Chalmers Tekniska Högskola
TDA367 - Objektorienterat Programmeringsprojekt

Marcus Pettersson
Martin Sixtensson
Gustav Svensson
Erik Sänne

Zombienado

Ett nätverksspel för laganda och gemenskap

Inledning

1.1 Bakgrund

Sen de första kommersiellt tillgängliga privatdatorerna såg dagens ljus har digitala spel varit närvarande i samhället. Idag ses det som en av de största underhållnings-formerna, om inte den största, i stora delar av världen.

Enligt en ny studie av asisstant-professor, John Velez, från Texas Tech University så kan kooperativa videospel ha positiva sociala effekter (Watson, 2015).

1.2 Syfte

Projektets syfte är att undersöka hur nätverk fungerar och implementera det i ett *"zombie shooter"*-spel. Utifrån undersökningen ska ett kooperativ flerspelar spel skapas som går att spela över ett nätverk med flera olika datorer.

1.3 Struktur

Rapportens första del tillägnas till att beskriva projektets bakgrund och syfte. I nästföljande del, teori, beskrivs de tekniker och begrepp som används i projektet. Under metod beskrivs projektetsgång och hur projektet utfördes. Tillsist presenteras projektetsresultatet tillsammans med en diskussion med projektets deltagare om resultatet.

2. Teori

2.1 Domändriven design

Domändriven design innebär att mjukvaran modelleras genom ett samarbete mellan utvecklare och domänexperter med språket riktat mot domänet (Evans, 2004). Då Zombienados slutanvändare ses vara ungdomar som redan besitter viss erfarenhet av datorspel, mycket likt utvecklarna bakom spelet, så riktas domänmodellen till just denna grupp.
I modellen beskrivs i stora drag hur mjukvaran skall vara uppbyggd, mer eller mindre utan några som helst tekniska termer. Domänspecifika termer som är vanligt förekommande i spel används för att tydliggöra specifikationen så mycket som möjligt och underlätta utvecklingsprocessen.

2.2 Arkitektur

2.2.1 Model-View-Controller

Model-View,Controller, eller MVC som det förkortas till är en systemarkitektur som bygger på att man ska separera logik och grafik. Logiken hamnar i model medan grafiken hamnar i view. Mellan dem ligger controller som tar hand om användarinput och ser till att model och view ändrar sig på rätt sätt beroende på inputen.

Fördelen med MVC är att man kan byta ut till exempel en view mot en annan utan att det påverkar modellen och tvärt om.

2.2.2 Server-Client

För att kunna kommunicera mellan datorer används antingen “User Datagram Protocol” (UDP) eller “Transmission Controll Protocol” (TCP). UDP är ett mindre kontrollerat protokoll som inte bryr sig om ifall paketet kommer fram eller inte. Detta gör att UDP ger mer “packet-loss” fast är mer prestandaeffektivt. TCP kollar upp att mottagaren verkligen fick paketet och skickar igen om det behövs. Detta gör att TCP tar mer bandbredd.

I ett spel så gör det inte så mycket om man tappar ett paket ibland eftersom det uppdateras så ofta och bandbredd är oftast viktigare. Därför använder sig projektet av UDP istället för TCP (Rouse, 2015).

Server-Klient är en arkitektur eller ett designmönster där

I detta sammanhang kommer Server-Klient arkitekturen användas för att kunna kommunicera mellan olika datorer över ett nätverk.

2.3 Testdriven utveckling

Testdriven utveckling bygger på att man skriver omfattande testprogram som koden måste klara innan man implementerar den. Detta gör man för att försäkra sig om att koden fungerar som den ska och på så sätt undviker en massa buggar. Det gör även att man inte måste gå tillbaka lika ofta och skriva om kod som man märker inte fungerar som den ska senare.

Ett bra hjälpmedel till testdriven utveckling är JUnit. JUnit låter utvecklaren köra metoder från programmet och sedan ange vad för resultat som är förväntat, till exempel att en variabel ska ha ett visst värde. JUnit kör sedan igenom alla tester och berättar hur många som klarade sig.

3. Metod

Inledningsvis skapades en domänmodell i enlighet med domändriven design. Utifrån domänmodellen fastställdes de högst prioriterade “use cases” eller användningsfall. Dessa användningsfall beskriver bland annat hur en spelare ansluter till ett spel och hur spelaren sedan faktiskt spelar spelet.

Innan någon kod började skrivas till applikationen skrevs ett “Requirment Analysis Document” (RAD). I RAD:en fastställdes applikationens mål och specifikationer. Ett sådant dokument är i verkligheten ett fast dokument från kunden som specifierar vad arbetstagaren ska göra. Då det inte fanns någon riktig kund under projektet var RAD:en något flexibel som kunde ändras under projektets gång.

Koden skrevs enligt testdriven utveckling. Då applikationen innehåller en del komplexa metoder kunde inte tester skrivas till allt.

