Pixel Zombie

En spelutvecklingsrapport

Eskil Brännerud

Marcus Medoc

Markus Hanna

Carl Gröning

Anton Cajander

Kungliga Tekniska Högskolan, VT 2021

Sammanfattning:

Pixel Zombie är en rapport som handlar om hur en grupp med studenter som studerar Datateknik på KTH och prövar på att utveckla ett pixel spel, som kan använda sig av en server och klient. Spelidéen kommer ifrån Box Head som är ett nostalgiskt spel för alla gruppmedlemmar. Denna rapport beskriver utvecklingsprocessen av spelet gick till och hur vissa delar löstes. Resultatet blev ett 1-4 spelar spel som använder sig av pistoler för att döda zombies, det kommer fler och nya zombies när de dör. Spelet startas samtidigt för alla spelare och när alla spelare dött så återkommer de till startskärmen.

Majoriteten av målen blev uppnådda med spelet och lite extra saker. Tyvärr blev en del problem på slutet som gjorde spelet ospelbart för vissa och ibland kraschade det.

Förord:

Rapporten beskriver mycket termer som endast folk med förstående för C-språket, eller någon liknande programmeringsspråks kunskap.

Vi vill även tacka våra handledare Jonas Willén och Johan Gustavsson.

Innehållsförteckning:

1.0 Inledning

1.1 Bakgrund

Alla behöriga förstaårsstudenter på Kungliga Tekniska Högskolan (KTH) Flemingsberg genomför en projektkurs omfattande 9 högskolepoäng i slutet på året. De med inriktningen Datateknik genomför också kursen inom datateknikens ramar, mer specifikt inom C-programmering och nätverksteknik.

Till hjälp och som förberedelse har studenterna läst kurserna HI1024 Programmering, grundkurs, HF1005 Informationsteknik och Ingenjörsmetodik, HF1006 Linjär Algebra och analys, HE1026 Digitalteknik. HE1028 Mikrodatorteknik, HF1201 Hållbar Utveckling och ergonomi samt HI1025 Operativsystem.

1.2 Affärsidé

Personer födda sent 1990-tal och tidigt 2000-tal har antagligen spelet Boxhead, ett spel där man är “Bambo” en före detta fånge som fått ett jobb av regeringen att döda zombies från ett misslyckat forskningsprojekt. Boxhead blev så populärt att skaparna utvecklade totalt åtta olika Boxheadspel. När Adobe tog ner Flash Player 31 December 2020 drogs flertalet nostalgiska spel med ner i marken, däribland Boxhead.

Flertalet andra utvecklare har försökt att återskapa Boxhead genom deras rendering och förnyande av spelet för att väcka nostalgiska minnen, men ingen har lyckats lika bra som originalet. Därför skapar vi **Pixel Zombie** för att skapa den nostalgiska känslan hos alla dem som en gång spelat Boxhead.

1.3 Avgränsningar

Projektkursens projektdel består av sex högskolepoäng vilket motsvarar 160 timmar per gruppmedlem. Det motsvarar 20 timmar per vecka i åtta veckor, för hela gruppen (5 studenter) blir det 100 timmar i veckan. Förutsättningarna för att expandera **Pixel Zombie** var bra och möjligheterna var många, men på grund av projektets tidsomfattning, gruppens icke existenta erfarenhet av SDL2, spelutveckling i allmänhet samt Scrummetoden var det viktigt att avgränsa omfånget av spelet.

1.4 Kravspecifikation

Kravspecifikationen nedan beskriver de krav som satts på **Pixel Zombie**.

- Produkten skall fungera för fyra klienter genom klient-server modellen.
- Huvuddelen av koden skall vara skriven i programmeringsspråket C.

- Koden skall vara väl och smart kommenterad.
- Koden skall vara strukturerad på ett logiskt vis i form av källkod- och headerfiler och abstrakta datatyper och dess placering i filerna.
- Installations- och instruktionsmanualer samt introduktionsfilm skall framställas.

