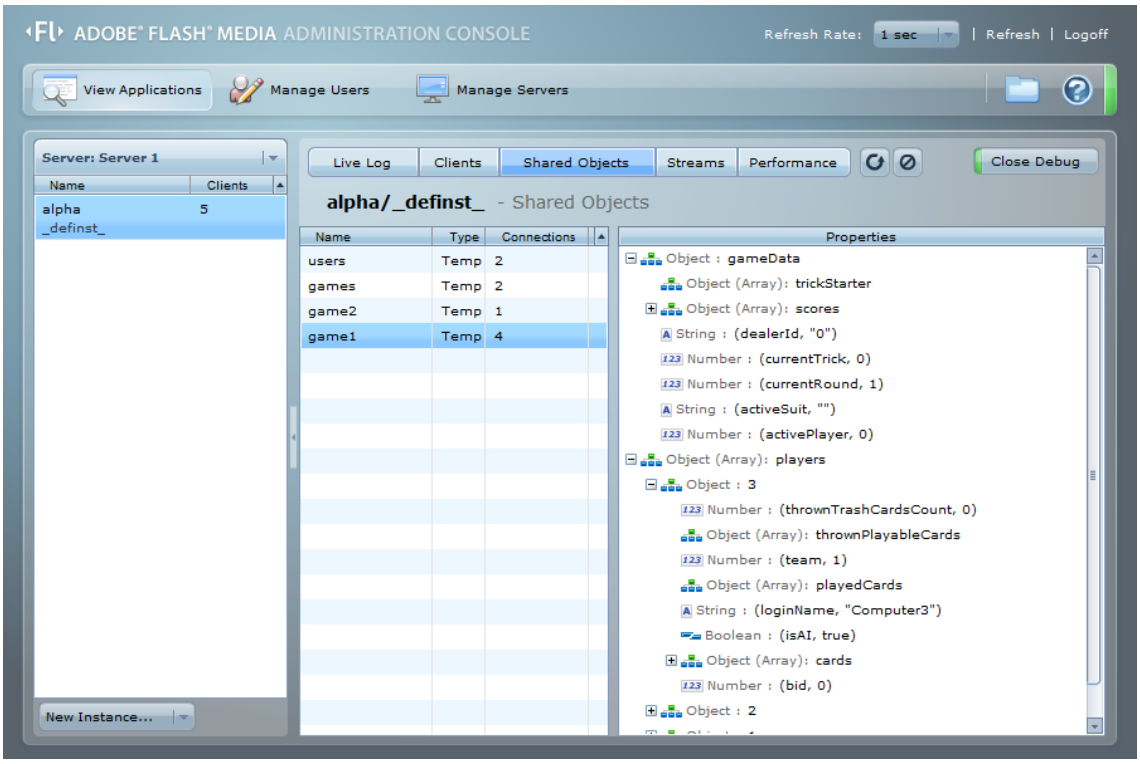
Figur 1 FMAC – Applikationsloggen för spelmotorn

Applikationsloggen för spelmotorn finns endast på engelska och det är endast serveradministratören som kommer åt den via FMAC.

Applikationsloggen uppdateras så gott som alttid då spelmotorn gör någonting.

Förutom loggen finns det en separat panel för delade objekt. I den kan man välja vilket delat objekt man vill titta på. När man valt ett delat objekt visas alla objekt och variabler som finns i det.

I Figur 2 visas ett delat objekt med namnet ”game1” och en del av det data som finns lagrat i det.



Figur 2 FMAC – Delade objekt

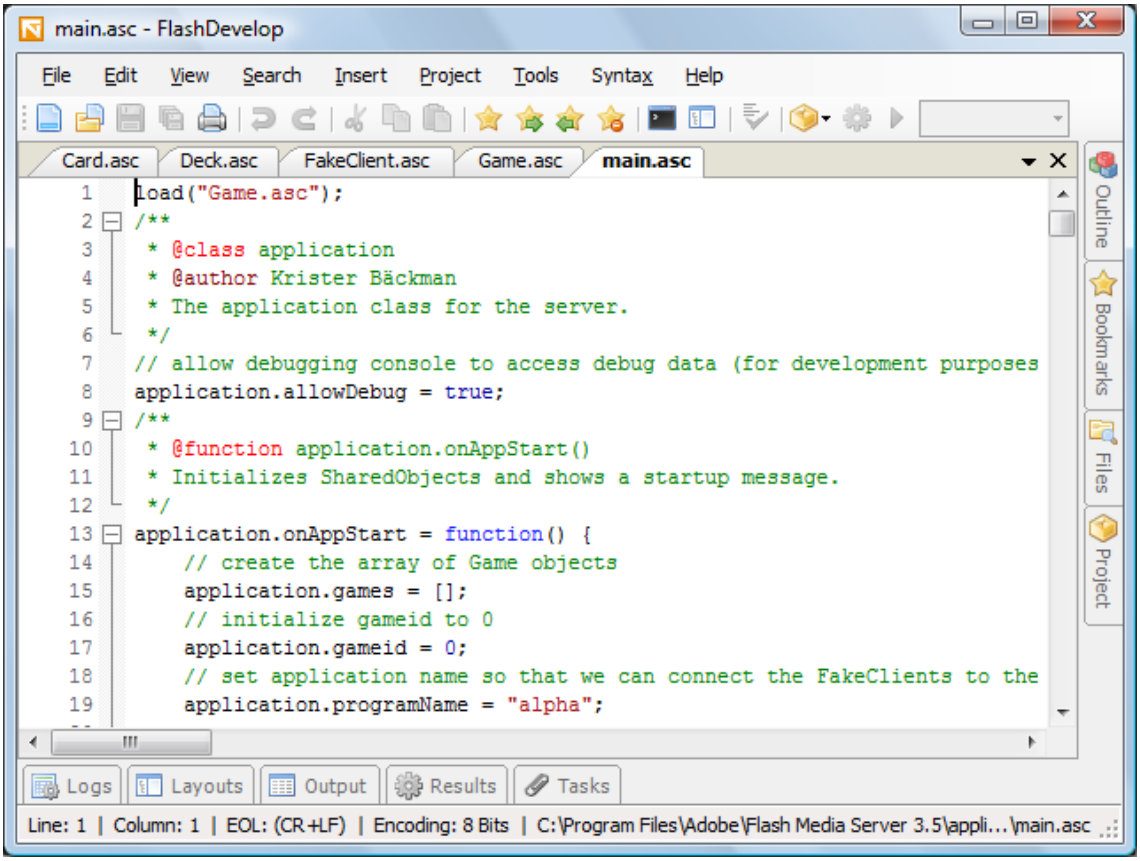
Delade objekten uppdateras också i realtid precis som applikationsloggen.

2.4 Programmeringsverktyg

För programmeringen av spelmotorn användes FlashDevelop. FlashDevelop är ett programmeringsverktyg med öppen källkod som är licenserat med MIT-licensen.

FlashDevelop valdes som programmeringsverktyg för spemotorn för att den erbjuder bl.a. syntaxfärgläggning för JavaScript. Den har också andra egenskaper som underlättar programmeringen. FlashDevelop användes som programmeringsverktyg även i klientdelen av projektet. Nyttan med att använda samma verktyg för både klient- och server-delen var att testklienter lätt kunde köras på båda utvecklingsmaskinerna. Då detta ytterligare kombinerades med Subversion för versionshantering kunde koden lätt uppdateras för hela projektet bara genom att använda ”svn commit” och ”svn update” kommandona.

I Figur 3 visas programmeringsgränssnittet i FlashDevelop med syntaxfärgläggning på koden. Koden är en del av Application-klassen i spelmotorn.



Figur 3 FlashDevelop – Programmeringsgränssnittet

För felspårande användes programvaran Mantis. Mantis är ett verktyg där man kan skapa felrapporter. Ifall det är frågan om en riktig bugg som går att återskapa korrigerar utvecklarna felet och stänger ärendet i Mantis.

