Chalmers tekniska högskola
LSP310 - Kommunikation och ingenjörskompetens

Marcus Pettersson
Martin Sixtensson
Gustav Svensson
Erik Sänne

Zombienado

Överlev och kämpa med dina vänner mot zombieapokalypsens odöda varelser

Sammandrag
1. Inledning…….. 1
1.1 Bakgrund
1.2 Syfte
2. Teori……... 2
2.1 Domändriven design
2.2 Arkitektur
2.2.1 Model-View-Controller
2.2.2 Server-Klient
2.2.3 Internet Protokoll
2.2.4 Nätverksport
2.3 Testdriven utveckling
2.3.1 JUnit
3. Metod…….. 5
3.1 Arbetsflöde
3.2 Tekniska lösningar
3.3 Nätverks lösningar
3.3.1 Protokoll
3.3.2 Spelets datapaket
3.3.3 Spelsessionen
4. Resultat……..…7
4.1 Beskrivning av applikationen
4.1.1 Systemflöde
4.1.2 Server
4.1.3 Klient
4.2 Spelprototyp
5. Diskussion…….. 11
5.1 Utbyggnadsmöjligheter
5.2 Alternativa designval
Källor och referenser…….13

Sammandrag

Med syftet att utveckla ett kooperativt flerspelar spel har spelmotor och nätverk undersökts. Under utvecklingen lades stort fokus på att strukturera koden på ett korrekt sätt och att inkorporera nätverkslösningar för att tillåta flera spelare i ett spel. Detta resulterade i spelet Zombienado, ett kooperativt *zombie-shooter*-spel. I Zombienado är ditt mål att överleva tillsammans med upp till tre andra spelare mot horder av zombier.

1. Inledning

1.1 Bakgrund

Sen de första kommersiella privatdatorerna blev tillgängliga har datorspel varit ett ständigt växande fenomen. Idag ses det som en av de största underhållnings-formerna, med över en miljard så kallade “gamers” runt om i världen(Takahashi, 2013). Trots detta kan spel ses som något osocialt enligt sociala normer. Under senare år har dock kooperativa spel ökat i popularitet i takt med att nätverks- och kontrollteknologier förbättrats. Målet med kooperativa spel är att samarbeta med andra spelare för att uppnå ett gemensamt mål, vare sig det är något destruktivt som att förstöra eller något kreativt som att bygga. Kooperativa spel har till och med visat sig kunna ha positiva sociala effekter, åtminstone enligt John Velez, biträdande professor från Texas Technology University. *(Watson, 2015)*

Kooperativa spel kan spelas på olika sätt, det kan vara fysiskt tillsammans, då alla spelare är samlade runt en dator eller konsol, eller över nätet, alla på var sin dator. För att ett spel ska fungera över ett nätverk krävs dock, utöver gemenskap, även någon form av kommunikation på teknisk nivå.

1.2 Syfte

Projektets syfte är att undersöka hur motorn till ett spel fungerar, dessutom med fokus på nätverk, och utifrån detta skapa ett kooperativt flerspelarspel av genren *zombie shooter*. Det skall struktureras efter, och följa de principer som lärts ut gällande mjukvaruutveckling under första året på IT-programmet på Chalmers.