Utvecklingen skedde iterativt, där ‘use-cases’ lades till undan för undan. Under utvecklingen skrevs en SDD för att specificera exakt hur spelet skulle utvecklas och för att kunna ge en överblick över spelets kod när allt var färdig utvecklat.

3.1 Tekniska lösningar

Projektet är skrivet i java, utan externa ramverk.

3.2 Nätverkslösningar

För att data skall skickas och tas emot på ett korrekt och synkroniserat sätt har modellen en server-klient-struktur där servern i korta drag motsvarar MVC-arkitekturens modell, och klienten motsvarar vyn. Vid uppstart väljer en spelare att stå som host* för spelsessionen och startar upp servern mot en viss port. Samtliga spelare ansluter sedan med sin klient mot servern med denna port och värdens IP-adress.

För att synkronisera spelets uppdateringsflöde körs en uppdateringsfrekvens på 60Hz på både server och klient enligt följande princip:

Klient:
1: Klienten lyssnar efter nya anrop från servern som temporärt lagras. När ny data tas emot skrivs den gamla över.
2: I början av en uppdateringscykel hämtas senast motagen data från servern.
3: Klientens vy uppdateras med mottagen data.
4: Användarens input översetts till data som kan användas av modellen.
5: När uppdateringscykeln är avklarad skickas förändrad data oavkortat till servern.

Server:
1: Serverns trådar lyssnar efter anrop från respektive klient. Alla förändringar sparas i realtid.
2: När en uppdateringscykel startar hämtas en sammanställning av all mottagen data från respektive klient.
3: Logik utförs
4: När uppdateringscykeln är avklarad skickas behandlad data till respektive klient.

För att kunna skicka data mellan olika datorer över internet kommer projektet använda User Datagram Protocol, eller UDP som det förkortas. Med UDP kan man skicka strängar med text över internet till en specifik IP adress. Dessa strängar får sedan tolkas av mottagaren till något vettigt. I programmet kan ett paket se ut “move;5;5;0.3”, vilket servern tolkar som att den ska göra kommando “move” med 5, 5, 0.3 som indata. Strängen delas upp med semikolon så att det ska bli lättare att tolka meddelandet.

3.3 Arbetsflöde

Projektet drivs frammåt med hjälp av regelbundna möten där gruppen diskuterar vad som har gjorts sedan förra mötet, vad som inte blev klart och vad som ska göras till nästa möte. Till en början ligger fokus på att komma på vad som ska vara i programmet genom att göra en domänmodell och olika use-cases.

4. Resultat

4.1 Beskrivning av applikationen

4.1.1 Server

I servern ligger hela modellen från MVC. Den har hand om att uppdatera alla spelares, zombies och projektilers positioner. Den kontrollerar även för kollisioner mellan units och väggar. Servern lyssnar och skickar information till klienterna genom att den gör en ny tråd för varje ny klient som ansluter. Dessa trådar ligger hela tiden och lyssnar efter ny indata från klienten och skickar ut uppdaterad data varje gång servern uppdateras.

4.1.2 Klient

I klienten ligger controller och view från MVC. Den har även en så kallad “proxy model”, en falsk model som istället för att uppdatera informationen när controllern säger till, skickar vidare till servern att uppdatera sig. Detta gör att klienten fortfarande kan följa MVC fastän den inte har en egen model. För att kunna skicka data till servern så har klienten en klass som heter “ServerCommunicator”. Denna klass har metoder som proxymodellen kan kalla på för att skicka data.

4.2 Spelprototyp

Spelet går ut på att överleva så många vågor av fiender (eller mer specifikt zombies) som möjligt. Spelet går att spelas med endast en spelare men har designats för att spelas med flera spelare.

5. Diskussion

5.1 Utbyggnadsmöjligheter

Något som kunde ha varit bra att lägga till är en “game lobby” innan spelet startar där man kan välja karaktär och se till så att alla kommer in innan spelet startar. Detta hade varit bra eftersom man hade haft tid på sig att förbereda sig. Nu släpps man rakt in i spelet när man ansluter.

En annan sak som skulle varit bra att implementera är en chatt funktion. Detta gör det lättare för spelare att kommunicera med varandra under spelets gång om de inte sitter i skype eller liknande.

I den nuvarande applikationen måste man starta en klient samtidigt som man startar en server. Att separera server och klient skulle göra det möjligt att ha dedikerad hårdvara till att endast köra en server och inte själva spelet. Detta skulle i sin tur avbelasta klienten som startar ett spel.

5.2 Alternativa designval

5.3 Arbetsmetoder

Agil arbetsmetod

Referenser

Evans, E (2004) Domain-driven design, Addison-Wesley Professional

(2015) *Model-View-Controller*. Hämtad 2016-05-08 från: https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

Rouse, M (2015), *What is UDP?*. Hämtad 2016-05-08 från: http://searchsoa.techtarget.com/definition/UDP

Watson, G (2015) *Professor shows cooperative video game play elicits pro-social behavior.* Hämtad 2016-05-09 från: http://tidings.ttu.edu/posts/2015/05/cooperative-video-game-play-elicits-pro-social-behavior