2.0 Produktbeskrivning

Pixel Zombie är ett spel på Windows och Mac för en spelare lokalt eller två till fyra spelare online. Spelet går ut på att man som Överlevaren ska med en pistol döda alla zombies som kommer mot sig. Zombierna kommer i olika antal tills det inte finns fler, då har du överlevt nivån och ytterligare en dag i ett zombieinfesterat land. Men det är inte så lätt som man tror, överlevaren har bara tre liv och när en zombie rör dig förlorar du ett liv.

Spelet började med en meny där en kuslig bakgrundsmusik spelas. I menyn gick det att välja att se en lista över skaparna till spelet samt skaparna för musik och texturer. I en annan del av menyn valdes hur många spelare som ska spela, det var även möjligt att skriva in IP-adressen till den server användarna skall kopplas mot för att spela över nätverket.

Spelet utspelade sig i en öken där spelarna har flytt från zombierna men inte längre har någonstans att ta vägen. Den enda alternativet är göra motstånd och döda dem. I början av spelet visades banan. Spelarna och zombierna inte kan gå utanför sanden, spelarna kunde inte heller gå dit zombierna kom ifrån. Samtidigt spelas en ny läskig bakgrundsmusik som blir mer och mer dramatisk med tiden. Överst i fönstret visas också förbrukad tid sedan spelet senast startades, vilken nivå spelarna är på samt hur många liv spelaren har kvar. Liven var grafiska i form av hjärtan.

När spelet började gick zombierna mot närmaste spelare, dem kunde inte stå i samma ruta utan var tvungna att förhålla sig till varandra. När en zombie nuddade en spelare, tog spelaren skada. Spelaren kunde bara ta skada en gång i sekunden och när den gjorde det försvann även ett hjärta från heads-up-displayen (HUD). När zombierna attackerade gjorde de ett ljud och när spelaren tog skada skrek den av smärta. När spelaren förlorat alla sina liv dog den, zombierna började då följa efter dem andra spelarna. Den döda spelaren kunde då se allt som hände men inte interagera med spelet. Förlorade båda spelarna alla sina liv fick de börja om igen från menyn.

Spelarna hade en pistol som de kunde skjuta med. Eftersom det är en pistol och inte ett maskingevär/kulspruta så hade spelaren en “cooldown” som gjorde det omöjligt att skjuta om det inte gått en specifik tid. Denna tiden var satt så att spelaren kunde skjuta så verklighetsnära som möjligt. Vådabekämpning var avstängt i spelet, alltså kunde inte spelarna skjuta på varandra. Klarade spelarna att döda alla zombier kom de till

nästa nivå där det var fler zombier. Nivåerna blev svårare och svårare ju högre upp spelarna klättrade bland nivåerna.

3.0 Systemarkitektur

3.1 SDL Net

För att göra detta spel till ett multiplayer-spel så använde vi oss av biblioteket SDL_net för att kunna skicka data mellan olika datorer över internet. SDL_net är likt sin storebror platforms-oberoende. Vi använde detta bibliotek för att kunna kommunicera mellan datorerna enligt Klient/Server modellen. Både klienten och serverns programvara byggdes i C.

3.1.1 Server

Servern använder sig av User Datagram Protocol även kallat UDP. Varför just UDP valdes att användas är för att det är ett “Connectionless protocol” vilket innebär att paketen inte kollas om de tagits emot. Detta må låta dåligt men hastigheten som paketen anländer behövs för ett realtids spel.

För att använda oss av UDP och kunna skicka data mellan användarna används SDL Net. Genom att skapa en server del och en klient del går det relativt enkelt att se vad för data som skall skickas till servern och vad som klienterna skall ta emot.

Genom att ha en input ruta i start menyn kan användarna välja IP-adressen som de vill ansluta sig till. Dessutom har porten satts till 2000 för att förenkla anslutningen mellan de andra användarna.

För att varje spelare ska få relevant data på ett så snabbt och resurseffektivt sätt som möjligt, så används servern som en spegel. Servern har port 2000 öppen konstant, och väntar på inkommande data. När sedan ett paket skickas från en av klienterna, så tar servern paketet och vidarebefordrar samma paket till övriga användare utan någon ytterligare bearbetning av paketet.