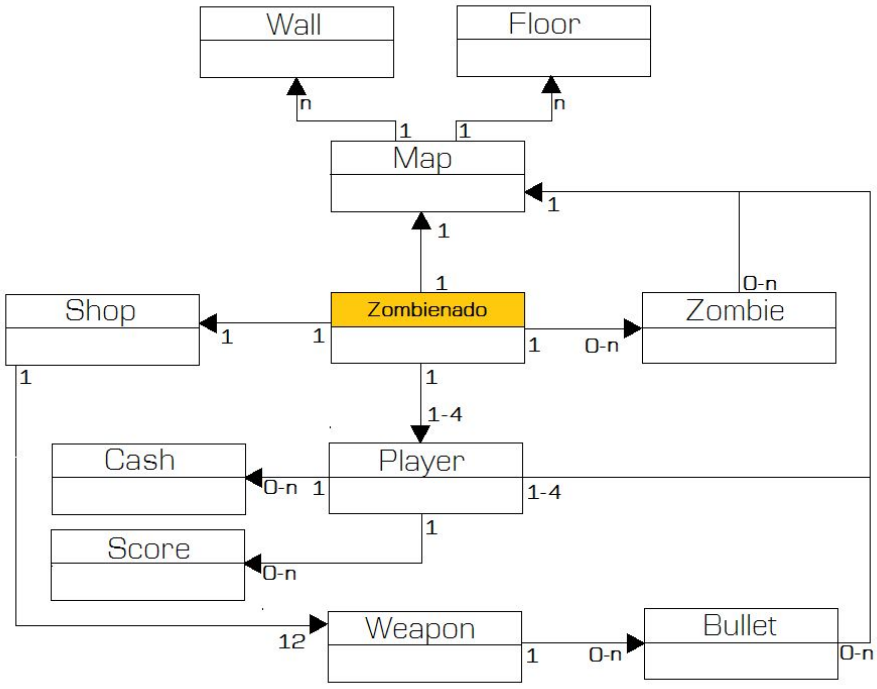
2. Teori

Här beskrivs teorin bakom de tekniker och metoder som använts i, och under utvecklingen av applikationen.

2.1 Domändriven design

Domändriven design innebär att mjukvaran modelleras genom ett samarbete mellan utvecklare och domänexperter med språket riktat mot domänen (Evans, 2004).

I en domänmodell beskrivs i stora drag hur mjukvaran skall vara uppbyggd, utan någon användning av tekniska termer. Domänspecifika termer som är vanligt förekommande i spel används för att tydliggöra specifikationen så mycket som möjligt och underlätta utvecklingsprocessen. Ett exempel på domänmodell kan ses i *figur 1*.



Figur 1: Ett exempel på domänmodell

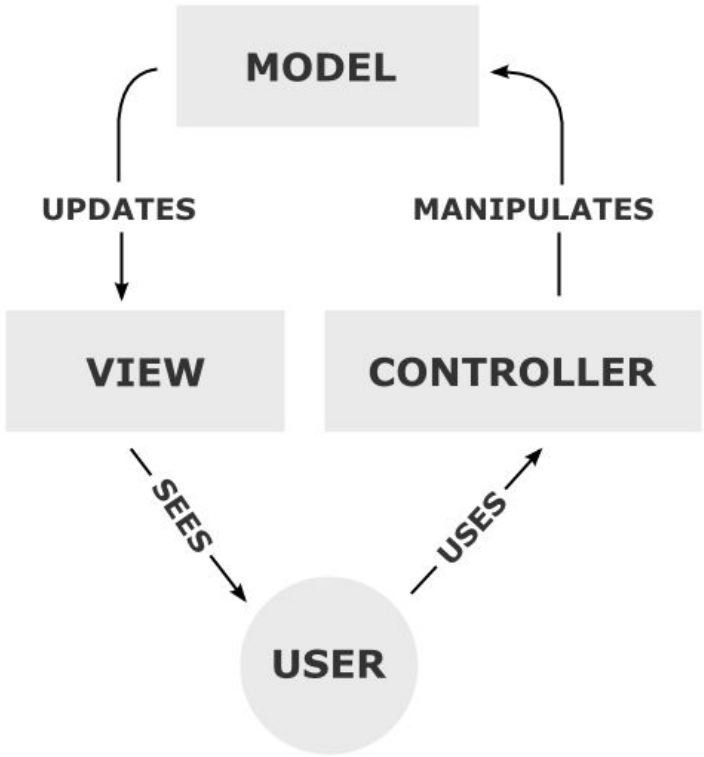
2.2 Arkitektur

2.2.1 Model-View-Controller

Model-View-Controller, eller MVC är en systemarkitektur som bygger på att separera logik och grafik.