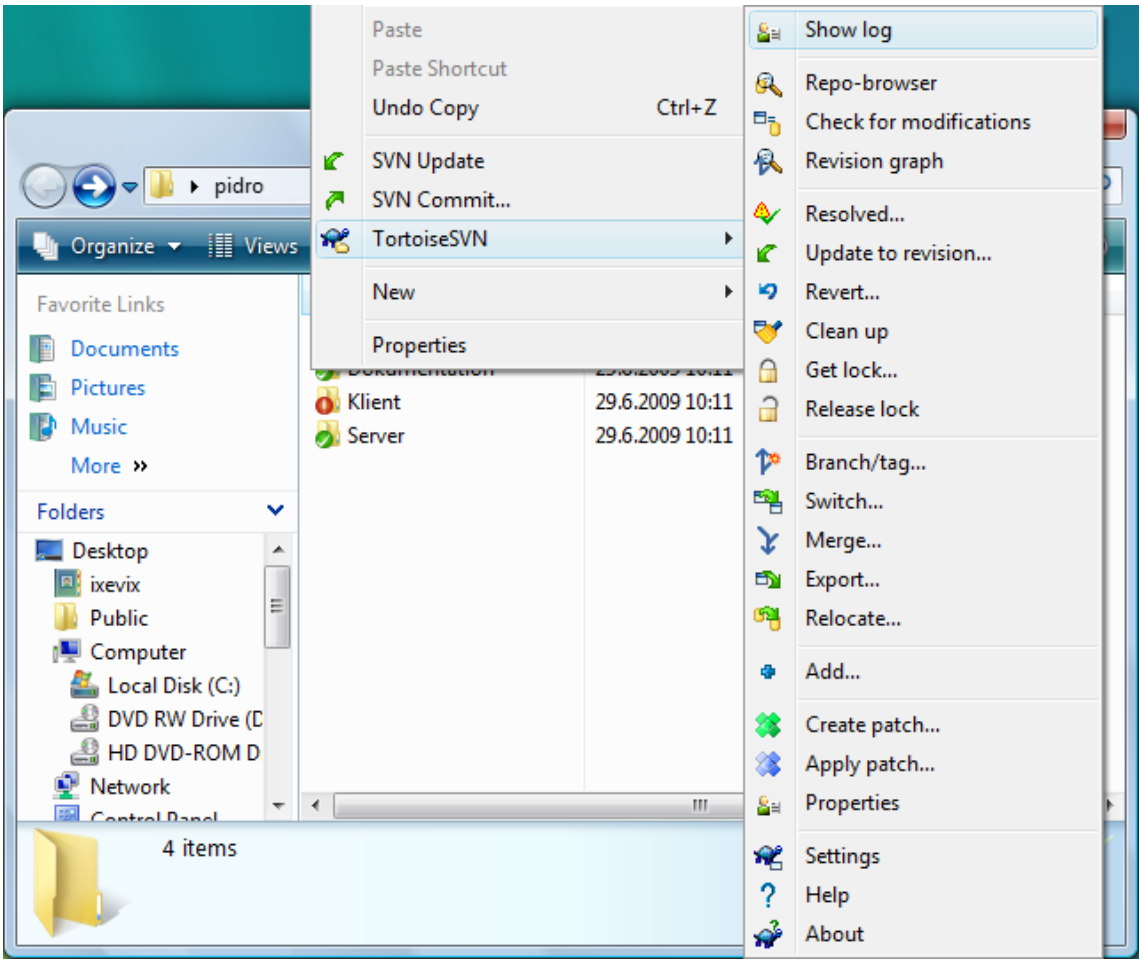
I Figur 4 visas Mantis webbgränssnitt. I gränssnittet kan man skapa en lista på alla ärenden som finns i systemet, skapa nya ärenden och redigera gamla ärenden.

Figur 4 Mantis – Webbgränssnittet

Mantis valdes som verktyg för att det var enkelt att ta i bruk och för att det underlättade arbetet för utvecklarna.

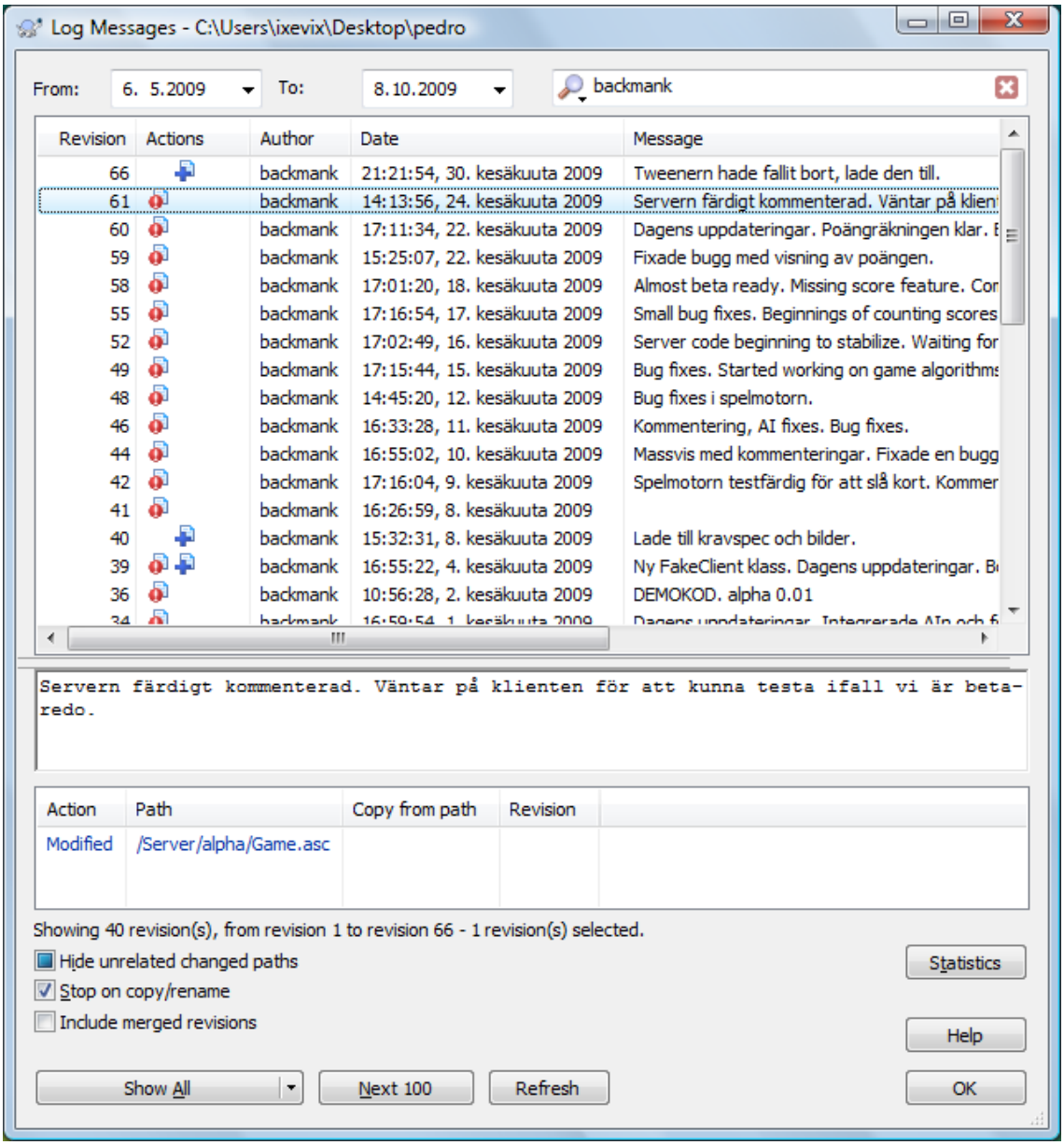
För versionshantering användes TortoiseSVN. TortoiseSVN bygger på versionshanteringssystemet Subversion.

I Figur 5 visas TortoiseSVNs användargränssnitt. Gränssnittet är integrerat i Windows Explorer och är lätt att använda då man hittar alla Subversion-funktioner under högerklickmenyn.



Figur 5 TortoiseSVN

I Figur 6 visas TortoiseSVNs loggfunktion. När man kör en ”commit” på koden lägger man till ett meddelande. Ifall man senare blir tvungen att gå tillbaka till en tidigare version av koden kan man lätt hitta rätt version via meddelandet.



Figur 6 TortoiseSVN - SVN log

I Pidro-projektet användes Subversion för att spara koden på ett säkert ställe och för att vid behov kunna gå tillbaka till en tidigare version. Om man t.ex. utvecklade någon ny funktion som krävde ändringar på flera ställen var det lätt att gå tillbaka till föregående version av koden m.h.a. Subversion.

3 SPELMOTORNS FÖRVERKLIGANDE

3.1 Spelets uppbyggnad

Pidro-spelet är uppbyggt så att det finns en server och flera klienter. Spelmotorn och datorspelarna körs på servern som en serverapplikation för FMS. Klienterna kan vara människospelare eller datorspelare. När en autentiserad människospelare kommer in på servern placeras den först i lobbyn. I lobbyn kan spelaren välja att starta ett nytt spel, gå med i ett spel som inte ännu har startat eller stänga av förbindelsen till servern. I lobbyn visas namnen på alla spelare som finns i lobbyn och de spel som inte ännu har startat.