Servern kan hålla koll på samtliga ansluitna klienter genom att tilldela ett ID till respektive ip-adress.. Detta är samma id som sedan skickas till klienten för att klienten ska kunna veta vilken av de 4 spelaren den är. Ett Id-nummer tilldelas vid den första kontakten mellan server/klient. Servern använder detta nummer så att servern enbart ska spegla tillbaka data till övriga spelare, och inte skicka tillbaka samma paket som klienten skickade in till servern.

3.1.2 Klient

Klienten innehåller vad för information som användaren tar emot och vad för variabler som skall ändras på klientens sida. En datavariabel som tas emot är en *flag* som i funktionen `reciveData()`. *Flag* är en integer som förklarar vad för ändring som hänt, som om en annan spelare gått eller om den skjutit. För att identifiera vilken användares data som skickas har ett slags ID system skapats där varje klient får ett unikt ID, som beror på när användaren anslutit sig till servern. Denna funktionen kallas för `reciveID()` som tillkallas tills användaren fått ett ID. Detta används över hela programmet då det styr både den egna gubben då gubben inte alltid har id noll och även styr de andra gubbarna.

3.2 Grafik

Det här avsnittet beskriver hur grafiken utvecklades i **Pixel Zombie**.

3.2.1 Fönstret

Fönstret skapades med hjälp av funktionen `SDL_CreateWindow()` där höjd och bredd valdes till 1024x1024 pixlar. Där valdes även att programmet skall centreras på skärmen vid start samt namnet på fönstret, tillika namnet på spelet: **Pixel Zombie**. För att användaren ska bli bekant med programmet och kunna urskilja det från andra program byttes den förinställda fönsterikonen ut mot en egentillverkad ikon gjord från så kallade spritesheets från spelaren och zombierna. Detta gjordes i bildredigeringsprogrammet paint.net [1]. För att sätta ikonen på rätt plats användes `SDL_SetWindowIcon()`.

3.2.2 Menyn

Menyn består av en bakgrundsbild som laddas in genom att tillkalla `renderBackground()`. Därefter laddas textrutor in på skärmen genom att tillkalla funktionen funktionen `createTextbox()`. Funktionen tar in positionen på textrutan genom ett x-värde och y-värde samten sträng som renderas in på skärmen. För att sen skifta mellan olika delar av menyn så används funktionen `checkmousestate()` för att se om spelaren klickar inom en specifik area i x-och y-led. När spelaren klickar i rutan så sätts en specifik variabel som styr vilken text som renderas på skärmen till 1 eller 2. Det här styr därefter vilken instans av menyn som spelaren befinner sig i och vilka variabler i menyn som spelaren därefter kan interagera med genom sina knapptryck. Genom menyn så har spelaren två val för att spela. Genom att klicka på “Host game” så får spelaren en egen lobby där det går att välja antalet spelare och när spelet ska starta. Spelaren kan även välja “Find game” och hamna i en lobby där en annan spelares lobby. När spelaren som valt “Host game”

sedan väljer att starta spelet så kommer spelaren i “Find game” lobbyn påbörja sitt spel.

3.2.3 Spelets bakgrund

Spelets bakgrund är uppbyggt av tiles som är 32 gånger 32 pixlar. Dessa tiles är gjorda i pixilart.com [2] genom att göra runt 20 styckna olika bakgrund texturer. Dessa texturer väljs utav en stor 32 gånger 32 array som kallas tileGrid som då blivit relativt slumpmässigt vald vilken textur som skall vara vart. Inuti griden används hexadecimala tal för att välja vilken textur som väljs och den hoppar 32 pixlar åt höger för varje ny textur. Detta tillkallas i `renderBackground()` som går igenom varje tile och tittar vilken textur som skall vara på den.

3.2.4 Texturer och animationer

För att skapa texturerna i spelet användes funktionen `SDL_CreateTextureFromSurface()` för att skapa ett bitmönster som laddades in från bilden. När det kom till animationen utav karaktärer och zombies användes “sprite sheets”. Varje “sprite sheet” innehöll flertalet bilder som tillsammans bildar en animation för karaktärerna. Eftersom ursprungsversionen av “sprite sheeten” för spelarna endast innehöll animationer för sidorörelser så är det färre frames för rörelser upp och ner än för åt sidan då dessa var tvungna att skapas i efterhand. Animationen bestod av nio texturer för rörelse sidleds och två texturer för rörelse upp samt ner.