I Modell-delen hanteras applikationens data, logik och regler. Vy-delen hanterar applikationens grafiska del. Det som händer eller ändras i modellen representeras grafiskt i vyn.
Kontroller-delen är den del som tar hand om applikationens användarinmatning, såsom musklick, tangentslag och datainmatning. (*Model-View-Controller*, 2015)

En enkel beskrivning är att användaren använder sig av kontrollen för att manipulera modellen som i sin tur uppdaterar vyn för att visa användaren vad som har hänt (*se figur 2*).



Figur 2: Ett förklarande diagram över Model-View-Controller

Fördelen med MVC är att man lättare kan byta ut komponenterna.
Ett exempel kan vara att byta ut en vy som representerar modellens information som ett stapeldiagram mot en vy som istället representerar det som ett tårtdiagram.

2.2.2 Server-Klient

Server-Klient är en IT-arkitektur som ämnar att dela upp arbetslasten mellan en server och en eller flera klienter. En server delar med sig av sina resurser och utför arbetsuppgifter åt sina klienter men en klient delar inte med sig av sina resurser till de andra klienterna om de inte går genom servern. Detta beror på att i en server-klient struktur kommunicerar aldrig klienterna

direkt med varandra utan endast med servern som i sin tur kommunicerar med de andra klienterna(Edwards, Harkey, Orfali, 1999).

2.2.3 Internet Protokoll

För att kunna kommunicera mellan datorer används främst “User Datagram Protocol” (UDP) eller “Transmission Controll Protocol” (TCP). UDP är ett mindre kontrollerat protokoll som går ut på att skicka iväg paket med data till en specifik IP-adress. Så fort datapaketet skickats iväg glöms det bort av avsändaren vilket gör att det inte finns någon garanti för att paketet faktiskt når mottagaren (vilket kallas paketförlust). Med en TCP anslutning etableras istället en session mellan avsändare och mottagare med en relativt pålitlig dataström. Till skillnad från UDP så följer felkorregering av datan, och det är också en större chans att datan når målet, men detta medför även fördröjningar. (Rouse, 2015)

2.2.4 Nätverksport

En port i nätverkssammanhang är kommunikationens ändpunkt. Nätverksportar är virtuella portar vars syfte är att dela upp inkommande trafik så att all data kommer till rätt program (*Hur fungerar en port*, 2008). När servern startas sägs den åt att lyssna på en viss port, och det är då viktigt att rätt port anges för alla klienter som vill ansluta.

2.3 Testdriven utveckling

Testdriven utveckling bygger på att ett omfattande testprogram skrivs som koden måste klara innan den implementeras. Detta görs för att försäkra att koden fungerar som den ska och på så sätt undvika större uppkomst av buggar. Det gör även att utvecklaren inte måste gå tillbaka lika ofta och skriva om kod som inte fungerar som den ska senare. (Palermo, 2006)

2.3.1 JUnit

Ett 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

Nedan följer hur utvecklingen av applikationen gått till väga.

3.1 Arbetsflöde

Inledningsvis skapades en domänmodell i enlighet med domändriven design. Då Zombienados målgrupp anses vara ungdomar med viss erfarenhet av datorspel, riktades domänmodellen till just denna grupp.

Utifrån domänmodellen fastställdes de högst prioriterade *use cases* eller användningsfallen. Dessa användningsfall beskriver hur en spelare ansluter till ett spel och hur spelaren faktiskt spelar spelet. Detaljerade beskrivningar av alla fastställda användningsfall finns i bilaga *Use Cases*.

Innan någon kod började skrivas till applikationen skrevs ett *Requirment Analysis Document* (RAD). I RAD-dokumentet fastställdes applikationens mål och specifikationer. Ett sådant dokument är i verkligheten ett fast dokument från kunden som specificerar vad arbetstagaren ska göra. Då det inte fanns någon riktig kund under projektet var RAD-dokumentet något flexibelt som ändrades under projektets gång.

Koden skrevs enligt testdriven utveckling till den mån det gick. Till detta användes programmet JUnit. Då applikationen innehåller delar som körs hela tiden, vare sig användaren ger någon inmatning eller ej, uppdateras de så ofta att det visade sig vara svårt att testa allt.