Då man startar ett spel visas man inte längre i lobbyn och då man har spelat färdigt och kommer tillbaka till lobbyn visas man där igen.

Spelarantalet i ett spel är fyra och man kan inte starta ett spel med fler eller färre än fyra spelare. Man kan inte starta ett spel utan minst en människospelare. Det finns alltså fyra olika kombinationer på antalet människo- och datorspelare i ett spel. Tabell 1 visar dessa kombinationer.

Tabell 1 Antal spelare i ett spel

Människospelare	Datorspelare
1	3
2	2
3	1
4	0

Spelmotorn är uppbyggd så att den kan hantera flera samtidiga spel med olika uppsättningar av spelare.

En människospelare kan välja att starta ett spel när som helst då han befinner sig i lobbyn.

Det går att starta ett spel på två olika sätt. Man kan starta ett spel med sig själv och tre datorspelare eller så startar man ett spel med att vänta på att andra spelare skall ansluta

sig. Om man väljer att spela enbart mot datorspelare sätter spelet igång genast, annars väntar man på andra spelare.

Då man väljer att starta ett spel blir man utsedd till värd för spelet. Värdens namn sparas och andra spelare kan ansluta sig via värdens namn från lobbyn. Ifall värden lämnar spelet blir någon annan människospelare värd. Ifall det inte finns människospelare i spelet så stängs spelet.

Under tiden man väntar på andra spelare kan man välja sitt lag. I spelet finns alltid två lag, det röda och det blåa. Alla spelare kan välja lag och man kan också välja att vänta i ett läge där man inte hör till något lag. Ett spel kan inte starta förrän alla människospelare har valt ett lag.

När alla spelare har valt sitt lag kan värden starta spelet. När värden startar spelet sätts spelet igång och värden börjar med att dela ut korten. När värden har delat ut korten börjar den som sitter till vänster om värden med att ge bud. Sedan ger alla bud i tur och ordning.

Det finns sammanlagt fyra handlingar som en spelare behöver göra under en spelrunda: bjuda, välja sort, kasta kort och spela kort. För dessa handlingar finns metoder i spelmotorn. Spelmotorn meddelar klienten när den behöver göra en handling genom att anropa en metod på klienten. Klienten svarar genom att göra ett anrop tillbaka. Anropet motsvarar den handling som klienten vill göra.

Varje spel har en egen logg som visas för klienterna. Loggen uppdateras av spelmotorn. När det sker en händelse i ett spel skickar spelmotorn ut ett meddelande åt alla klienter i spelet.

En spelrunda är slut då ingen spelare har kort kvar på handen. När spelrundan är slut börjar en ny spelrunda. När något lag har 62 poäng eller mera slutar spelet. När spelet är slut frågas spelarna ifall de vill spela ett nytt spel med samma uppsättning av spelare. Ifall en spelare väljer att inte fortsätta skickas han tillbaka till lobbyn. För de resterande spelarna börjas ett nytt spel.

Man kan lämna spelet då spelet pågår. Om man lämnar spelet ersätts man av en datorspelare. Ifall alla människospelare lämnar ett spel stängs spelet.

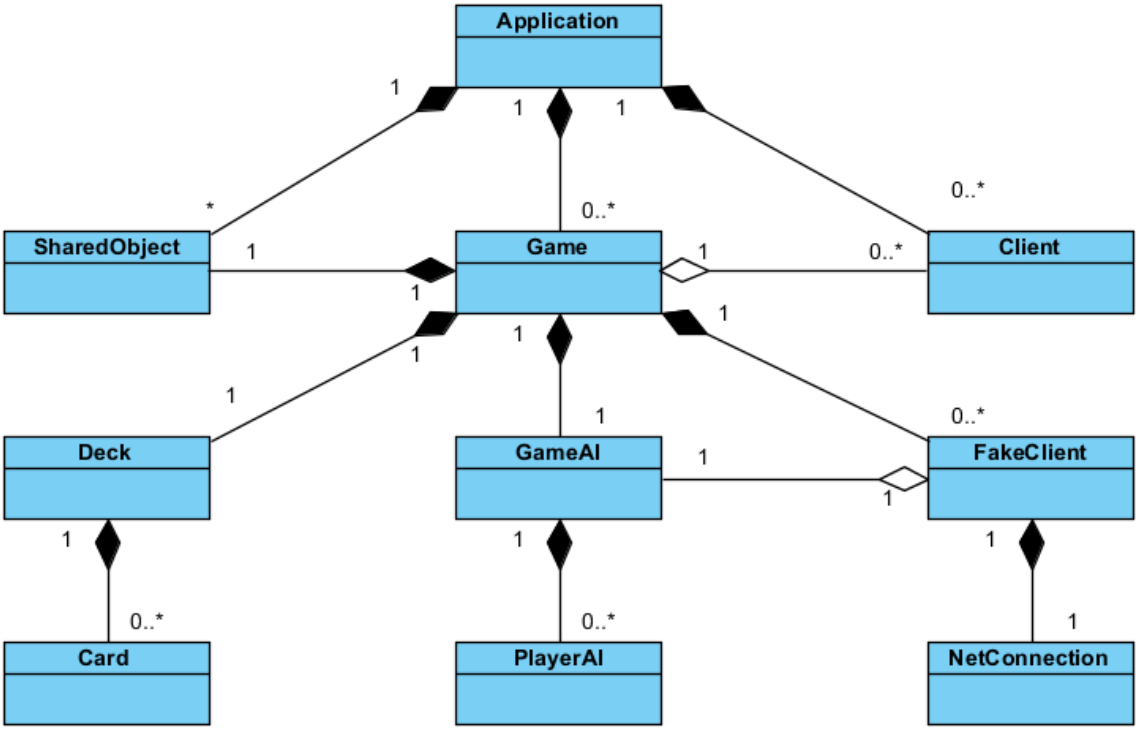
Om serverapplikationen kraschar eller stängs av försvinner all data om spelarna och spelen eftersom delade objekten inte är skapade beständiga.

3.2 Klasstrukturen

Spelmotorn använder sig av och utvidgar SSAS-klasserna: Application, SharedObject, Client, NetConnection och implementerar de egna klasserna: Game, Deck, Card och FakeClient. Spelmotorn använder sig också av GameAI-klassen, som är ett gränssnitt till AI-delen av projektet.

3.2.1 Klassdiagram

Figur 7 visar klassdiagrammet för spelmotorn. Klassdiagrammet visar hur de olika klasserna är relaterade till varandra genom att visa associationer och multiplicitet.



Figur 7 Klassdiagram

När spelmotorn startas skapar FMS en instans av Application-objektet. Application-objektet skapar instanser av SharedObject-, Game- och Client-klasserna. Game-klassen skapar instanser av Deck, SharedObject-, GameAI och FakeClients-klasserna. Game-klassen sparar Client- och FakeClient-objekt i en tabell för att komma åt spelarna i spelet. Deck-klassen skapar instanser av Card-klassen och sparar dem i en tabell. GameAI-klassen skapar instanser av PlayerAI-klassen. FakeClient-klassen skapar en instans av NetConnection-klassen. FakeClient-klassen känner till GameAI-klassen för att när den skapas får den med en referens till ett GameAI-objekt i konstruktorn.

3.2.2 Application-klassen

Application-klassen sköter om initialisering av variabler och delade objekt, förbindelser från klienter, initialisering av nya spel, uppdatering av informationen i lobbyn och informationsutbyte mellan klienter och Game-objekt.

Application-klassen är en inbyggd klass i SSAS som det bara finns en instans av så länge som ett program körs. Om flera instanser av ett program körs samtidigt på FMS

kan flera Application-objekt existera samtidigt. I klassen kan man programmera åtkomstkontroll och bestämma vad som skall göras med anslutningarna från klienterna.

När man startar vilket som helst program på FMS skapas en instans av Application-klassen och dens konstruktor ”onAppStart” körs. Konstruktorn i spelmotorn initialiserar delade objekten ”users” och ”games”, som innehåller informationen som visas i lobbyn. I ”users”-objektet sparas de inloggade användarnas användarnamn och i ”games”-objektet sparas information om spelsessioner som ännu väntar på mera spelare.