Zombiernas “sprite sheet” som användes var två alternerande bilder för animation i både x- och yled. För att byta bild skapades funktionerna `changeZFrameX()` samt `changeZFrameY()` som tog emot tre parametrar; den initiala bilden, den alternerande bilden samt vilket zombie. Denna funktionen tillkallades när zombien bytte position. Funktionen beräknade hur många pixlar zombien har kvar innan det skall byta bild. Gick zombien diagonalt i någon riktning skulle bilderna för höger respektive vänster visas då diagonalt inte fanns i “sprite sheeten”.

3.2.5 Heads Up Display (HUD)

En HUD kan beskrivas som en genomskinlig skärm över den vanliga skärmen med information viktigt för användaren. I spelet realiseras detta genom att rendera in informationen över all annan befintlig grafik. På spelets HUD visas passerad tid sedan senaste start, hur många hälsopoäng som den aktuella spelaren har kvar samt vilket nivå spelaren är på.

Tiden som spelaren har varit aktiv i spelet visas i HUDen. Tiden presenteras på toppen av skärmen i mitten. För att få den korrekta tiden används funktionen `createTimer()` för att sedan rendera sekunder, minuter och timmar genom `createTextFromInt()` . Informationen om hur lång tid som gått från att

spelsessionen startats `getSeconds()` som returnerar antalet sekunder spelet har varit aktivt från funktionen `startGameTimer()`.

I HUDen fanns även hälsopoäng som tillät spelaren att grafiskt, i form av hjärtan, se hur många liv denne hade kvar. Precis som med de andra texturerna laddades dessa hjärtan in från en .png fil till en `SDL_Surface` struktur. Denna konverterades sedan till en `SDL_Texture` struktur med hjälp av `SDL_CreateTextureFromSurface()`. Rätt antal hjärtan renderades sedan in genom att kontrollera hur många hälsopoäng den aktuella spelaren har kvar och endast rendera det antalet.

Nivån som spelaren är på visas när spelet startat i I HUDen. för varje "våg" av zombies som aktiva spelare dödar så ökar nivån för spelarna. Nivån visas högst upp i det vänstra hörnet för varje spelare. Nivån som visas beror på variabeln `currentlevel` som tillkallas med hjälp av `getCurrentLevel()`. Nivån renderas sedan in genom funktionen `createTextFromInt()`.

3.3 Ljud och oljud

Grafik är ofta den största delen en användare tänker på när denne spelar, men det som skapar känslor medvetet och omedvetet är ljud. Därför var det viktigt att lägga till bakgrundsmusik samt ljudeffekter i spelet. Som tidigare nämnt finns i programmet finns två olika typer av ljud, bakgrundsmusik i form av .mp3 filer vilka är längre än 10 sekunder samt .wav filer vilka är kortare än 10 sekunder. Initieringen av ljudet kräver att man öppnar upp frekvens, format, kanaler samt bitstorlek. Att ladda in ljuden från mapp är enkelt då det endast behöver ladda in `Mix_LoadMUS()/Mix_LoadWAV()` i en `Mix_Music/Mix_Chunk` variabel. Att spela upp dessa var ännu enklare då endast ett anrop till `Mix_PlayMusic()` eller `Mix_PlayChannel()` tillsammans med korrekta parametrar krävdes. Till vissa ljudeffekter användes `rand()` och `srand()` för att den till synes skulle spelas oregelbundet.

3.4 Spelmekanik

3.4.1 Tangentbordsinmatning

För att kunna styra karaktären i spelet använder man tangentbordet och i själva koden används SDLs interna händelse system (`SDL_Event`) för att registrera när en knapp är intryckt. Spelet tillåter i nuläget inte att två knappar är nedtryckta samtidigt, då kommer bara den senaste tryckta tangentens operation att utföras.

Funktionen som känner av ifall en tangent är nedtryckt eller inte heter `SDL_KEYDOWN` och använder sig av en boolesk SDL array som heter `keyboardState`. Den booleska tabellen returnerar alltså antingen 1 eller 0 beroende på om knappen är aktiv eller ej.