Utvecklingen skedde iterativt, där användningsfallen lades till allteftersom. Under utvecklingen skrevs ett *System Design Document*(SDD) för att specificera exakt hur spelet skulle utvecklas och för att kunna ge en överblick över spelets kod när allt var färdigutvecklat.

3.2 Tekniska lösningar

För att lyckas åstadkomma ett spel som konstant uppdaterar i realtid behövs ett kontinueligt flöde med operationer. En ganska simpelt men väl använd princip är att detta flöde består av tre faser, en s.k initialiserings-/inladdningsfas, en logikfas, och en renderingsfas. När spelet exekveras skall alltså all nödvändig data laddas in och när själva spelsessionen startas så skall detta flöde enbart alternera mellan logik- och renderingsfasen. Logikfasen är alltså ämnad att hantera alla händelser i spelet (inkluderat inmatning), och renderingsfasen är det som presenterar spelet på skärmen. På grund av att spelet är nätverksbaserat behövs dock denna modell utökas något. Hur detta gjorts nämns senare i rapporten.

3.3 Nätverkslösningar

3.3.1 Protokoll

Spelets koppling mellan server och klient sker via User Datagram Protokollet på grund av dess lägre bandbreddspåverkan och dess lägre fördröjningar. Trots risken med paketförlust utgör detta i praktiken ingen större fara för spelet, eftersom ny data skickas såpass ofta att felen med största sannolikhet förblir oupptäckta.

3.3.2 Spelets datapaket

I spelet består datan i dessa paket enbart av kommandon som specificeras i strängar. Dessa strängar tolkas av mottagaren och översätts till någon form av data som är användbar för applikationen. En sådan sträng kan exempelvis vara utformad enligt följande: “move;5;5;0.3”, vilket servern tolkar som att den ska göra kommandot “move” med argumenten 5, 5, 0.3 som indata. I just detta fall symboliserar de två första “talen” spelarens positionsförändring i x- respektive y-led, och det sista talet är spelarens absoluta rotation. Strängen delas upp med semikolon så att det ska bli lättare att tolka meddelandet.

3.3.3 Spelsessionen

För att data skall skickas och tas emot på ett korrekt och synkroniserat sätt följer modellen en server-klientstruktur där servern i korta drag motsvarar MVC-arkitekturens modell, och klienten motsvarar vyn. Vid uppstart väljer en spelare att stå som värd, såkallad ‘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.

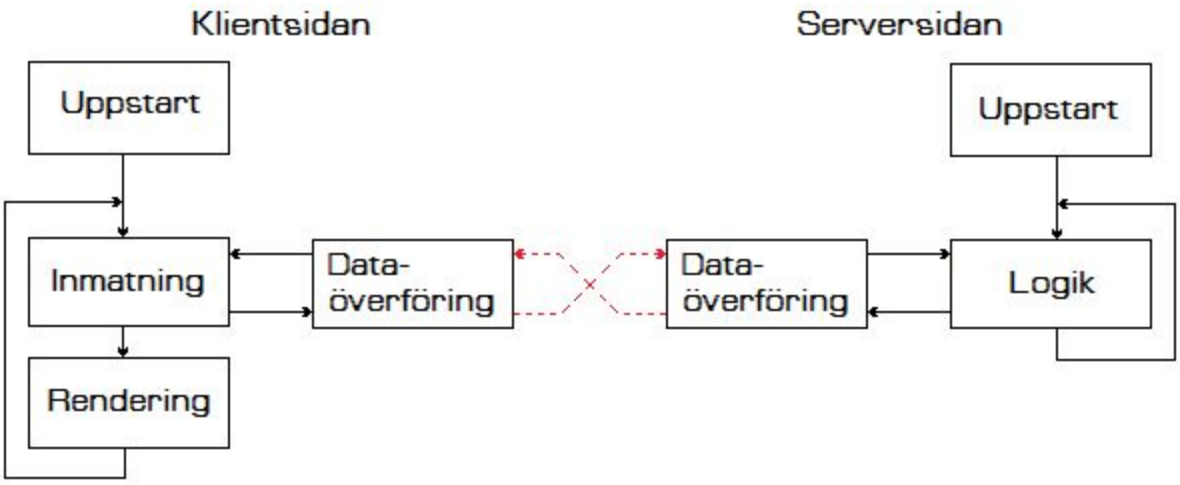
4. Resultat

Här presenteras den slutgiltiga strukturen, och designen för programprototypen.