3.2.3 SharedObject-klassen

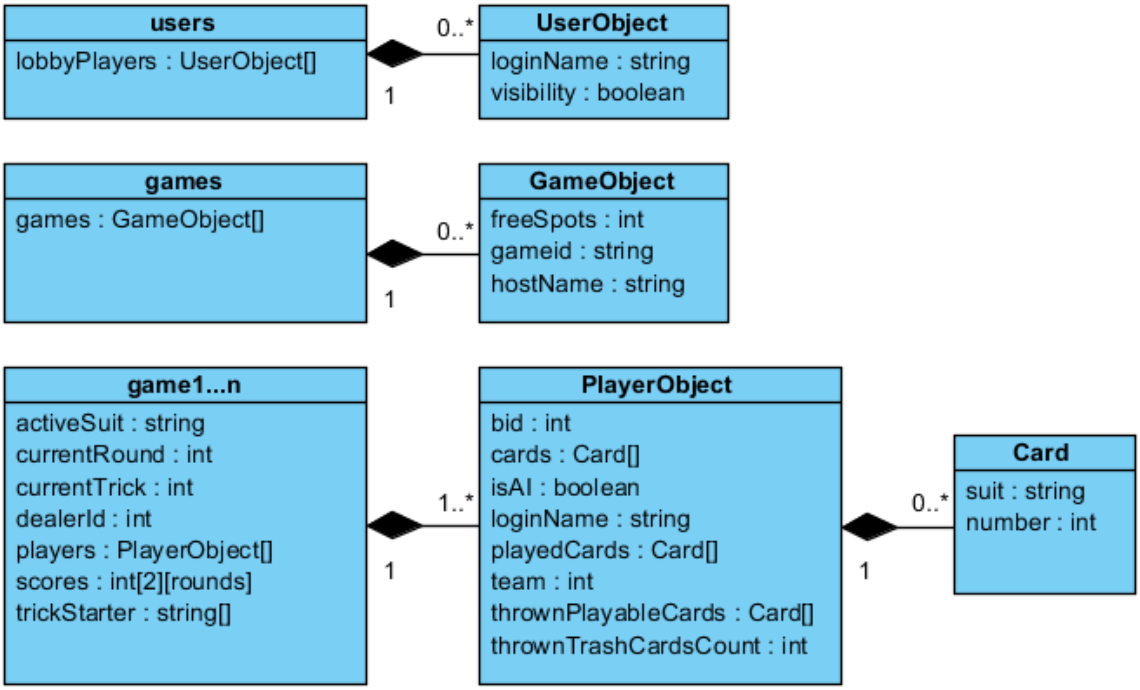
Delade objekt är ett sätt att spara data i SSAS. Delade objekt är speciella objekt som kan användas av flera klienter och servrar samtidigt.

Spelmotorn använder SharedObject-klassen för att spara inloggade användare och speldata. När variablerna på ett delat objekt ändras, uppdateras värdena för alla klienter som är uppkopplade till det.

Man kan också använda ”send”-metoden för att skicka data till klienter som lyssnar på det delade objektet. Metoden tar in ett metodnamn som den anropar med de övriga parametrarna. Denna metod används i spelmotorn för att förverkliga loggen som syns för klienterna.

I Flash kan delade objekt skapas av både klienter och servrar.

De delade objekt som används i spelmotorn är: ”users”, ”games” och ett delat objekt för varje spel som skapas (game1 ... gamen). I Figur 8 ser vi hur objekten är uppbyggda.



Figur 8 Delade objekt – uppbyggnad

”users”-objektet innehåller en tabell med användare. Användarna har ett namn och ett attribut som anger om de skall visas lobbyn.

”games”-objektet har en tabell med spel. Spelen har attribut för hur många lediga platser det finns, spelets id och vad värden för spelet heter.

Ett ”game1…n”-objekt sparar speldata och spelarobjekt. Speldata består av vilken sort som skall spelas, vilken spelrunda som spelas, vilket varv i spelrundan som spelas, vem som delar ut korten, vilka spelare som finns i spelet och vem som började varvet.

Spelarobjektet visar spelarens bud, kort, om spelaren är en datorspelare, spelarens namn, vilka kort spelaren har spelat i den aktuella rundan, vilket lag spelaren hör till, vilka spelbara kort spelaren har slängt och antalet ospelbara kort som spelaren har slängt.

Delade objekt kan skapas som beständiga eller icke-beständiga.

Beständiga delade objekt, sparas i underkatalogen
”sharedobjects/<applikationinstansens namn>” i serverapplikationens egen katalog

(t.ex. ”C:\Program Files\Adobe\Flash Media Server 3.5\applications\Test\sharedobjects_definst_”).

Icke-beständiga delade objekt i Flash försvinner då serverapplikationen stängs av eller kraschar. Beständiga delade objekt hålls kvar eftersom de kan läsas in på nytt från filen.

3.2.4 Client-klassen

Client-objekt innehåller information om klienten och dess anslutning till servern. Från Client-objektet kan man få fram bl.a. klientens version, operativsystem, och IP-adress.

För varje anslutning till en FMS skapas det ett Client-objekt.

3.2.5 NetConnection-klassen

NetConnection-klassen används av FakeClient-klassen för att skapa uppkopplingar till spelmotorn.

NetConnection-klassen i SSAS kan användas för att ansluta till både klienter och andra servrar. Man kan även välja om trafiken skall krypteras. Klassen har en ”call”-metod som kan anropas för att köra RPC-metoder. En klient kan kalla på metoden för att anropa en RPC-metod på en FMS. En FMS kan också programmeras för att anropa en RPC-metod på en klient. Metoden används i spelmotorn för att förverkliga logiken mellan klienterna och servern.

3.2.6 Game-klassen

Game-klassen är den största klassen i spelet. Den sköter om lagring av speldata, poängräkningen, spelets gång och initialiseringen av AI:n.

För varje Game-objekt som skapas, skapas ett delat objekt med namnet ”game” + ”gameid”. ”gameid” är ett positivt heltal som ökas varje gång ett nytt spel skapas.

Annan information som sparas i Game-klassen är värdens namn, spelets id-nummer, ett GameAI-objekt, en kortlek (Deck-objekt), diverse interna värden och Client-objekten för spelarna i spelet. Alla dessa värden initialiseras när det skapas en ny instans av Game-klassen.

3.2.7 Deck-klassen

Deck-klassen är en enkel klass för att skapa en kortlek med 52 kort, spara korten och dela ut korten. För att dela ut ett kort anropas ”deal”-metoden som returnerar översta kortet i packen.

Korten i Deck-klassen sparas i en array av Card-objekt.

3.2.8 Card-klassen

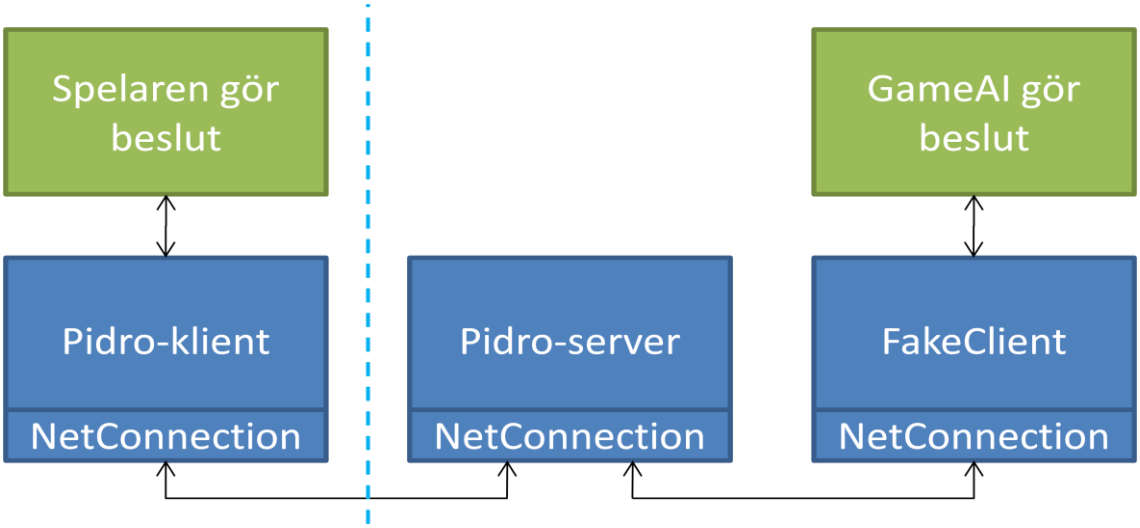
Card-klassen är också en enkel klass vars enda uppgift är att hålla värdena valör (eng. suit) och värde (eng. number) för ett kort.

3.2.9 FakeClient-klassen

FakeClient-klassen är en s.k. ”wrapper”-klass. Dess uppgift är att göra det enklare att programmera spelmotorn. Med hjälp av FakeClient klassen behöver programmet inte bry sig om den uppkopplade spelaren är en människo- eller en datorspelare. Det här förverkligas genom att varje gång en datorspelare skapas, skapas det också en instans av FakeClient-klassen. Instansen håller reda på vilket spel och vilken datorspelare som den hör ihop med. Instanserna sparas i det tillhörande Game-objektet tillsammans med Client-objekt som tillhör människospelare.