De knappar som används för att styra spelaren är: W(upp), S(Ner), A(Vänster), D(Höger) och LCTRL(Skjuta med vapnet).

3.4.2 Zombie AI

Zombierna fungerar genom att de konstant skannar var den närmsta spelaren befinner sig. Efter att de har uppfattat den närmsta spelarens X och Y koordinater kommer de att förflytta sig mot den punkten, antingen tills de kommer fram eller tills dess att spelaren flyttar på sig/en annan spelare är närmare. De kommer alltid sträva efter att ta den kortaste vägen för att komma till närmsta spelare, men det finns dock undantag som tvingar dem att ta en alternativ rutt.

Vi har designat zombierna så att de både skannar efter spelare men också efter andra zombies, detta har vi gjort för att undvika att de staplas flera zombies på varandra, något som skulle förvirra spelarna. Om en zombie känner av att den kommer kollidera med en annan zombie så tvingas den att stanna/byta riktning för att de inte ska gå in i varandra. Detta är en bra ide men vi märkte att det uppstod ett problem, om det är tillräckligt många zombies kan det skapa en viss förvirring i deras AI och göra att de inte längre förflyttar sig helt logiskt. Detta är inget som påverkar spelet i stort så vi valde att spendera tiden på uppgifter med högre prioritet.

3.4.3 Kollisionsdetektering Spelare/Zombie

När zombien väl lyckats ta sig fram till spelaren kommer den att stanna några pixlar bredvid spelaren och inte röra sig förrän spelaren byter position/dör.

Spelaren kommer i sin tur att bli skadad 1 hälsopoäng/sekund om detta inträffar. Detta med hjälp av en funktion `msTimer()` som endast tillåter spelaren att ta skada en gång i sekunden. Till skillnad från zombierna så kan spelaren gå “igenom” fienderna om det skulle behövas. Men inte utan att bli skadad.

Själva funktionen fungerar genom att konstant skanna ifall någon zombie är tillräcklig nära någon spelares X och Y koordinater.

3.4.4 Kollisionsdetektering Vapensystem

Varje spelare har en pistol som de kan avfyra vid valfritt tillfälle. När de väljer att använda sitt vapen kommer en projektil att avfyras i den riktning som karaktären är vänd åt. På liknande sätt som all annan kollisionsdetektering i detta spel så skannar projektilen konstant ifall den kolliderar med någon zombie i dess bana. När detta

sker kommer 1 hälsopoäng att dras från den första zombien som träffas. Projektilen kommer sedan att försvinna efter att den träffat en fiende.

3.4.5 Spelare/Zombie hälsopoäng

Spelarnas och zombierna hälsopoäng fungerar på samma sätt. När man skapar spelaren/zombien använder vi oss av en struct där "`hitpoint`"/"`Alive`" finns. Spelaren har 3 hälsopoäng initialt och zombien har 1 hälsopoäng.

Om zombien blir träffad av en projektil från spelarens vapen kommer kollisions detekteringen sätta "`Alive`" till 0. Det som behöver hända nu är alltså att zombien försvinner. Vi har löst det genom att använda oss av en "`If`" sats i zombie redigeringsfunktionen, zombie kollisionsdetekteringen mot andra zombies, zombie kollisionsdetekteringen mot spelare samt funktionen som känner av ifall en projektil träffar en fiende. "`If` "satsen känner av ifall "`Alive`" är 0 och väljer då att inte utföra funktionen för den specifika zombien.

Man kan alltså säga att zombien inte dör, utan bara blir osynlig, ofarlig och inte kan träffas av projektiler eller andra zombies. Minnet som allokerats till den specifika zombien finns fortfarande kvar, och zombien själv försöker fortfarande att gå till närmsta spelaren.

Anledningen att vi valde denna lösning istället för att använda "`free()`" funktionen och släppa zombien helt från minnet, är att det underlättar att ha kvar zombien sen när man vill skapa nästa "våg" av fiender. När alla zombies har "`alive`" som 0 kommer en funktion som heter "`respawnZombie()`" att aktiveras och sätta alla zombies "`alive`" till 1 igen, vilket alltså gör dom "levande" igen.