4.1 Beskrivning av applikationen

4.1.1 Systemflöde

För att spelet ska fungera som det gör utförs ett flertal sekvenskedjor. Eftersom mycket i applikationen sker parallellt är det svårt att visualisera hela flödet, men en övergripande beskrivning av de primära sekvenserna beskrivs i figur 3.



Figur 3: De primära flödena i spelet

Både klienten och servern har en uppdateringsfrekvens (s.k. tickrate) på 60Hz för att allting ska uppdateras på ett synkroniserat och visuellt behagande sätt. Om en användares dator inte klarar av att upprätthålla den frekvensen, så tar klienten hänsyn till detta och anpassar datan som skickas till servern, så att alla spelare exempelvis går lika fort trots att en har en långsammare uppdateringscykel.

4.1.2 Server

I servern ligger hela modellen från MVC-strukturen. Den har hand om att uppdatera alla spelets entiteters positioner, och den ansvarar även för andra händelser så som kollision.
Servern tilldelar en egen parallell (skild från huvudflödet) process för varje klient som ansluter, vars enda uppgift är att oavbrutet, under hela spelsessionen, lyssna efter anrop från sin specifika klient.

I början av en uppdateringscykel hämtas en sammanställning av den senast mottagna datan från respektive klient, och när denna behandlats så skickas den nya informationen tillbaka till klienterna.

4.1.3 Klient

I klienten ligger kontrollen och vyn från MVC. I dess flöde översätts användarens inmatning i form av knapptryckningar och musklick, till handlingar i spelet. I varje uppdateringscykel hämtas den senast mottagna datan från servern, och efter att användarens handlingar har utförts så skickas den nya informationen tillbaka.

När detta är klart inleds renderingsprocessen genom att skapa en ny bildruta (s.k frame) som ska representera spelets befintliga stadie. Alla spelets komponenter målas ut på denna efter den data som erhållits från servern och när bildrutan är klar presenteras den för spelaren. Då detta (för de flesta) sker 60 gånger per sekund, så upplevs det som att saker rör sig på ett cinematiskt sätt.

4.2 Spelprototyp

Zombienado går ut på att överleva så många vågor av zombier som möjligt. Spelet går att spelas med endast en spelare men har designats med upp till fyra spelare i åtanke. För att ett spel ska kunna starta måste en spelare välja att vara värd genom att skapa och husera en spel-server. De andra medspelarna kan sedan ansluta till denna server genom att skriva in värdens IP-adress och värdens angivna port.

När servern har skapats börjar spelet. När en annan spelare ansluter sig till servern skapas hens karaktär i spelet och kan medverka i spelet. Zombier börjar skapas när spelet startar och flockas kring närmsta spelaren(*se figur 4*).

Figur 4: Skärmbild på spelet där zombier flockas kring spelaren.

I spelet kan spelare avfyra ett flertal vapen och på så sett döda zombier. Dödandet av en zombie belönar alla spelare med pengar och resulterar i att zombien i fråga tas bort ur spelet. Spelarna tar skada när de vidrör zombierna. Om en spelare tar tillräckligt med skada för att få slut på liv, dör den och kan inte längre påverka spelet. De döda spelarna kan observera spelet som osynliga spöken men får inte längre några pengar från dödade zombier. Överlever dock resten av spelarna en våg, återupplevas alla döda spelare.

Spelare börjar alltid med ett vapen. Mellan varje våg kan spelaren välja att köpa ett nytt vapen(*se figur 5*). För att kunna köpa ett vapen används spelarens individuella pengar som spelaren erhåller genom att zombier som dör.