Instansen av FakeClient-klassen skapar en ny instans av NetConnection klassen och använder den för att skapa en ny förbindelse till servern (i detta fall till sig själv). Förbindelsen upptas av spelmotorn och ett Client-objekt skapas för datorspelaren precis som om den vore en människospelare.

Figur 9 visar informationsflödet mellan de olika delarna av projektet. Den streckade linjen visar gränsen mellan klient och server.



Figur 9 Informationsflöde mellan klient och server

För att både FakeClient- och Client-objekten sparas i samma tabell kan man bara iterera genom tabellen och anropa ”call”-metoden för alla objekt i tabellen. För Client-objekt anropas Client-klassens ”call”-metod och för FakeClient-objekt anropas FakeClient-klassens ”call”-metod.

När spelmotorn anropar Client- eller FakeClient-objektets ”call”-metod med olika parametrar behöver den inte veta om objektet är ett Client- eller FakeClient-objekt eftersom båda klasserna har en egen ”call”-metod.

Metoden i FakeClient-klassen kontrollerar vilket kommando som spelmotorn vill köra, frågar AI:n via GameAI-objektet vad den vill göra och skickar AI:ns svar vidare till spelmotorn via NetConnection-objektet.

3.2.10 GameAI-klassen

GameAI-klassen fungerar som ett gränssnitt mellan AI:n och spelmotorn. GameAI-klassen har fyra publika metoder som motsvarar de val (bjuda, välja sort, slänga kort och spela kort) som en spelare måste göra i ett spel. FakeClient-klassen har också motsvarande metodnamn och de anropar GameAI-klassens motsvarande metoder.

GameAI-klassen läser speldata från delade objekt och skickar det vidare åt de PlayerAI-objekt som den skapar.

3.3 Kommunikationsmodellen

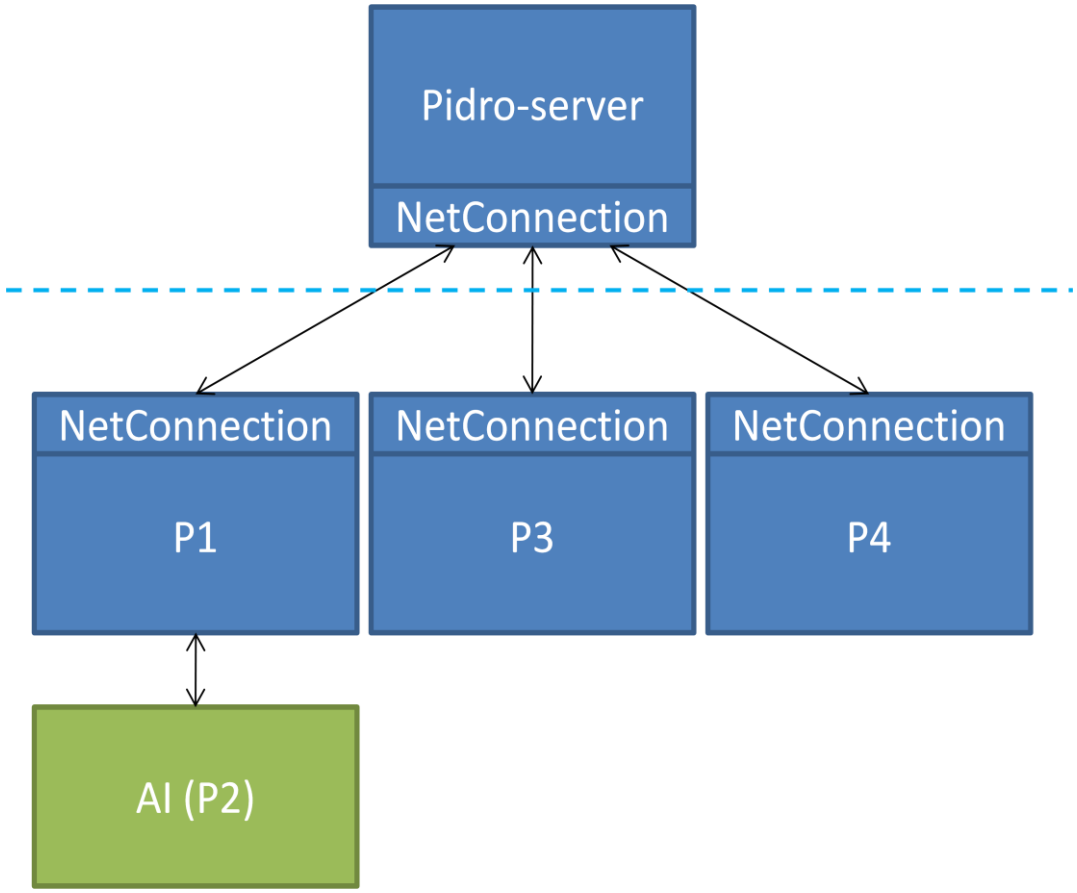
Ett av problemen som löstes under utvecklingen var placeringen av AI-spelaren. Skall den placeras på servern eller klienterna? Ett krav som ställdes från Rundradion var att ifall någon klient lämnar spelet då en spelsession är igång skall den ersättas med en AI-spelare.

Det var ett kritiskt designval ur projektets synvinkel att placera datorspelaren på servern. FMS programmeras i SSAS och klienter i Flash programmeras i CSAS (ActionScript 3). Valet måste göras så snabbt som möjligt så att utvecklarna visste vilket programmeringsspråk de skulle använda för utvecklingen av datorspelaren. När placeringen av datorspelaren blev klar kunde utvecklarna koncentrera sig på sina egna delar av projektet.

3.3.1 Alternativ 1: AI-spelaren placeras på klienten

Om vi placerar AI-spelaren på klienten uppstår problem ifall spelaren hoppar ut ur spelet eller blir bortkopplad från servern mitt i en spelsession. Om förbindelsen bryts får spelmotorn inte det data som AI-spelaren har sparat lokalt på klienten. Om spelmotorn ändå skulle spara en kopia av det data som AI-spelaren har på klienten uppstår redundans och onödig nätverkstrafik då data uppdateras i varje steg av spelet. Det borde också finnas en mekanism för att starta en ny AI-spelare på en annan klient och en mekanism för att överföra data till klienten som startar den nya AI-spelaren mitt i spelet. Fastän informationen skulle sparas i spelmotorn uppstår ändå problemet med att starta AI-spelaren på någon annan klient ifall nätverksförbindelsen avbryts.

Figur 10 visar kommunikationsmodellen då datorspelaren placeras på klienten. Den streckade linjen markerar gränsen mellan server och klient. ”Pn” står för spelare nummer n.



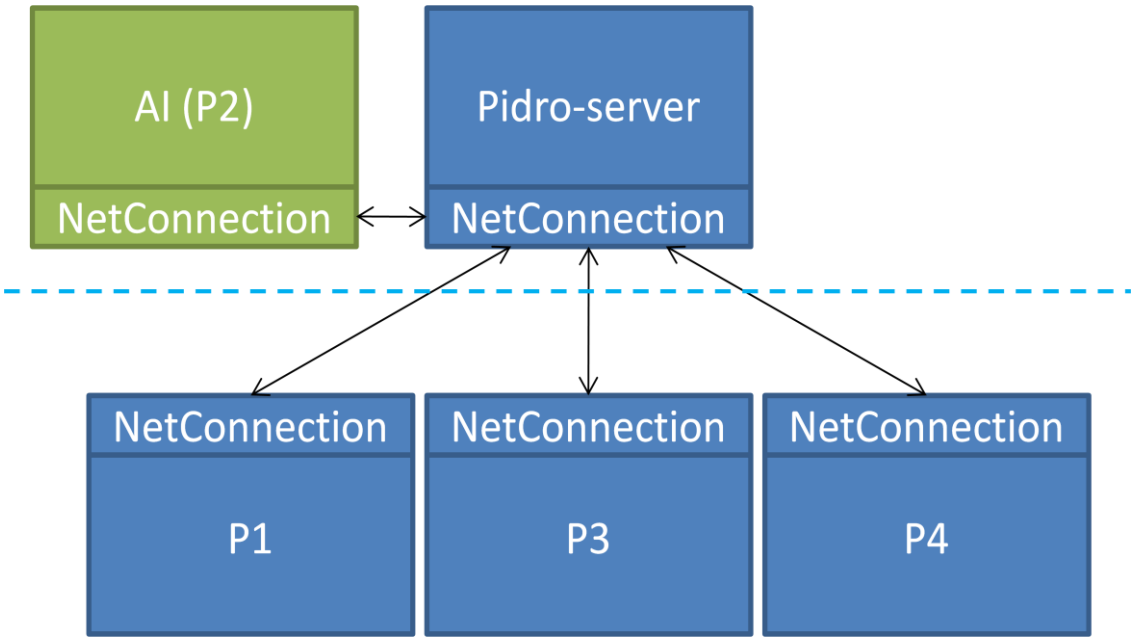
Figur 10 Alternativ 1 - datorspelaren placeras på klienten

3.3.2 Alternativ 2: AI-spelaren placeras på servern

Då AI-spelaren placeras på servern kan vi spara informationen den behöver där. Om en klient hoppar ut ur spelet påverkar det inte AI-spelaren. Detta var den lösningsmodell som har förvekligats. Fördelen är att vi inte behöver starta en ny AI-spelare om klienten med AI-spelaren blir bortkopplad från servern. När det behövs en AI-spelare för ett spel skapas det en instans av FakeClient-klassen. FakeClient-klassen öppnar en ny NetConnection till servern och använder den för att anropa RPC-metoder. FakeClient-klassen implementerar samma metoder (bid, chooseSuit, throwCards och playCard) som klienten. Då behöver spelmotorn inte bry sig om vilken sorts spelare som den ber att ge

bud (bid) eller att slänga kort (throwCards) utan den bara kallar på RPC-metoder som finns implementerade i Client- och FakeClient-klasserna.

Figur 11 visar kommunikationsmodellen då datorspelaren placeras på servern. Den streckade linjen markerar gränsen mellan server och klient. ”Pn” står för spelare nummer n.



Figur 11 Alternativ 2 - datorspelaren placeras på servern

3.4 Nätverkskommunikation

När en klient tar kontakt med spelmotorn körs Application-klassens ”onConnect”-metod. ”OnConnect”-metoden är inbyggd i SSAS och körs alltid när en klient försöker ansluta sig till FMS. Då metoden körs skapar SSAS ett Client-objekt som innehåller information om klientens uppkoppling. Genom att implementera metoden ”onConnect” i Application-klassen kan vi bestämma hur anslutningen skall hanteras. Klienten behöver endast skicka sitt användarnamn åt spelmotorn. Användarnamnet sparas i Client-objektet.

Om klientens användarnamn är ”Computer1”, ”Computer2”, ”Computer3” eller ”Computer4” kontrollerar spelmotorn om anslutningen kommer från den lokala datorn.

Om inte, stänger spelmotorn av anslutningen eftersom det då inte är frågan om en datorspelare.

Om klienten inte är en datorspelare kontrollerar spelmotorn om spelaren har ett giltigt användarnamn och ifall spelaren redan finns inloggad. Då spelarens anslutning har godkänts skickas spelaren till lobbyn där han kan göra sina första val.

Till skillnad från klienterna kommer AI-spelarnas anslutning alltid från den lokala datorn för att AI-spelarnas anslutning skapas av FakeClient-klassen i spelmotorn. Detta ger en möjlighet för vidareutveckling då datorspelarna kan köras helt och hållet på en annan server. Då kan man kolla ifall anslutningen kommer från den andra serverns IP-adress. Genom att köra datorspelarna på en annan server kan man frigöra resurser på servern som spelmotorn körs på. Detta kan göras ifall det kommer fram i Rundradions testning att spelmotorn inte klarar av att sköta nätverkskommunikationen med klienterna och köra flera datorspelare på samma gång.

3.5 Datasäkerhetsaspekter

3.5.1 Krypterade nätverksprotokoll

FMS erbjuder stöd för två olika krypterade nätverksprotokoll, RTMPE (Encrypted RTMP) och RTMPS (SSL).

RTMPE har 128-bits kryptering men stöder inte certifikat. RTMPE är uppbyggt för prestanda och för tjänster som kan använda andra metoder för autentisering.

När man använder RTMPE måste klientapplikationen vara Adobe AIR eller nyare än version ”9 update 3” av Flash Player.

RTMPS erbjuder bättre krypteringsalgoritmer men kräver mera prestanda än RTMPE. RTMPS låter programmeraren välja vilken krypteringsmetod han vill använda. RPTMS har stöd för SSL certifikat.

Grundinställningen i FMS är att RTMPE tillåts men för att ta i bruk RTMPS måste man konfigurera servern för SSL. När man vill göra en anslutning med RTPME eller RTPMS måste man specificera ”rtmpe://<adress>” eller ”rtmps://<adress>” som parameter när man gör anslutningar via ett NetConnection-objekt. (Adobe 2009a) (Adobe 2010c)

Ingendera av de ovannämnda protokollen är i bruk i den prototyp som utvecklades för Rundradion.

3.5.2 Verifiering och autentisering av klienter

FMS stöder olika sorters verifiering av klienter, verifiering av Flash-filer, åtkomst för klienter från specifika domäner och kontroll av den ursprungliga länken.
Man kan konfigurera en FMS att endast acceptera anslutningar från Flash-filer som man själv har skapat. Detta förhindrar tredje parter att ansluta sig till servern med en modifierad Flash-fil.

Verifieringen av Flash-filer kan tas i bruk i spelmotorn för att förhindra fusk.
Med den andra typen av verifiering kan man tillåta eller blockera klienter som kommer från specifika domän. (Adobe 2009a)

För att kontrollera den ursprungliga länken som klienten har använt för att ansluta sig till en FMS används Client-objektets ”referer”-attribut.

Exempel:

```
referrerList = {};
referrerList["http://www.example.com"] = true;
referrerList["http://www.abc.com"] = true;

if (!referrerList[client.referrer]) {
  application.rejectConnection(client, {"Access Denied"} );
}
```

Exemplet kunde användas för att endast tillåta anslutningar som kommit via ”http://www.yle.fi”.

3.5.3 Åtkomstkontroll för delade objekt

Åtkomstkontroll för klienter som ansluter sig till en FMS implementeras genom attribut på Client-objektet. Client-objektet har ”readAccess” och ”writeAccess” attribut som kontrollerar vilka kataloger klienten har läs- och skrivrättigheter till. Grundinställningen är ”/” och den tillåter läs- och skrivrättigheter till alla resurser som finns på servern. Attributen är av typen sträng och kan innehålla flera sökvägar om man separerar dem med ett semikolon.

Andra resurser som kan finnas på FMS är t.ex. video- eller datafiler.

För att kunna separat specificera till vilka delade objekt klienten har rättigheter till måste man skapa skilda kataloger för de delade objekten eftersom man inte kan ändra rättigheterna för enskilda filer utan bara för hela kataloger. (Adobe 2009b)

Åtkomstkontroll för delade objekt är inte implementerat i spelmotorns prototyp men kunde implementeras för ökat skydd mot fusk.

3.5.4 Sårbarheter i Flash Media Server

Den 30 april 2009 just under den mest intensiva utvecklingsperioden publicerade Adobe en sårbarhet i FMS 3.5.1 och tidigare versioner. Sårbarheten tillåter fjärrangripare att köra RPC-metoder på servern utan att autentisera sig. Adobe lappade RPC-sårbarheten i FMS 3.5.2 och rekommenderade alla att uppgradera FMS till 3.5.2. (Adobe 2009c) (NVD 2009a)

Den 21 december 2009 publicerade Adobe två kritiska sårbarheter i FMS 3.5.2 och tidigare versioner och släppte ut FMS 3.5.3 som åtgärdade sårbarheterna. (Adobe 2009d) (CERT-FI 2009)

Den första sårbarheten tillät att en DoS-attack kunde utföras på FMS genom att använda upp alla serverns resurser. (NVD 2009b)

Den andra sårbarheten lät angripare ladda godtyckliga DLL-filer via ospecificerade vektorer. (NVD 2009c)

4 DISKUSSION

Det här examensarbetet visar att Flash lämpar sig för spelutveckling. Flash erbjuder en god bas för spelmotorn då det går att använda SSAS-klasserna för att implementera nätverkskommunikationen och lagringen av data i delade objekt. Det kvarstår dock att jämföra Flash med andra utvecklingsmiljöer och hur de lämpar sig för spelutveckling.

Spelmotorns design är enkel, effektiv och använder sig av lösningar som underlättar programmeringen och vidareutvecklingen den. Spelmotorn sköter om spellogiken och att klienterna får den information de behöver visa. Spelmotorn tar emot klienternas val och ser till att spelet fortsätter. Spelmotorn körs på FMS.