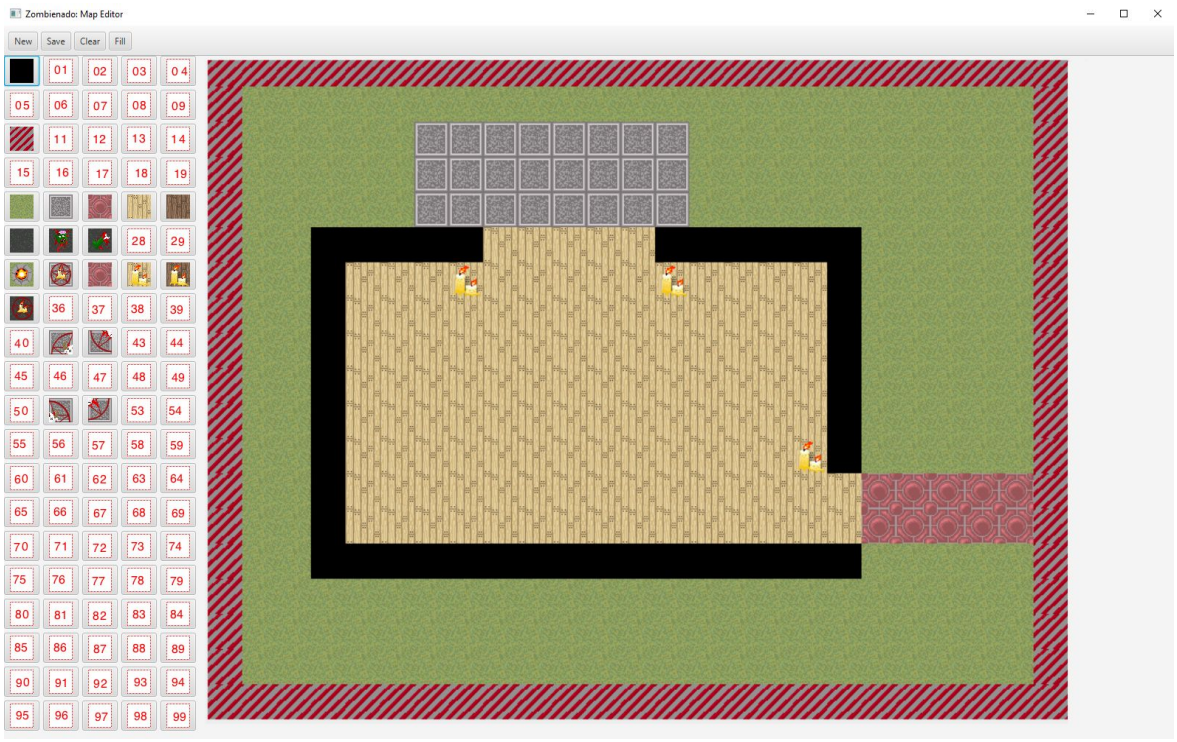
Figur 5: En skärmbild av spelets ‘affär’ där du har möjlighet att köpa vapen mellan vågor.

Det finns inget slutmål i spelet och spelet kan därmed aldrig vinnas. Spelet slutar när alla spelare har dött under en och samma våg. Spelarnas sammanlagda poäng visas då och upp spelet avslutas. För att utmana spelarna trappas svårighetsgraden upp varje våg genom att höja antalet fiender som skapas.

För att skapa ett atmosfäriskt spelscenario tillämpades en del visuella och hörbara effekter.
De visuella effekterna som tillämpades var skuggor och animationer. Ett ljussystem tillämpades i spelet. Systemet gör hela banan mörk förutom en liten del av spelarens synfält. Mörkret i spelet lägger till djup och ovisshet om vad som befinner sig utanför synfältet. Utöver det inkluderades animationer för avfyrade vapen och kikarsikten. Animationerna ger feedback till spelaren att något hänt.

För att uppsluka spelaren i spelet tillämpades några hörbara effekter: omgivande bakgrundsljud, vapen-unika skottljud och att ljud avtar över avstånd från ljudkällor. De bakgrundsljud som används i spelet är egenproducerade och ljudeffekter är tagna från copyright fria källor.