Spelmotorn planerades och skrevs om från början två gånger under utvecklingen. Första gången den skrevs om var för att vi inte kände till att man kan spara objekt i delade objekt. När vi upptäckte detta skrevs spelmotorn om så att den sparade objekt i delade objekten. Tidigare sparades data i tabeller. För att vi nu kunde använda objekt blev utvecklingen snabbare. Vi behövde inte längre indexera en tabell utan kunde direkt komma åt medlemsvariablerna via deras namn. Detta visade sig vara en bra sak eftersom utvecklingen blev enklare och snabbare efter upptäckten.

Valet att ta i bruk FakeClient-klassen gjordes för att undvika onödiga kontroller om spelaren är en människo- eller datorspelare. FakeClient-klassen kom med sent i utvecklingen då spellogiken programmerades. FakeClient-klassen var orsaken för att spelmotorn måste skrivas om för andra gången. Detta berodde på att Client-objekt inte tidigare sparades i Game-objektet.

Examensarbetet behandlar utvecklingsverktygen och ger orsaker varför verktygen har använts. Man får en bild av vilka verktyg som lämpar sig för denna sorts projekt. En jämförelse mellan kommersiella produkter och gratis verktyg kunde göras för att se hur de skiljer sig.

Examensarbetet ger förslag för vilka metoder som kan användas i vidareutvecklingen spelmotorn.

För att motverka fusk i spelet kunde man sätta på verifieringen av Flash-filer. Det skulle förhindra att någon tar kontakt till servern med en modifierad klient och försöker få en fördel i spelsituationer. Kryptering kunde appliceras på nätverkskommunikationen för att förhindra någon att avlyssna trafiken och således få fram t.ex. vilka kort som en spelare har på handen. Krypteringen kan däremot påverka prestandan negativt. Eftersom spelet skall spelas på Internet och främst av finlandssvenskar kan det vara en bra idé att endast tillåta anslutningar från inhemska IP-adresser eller anslutningar som kommer via Rundradions egna sidor. Genom att endast tillåta inhemska IP-adresser försvåras en DDoS-attack. För att göra spelinformationen riktigt säker kan programmera åtkomstkontroll för delade objekten.

Som vi lärt oss har även FMS haft säkerhetsproblem men vilken mjukvara har inte? Fastän FMS har haft flera kritiska sårbarheter under de senaste åren tror jag att bara Adobe fortsätter att åtgärda dem kommer FMS att vara lika säkert som andra motsvarande produkter. För att hålla Pidro-spelet säkert kräver det dock av serveradministratören att hålla programvaran aktuell och uppdatera den när nya sårbarheter hittas.

För att autentisera användare kan Rundradion tillåta endast användare som är inloggade på X3M eller Vega att komma åt spelet. Användarnamnet kan tas från en sessionsvariabel och skickas till spelmotorn då användaren öppnar Flashen.

KÄLLOR

Adobe 2009a, Adobe Flash Media Server 3.5 Technical Overview [www]. Hämtat 7.9.2009.
http://help.adobe.com/en_US/FlashMediaServer/3.5_TechOverview/flashmediaserver_3.5_tech_overview.pdf

Adobe 2009b, Server-Side ActionScript Language Reference for Adobe Flash Media Interactive Server 3.5 [www]. Hämtat 7.9.2009.
http://help.adobe.com/en_US/FlashMediaServer/3.5_SS_ASD/flashmediaserver_3.5_ssl r.pdf

Adobe 2009c, Adobe - Security Advisories : APSB09-05 - Updates available to address Flash Media Server privilege escalation issue [www]. Hämtat 7.4.2010. Senast modifierat 30.4.2009.
http://www.adobe.com/support/security/bulletins/apsb09-05.html

Adobe 2009d, Adobe - Security Bulletin APSB09-18 - Security update available for Flash Media Server [www]. Hämtat 7.4.2010. Senast modifierat 28.1.2010.
http://www.adobe.com/support/security/bulletins/apsb09-18.html

Adobe 2010a, Adobe Flash Media Interactive Server 3.5: System requirements and languages [www]. Hämtat 18.1.2010.
http://www.adobe.com/products/flashmediainteractive/systemreqs/

Adobe 2010b, Adobe Flash Media Server End User License Agreement [www]. Hämtat 18.1.2010. Senast modifierat 12.8.2008, 23.00.
http://www.adobe.com/products/eulas/pdfs/fms3_5_eula.pdf

Adobe 2010c, Adobe Flash Media Flash Media Server 3.5 Configuration and Administration Guide [www]. Hämtat 6.4.2010. Senast modifierat 3.2.2010.

http://help.adobe.com/en_US/FlashMediaServer/3.5_AdminGuide/flashmediaserver_3.5_config_admin.pdf

Adobe 2010d, Adobe Flash Media Server 3.5 Developer Guide [www]. Hämtat 6.4.2010. Senast modifierat 3.2.2010. http://help.adobe.com/en_US/FlashMediaServer/3.5_Deving/flashmediaserver_3.5_dev_guide.pdf

CERT-FI 2009, CERT-FI - Haavoittuvuuksia Adobe Flash Media Server-ohjelmistossa [www]. Hämtat 7.4.2010. Senast modifierat 23.12.2009. http://www.cert.fi/haavoittuvuudet/2009/haavoittuvuus-2009-130.html

NVD 2009a, National Vulnerability Database (CVE-2009-1365) [www]. Hämtat 7.4.2010. Senast modifierat 19.5.2009. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-1365

NVD 2009b, National Vulnerability Database (NVD) National Vulnerability Database (CVE-2009-3791) [www]. Hämtat 7.4.2010. Senast modifierat 22.12.2009. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-3791

NVD 2009c, National Vulnerability Database (NVD) National Vulnerability Database (CVE-2009-3792) [www]. Hämtat 7.4.2010. Senast modifierat 22.12.2009. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-3792

Prayaga, Lakshmi; Suri, Hamsa. 2007. Beginning Game Programming with Flash. Course Technology. 266 s. ISBN: 9781598633986, 9781598636529

Rundradion 2009a, YLEs organisation [www]. Hämtat 19.8.2009. http://www.yle.fi/yleista/sve/organisation.shtml

Rundradion 2009b, YLEs grundläggande värden [www]. Hämtat 19.8.2009. http://www.yle.fi/yleista/sve/varden.shtml

Rundradion 2009c, Svenska YLE [www]. Hämtat 19.8.2009.
http://svenska.yle.fi/svenskayle.php

Wikipedia 2010a, Adobe Flash Media Server – Wikipedia, the free encyclopedia [www]. Hämtat 18.1.2010. Senast modifierat 3.1.2010, 10:38.
http://en.wikipedia.org/wiki/Flash_Media_Server

Topic, Michael. 2002. Streaming Media Demystified. McGraw-Hill Professional Publishing. 546 s. ISBN: 9780071388771, 9780071409629

Zakas, Nicholas C. 2005, Professional JavaScript for Web Developers, John Wiley & Sons Incorporated. 675 s. ISBN 9780764579080, 9780764597633

BILAGA 1: KRAVSPECIFIKATION

Pidro projekt för YLE (06.02.2009)

Regler och begrepp

- 4 spelare (2 mot 2)
- Spelvärd (host) den spelare/användare som startar ett nytt spel
- Datorspelare (AIn) heter DatorN (N=1,2,3)
- Färg eller sort är benämningen på den sort man spelar i, t.ex. hjärter, spader osv.
- Pidro är båda femmorna i den färg som spelas (Hjärter och ruter fem, alternativt spader och klöver 5).
- Poängkort är Ässet = 1p, Knekten = 1p, Tian = 1p, Tvåan = 1p, 2 st Pidro = 5p vardera i vald sort. Totalt alltså 14p.
- Max 14 poäng per omgång
- Spelbara kort = 14 kort (vald sort 13 st + samma färgs Pidro 1 st)
- Icke-spelbara kort = Alla förutom spelbara kort
- Varv eller stick är en del av en omgång när varje spelare sätter ett kort vardera i samma färg (om man inte är utan kort i färgen som spelas).
- Vinnare av hela spelet blir det lag som först erhållit eller överskridit 62 poäng. Om båda lagen går över 62 poäng under samma omgång vinner det lag som bjöd
- budgivning:
  om tre första buden är pass, bjuder kortgivaren minst 6
  om någon bjuder 14, vinner den budgivningen
- med 9 kort på hand:
  kastas 3 - 9 kort
  man får inte kasta spelbara kort, förutom om man har fått mera än 6 st av dem
  man får aldrig kasta poängkort
  man får behålla icke-spelbara kort
- sista spelaren (den som är tredje i turen) blir utan kort, ifall korten inte räcker till
- bortkastade och spelade ”spelbara” kort skall synas för alla spelare