Inkluderat i spelet är ett verktyg för att skapa egna banor. Verktyget ger användaren tillgång till spelets alla olika markplattor, väggplattor och ljusplattor. Med det kan du skapa nya banor till spelet genom att rita på ett rutnät av valfri storlek(*se figur 6*).



Figur 6: Skärmbild av Zombienados *banredigerare.*

5. Diskussion

Följande kapitel diskuterar den slutgiltiga applikationen, hur den kan förbättras och hur den kunde utvecklats annorlunda.

5.1 Utbyggnadsmöjligheter

Då spelet endast utvecklats som en prototyp finns det många utbyggnadsmöjligheter av spelet som har diskuterats. Något som diskuterades väldigt tidigt i utvecklingen var en så kallad *spellobby* som spelare förs in i innan spelet startar. En *spellobby* skulle göra det möjligt att se vilka som är med och spelar innan spelet startar. Detta för att kunna se till att alla kommer med men även sådana funktioner som att välja hur din spelkaraktär ska se ut eller vad den ska ha för vapen. Detta hade även gett spelare lite tid att förbereda sig innan spelet börjar. Nu släpps spelarna rakt in i spelet när man ansluter.

Spelets fokus ligger på kooperativt spelande. För att kunna samarbeta med varandra behöver spelare kunna kommunicera med varandra. Någon möjlighet till att kunna kommunicera med varandra finns ej i spelet utan får ske utanför applikationen. En simpel chatt-funktion eller gest-system(*emotes*) skulle förbättra spelets kooperativa känsla markant.

På den mer tekniska sidan finns även där utbyggnadsmöjligheter. 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.

För att starta ett spel eller gå med i ett pågående spel krävs det att du skriver in vilket port nummer som ska användas. Då det krävs en del kunskap om datorer och nätverk för att veta vad en port är och varför den behövs kunde det förenkla spelet om det fanns en alternativ lösning där spelet använder sig av en förbestämd port istället för en som spelaren själv skriver in.

Då det är möjligt att skapa egna banor till spelet vore en funktion för att dela med sig av sina banor till andra spelare väldigt användbar. När en bana skapas så sparas den som en textfil med ett rutnät av siffror som applikationen kan läsa av för att identifiera hur banan ska återspeglas i spelet. En bana läses bara in en gång per spelsession vilket innebär att servern skulle kunna skicka den information till klienterna innan spelet börjar. Ett problem med detta är att då vi använder oss av UDP som inte alltid är tillförlitligt skulle det kunna ge klienter felaktiga banor. Alternativt skulle en separat tjänst kunna utvecklas för att användare ska kunna dela med sig av banor.

5.2 Alternativa designval

Att låta spelet utnyttja sig av färdiga ramverk hade haft en stor påverkan på slutprototypen.

Ett ramverk för spelutveckling hade hjälp till att skapa en bättre kodstruktur och även påverkat prestandaåtgången i renderingsfasen positivt, vilket skulle underlätta ifall spelet blivit mer omfattande. Ännu större påverkan hade ett ramverk för nätverkskopplingen haft, det hade också bidragit till en signifikant bättre struktur, och dessutom tillåtit mer tid till utvecklingen av själva spelet, då kopplingen mellan server och klient varit det mest tidskrävande.

Referenser

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

Datromagazin (2008) *Hur fungerar en port?*, Hämtad 2016-05-26 från: http://www.datormagazin.se/2008/hur-fungerar-en-port/

Edwards, J. Harkey, D. Orfali, R. (1999) *Client/Server Survival Guide, 3rd Edition*, Wiley

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

Palermo, Jeffrey (2006) *Guidelines for Test-Driven Development.* Hämtad 2016-05-26 från: https://msdn.microsoft.com/en-us/library/aa730844(v=vs.80).aspx

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

Takahashi, D (2013) *More than 1.2 billion people are playing games* Hämtad 2016-05-27 från: http://venturebeat.com/2013/11/25/more-than-1-2-billion-people-are-playing-games/

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