Allmänt

- Utvecklingsomgivning: FlashDevelop, Flash Media Development Server
- Utvecklingsomgivningen erbjuder maximalt 10 anslutningar till utvecklingsservern. Detta betyder maximalt på 2 simultana spel under utvecklingsskedet
- "load balance" testning kan inte göras i utvecklingsskedet
- Man måste vara inloggad på X3M's community för att kunna spela
- Man kan endast spela i ett pågående spel per loginnamn
- Användarnamn fås från cookie (exempel fås från YLE)
- Man kan inte hoppa in i ett spel som redan har börjat
- Man kan inte följa med andras spel (spectator)
- Om man blir disconnected under ett spel ersätts man av en datorspelare
- Spelaren meddelas med ljud då spelet kräver uppmärksamhet
- Resolutionen på flashen får vara max 780x440 (439) (relation 16:9)
- Inga hårdkodade texter
- Alla drag som sker efter att spelet har satt igång är tidsbegränsade
-

Kravspecifikation för spelmotorn (Krister Bäckman)

Lobbyn

player connects to the game

new player object is created

present GUI **(draw GUI)**

GUI consists of buttons for new game with 3 AIs, new game, list of games that have not started, help, and status (how many games are currently active, how many games have been played)

press button to start game with 3 AIs → start new game → jump to "spelets gång"

press new game → you start a new game and are shown the "new game screen"

press a game in the list → you are taken to the "new game screen" of that game

press help → show the game rules

Help screen

Show the rules and a back button **(if the rules don't fit on one page include buttons for viewing multiple pages)**

press the back button → you are taken back to the lobby

“New game screen”

In the new game screen there are four buttons corresponding to the seats in the table and a leave button. The host also has a start game button. The game gets the name of the host, the host name is also visible in the lobby

as long as there are seats free the game shows up in the lobby new game list

when you enter the “New game screen” you are automatically placed in a seat

the host can start the game with any ratio of human / AI players

press an empty seat button → you switch seats

press the leave button → you are taken back to the lobby

host presses the start button → jump to “spelets gång”

if there are free seats when the host presses start the seats are filled by AI players

Spelets gång

Initialization

set up game log **(draw log)**

set up players (AI / human) **(draw player icons)**

create deck **(draw deck)**

shuffle deck

deal from deck 9 cards per player, direction: left of dealer **(draws cards)**

dealer rotates left after every round **(draw dealer mark)**

the host is the first dealer

Betting

player left of dealer starts

everyone takes turns rotating around the table **(needs AI decision) (draw gui for selecting bet and draw everyone's bet)**

the cycle ends at the dealer

minimum bet is 6

max bet is 14

you have to bet higher or pass

if everyone else passes dealer bets minimum 6

the one with the highest bet chooses sort to play **(draw selected sort)**

players can now select which cards to throw away or keep **(needs AI decision) (draw**

cards and make selection possible)

players can have max 6 cards in their hand

players then get cards from the deck until they have 6

if the cards run out they play with less than 6 cards

dealer gets to choose from all the leftover cards

if you have more than 6 playable cards you have to “kill” overflow cards

overflow cards are shown on the table

you cannot throw playable cards other than when you have more than 6 of them

you cannot throw point cards

Playing the game

Round 1

the highest bidder selects a card to play **(needs AI decision) (draw cards for human players)**

players take turns selecting cards to play **(needs AI decision) (draw cards for human players)**

if a player runs out of cards to play he throws them on the table **(draw cards)**

points are shown all the time **(draw points)**

the highest card wins the round 2,3,4,...J,Q,K,A

when round is over bet is checked against points and points are given to the winning player pair **(draw scoreboard)**

players play consecutive rounds until 62 points

at 62 points the game ends

players are asked if they want to play again and if they all say yes a new game is created for them

Game over

clients are disconnected and directed back to the lobby (player object is reset)

AIs are freed from memory (AI destructor cleans up)

results are saved for statistics

Pidro på X3M commnunityn

- Skall finnas en API som möjliggör att man kan hämta information om status
- Lyfter info om hur många som finns i lobbyn
- Visar hur många spel som är igång

Pidro Datorspelare (AI) – Kravspecifikation (Peter Saloviin)

Följande utförs av datorspelaren:

- ger bud
- väljer sort, ifall högsta bud
- slänger kort
- under varv/stick: sätter ut kort

Den får inte veta de andra spelarnas kort, även om dess par är också AI. AI:n diskuterar inte sinsemellan för att förenkla deras spelande. AI:n tar inte i beaktan vem den spelar emot (användarnamn eller AI). AI:n väntar inte med sina beslut.

Det görs intervjuer med erfarna Pidro-spelare för att använda deras kunskap om spelet vid utveckling av spelstrategin. Litteratur om spelteori för Pidro, eller kortspel med liknande spelproblem, undersöks också.

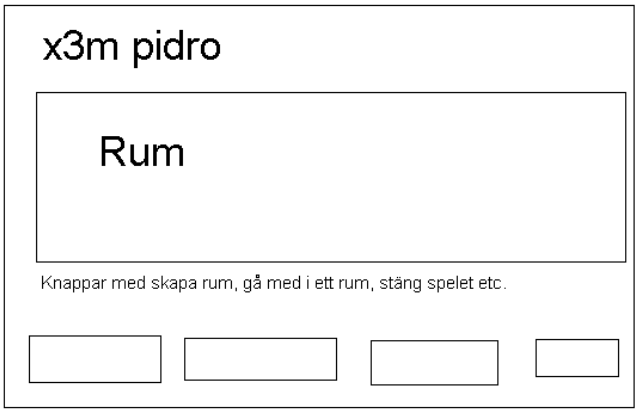
Pidro Spelplan (Ron Holmström)

Klient delen

- Grafiken levereras från Yle inom två veckor (20.2.09)
- Hurudan layout skall det vara?
- → Uppifrån, som normala hearts spelet?
- → Hurdana animationer skall det finnas?
- → Var skall poängräckningen finnas?
- → Kan man använda spelbordet som bakgrund för lobbyn, med en genomskinlig svart layer ovanpå?

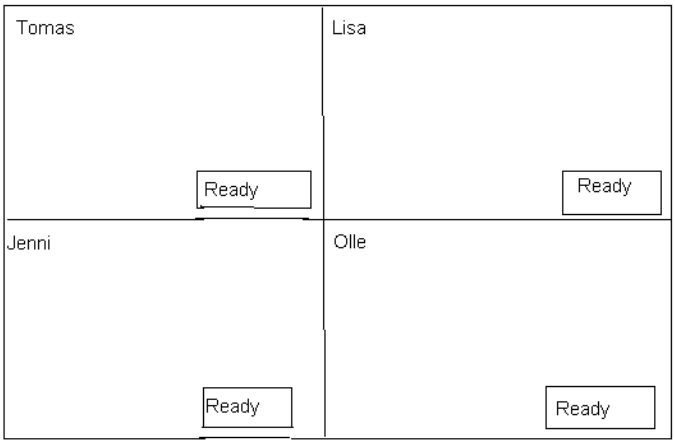
Lobby,

med rubrik, rummen start, join, help och exit knappar

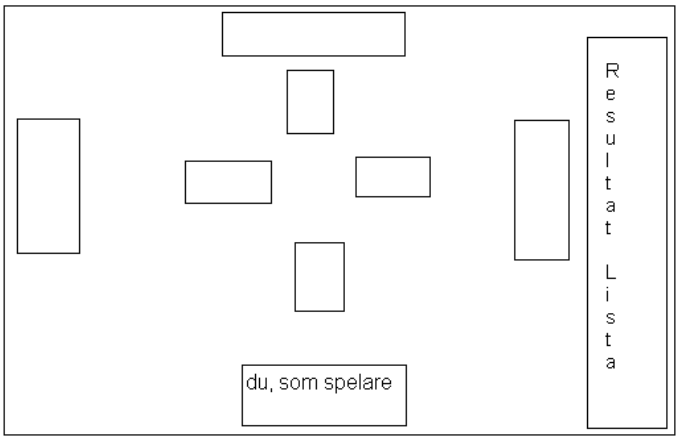


”Väntrum”,

med ready knappar alla får sin egen ”ruta”



Spel vy



- Skall man lyfta korten eller bara klicka?
- Ljud/musik för kort utdelning, varnings ljud för din tur att ge kort, bakgrunds musik? Levereras från YLE 20.2.2009

Vidareutveckling

- Individuell statistik
- Lösenord till spel
- Olika AIn
- Spara spelet
- Spela utan inloggning
- Chatt
- “Load balance”-testning