"`respawnZombie()`" funktionen flyttar tillbaka zombierna till deras ursprungliga position samt skapar 2 nya zombier per nivå. Detta är vad som gör spelet utmanande.

4.0 Specifikation

Resultatet av detta projektarbete blev en “coop-shooter”. Spelet ses från en topdown-view och har stöd för upp till fyra spelare. Spelet är utvecklat och testat för Windows, samt MacOS, men borde också fungera för linux.

För kommunikation mellan användare så skickades data mellan spelare med UDP-protokollet och spelet byggdes med klient-server modellen.

Programmet skrevs i C, och använde sig av SDL samt underliggande bibliotek SDL, SDL_net, SDL_image, SDL_mixer, SDL_timer, SDL_TTF.

Följande funktioner fanns med i spelet:

- Zombierna som följer efter spelarna drivs av en AI
- Varje spelare hade tre hjärtan
- Spelet var baserat på olika nivåer med fler zombies
- HUD som visar antalet hjärtan, tid och nivå
- Spelaren har ett vapen
- Interaktiv meny
- Möjligheten att gå in i en credits del av menyn

Utifrån följande kravspecifikationer uppfy

- Produkten skall fungera för fyra klienter genom klient-server modellen.
- Servern fungerar för flera klienter samtidigt kopplade till en server.
- Huvuddelen av koden skall vara skriven i programmeringsspråket C.
- Huvuddelen av koden var skriven i programmeringsspråket C.
- Koden skall vara väl och smart kommenterad.
- Koden var väl och smart kommenterat.
- Koden skall vara strukturerad på ett logiskt vis i form av källkod- och headerfiler och abstrakta datatyper och dess placering i filerna.
- Koden har struktureras logiskt efter källkod- och headfiler samt abstrakta datatyper i samband med placering i filerna.
- Installations- och instruktionsmanualer samt introduktionsfilm skall framställas.
- Installations- och instruktionsmanualer samt introduktionsfilm har framställts för spelet.

5.0 Slutsats

5.1 Sammanfattning

Projektet är nu avslutat och vi har haft vår sista SPRINT. Under kursens gång har vi dragit många lärdomar både gällande programmering samt när det kommer till SCRUM metodiken. Ingen av oss i gruppen har tidigare jobbat med ett så stort projekt men vi har klarat det bra trots bakslag och diverse utmaningar.

Om vi börjar med programmeringen så visade det sig snabbt att det är en stor skillnad på att jobba själv kontra jobba i grupp. Hastigheten som spelet utvecklades i var över vår förväntan vilket sätter större krav på den enskilde. Något som vi upptäckte som var otroligt viktigt var därför kommunikation, det gäller att hålla gruppen informerad om vad som händer.

När det kommer till SCRUM metodiken känner vi att det fungerat på ett effektivt sätt. Hemsidan Version One är ett stort hjälpmedel eftersom det skapar en överblick om vad som ska göras och vad som redan är gjort. Vi lyckades implementera nästan alla användarberättelser som vi skapade i början av kursen, inklusive några som skapats under kursens gång. GitHub är ett program som också underlättat enormt. Det hände flera gånger att vi var tvungna att återgå till en äldre version av spelet för att lösa diverse buggar som uppstått. Men de gäller också att skapa regler för vad som laddas upp, därför var vi noga med att alltid prata med varandra och komma överens om vad som fick laddas upp till master filen.

Vi är väldigt nöjda med vår insats och känner att vi skapat ett intressant och bra spel.

5.2 Utvecklingsmöjligheter

Spelet i sin helhet uppfyller alla de krav som gruppen satte vid projektstarten. Det finns dock självklart alltid områden som kan förbättras och nya funktioner som kan implementeras. Det vi önskar addera ifall tiden fanns är fler alternativ för konsumenten, alltså större variation på “banor” att välja mellan, olika spelbara karaktärer, fler vapensystem och fiender med unika egenskaper. Det jag precis nämnt är eftertraktat bland konsumenter men ändå så valde vi att ge dem en låg prioritet eftersom dessa funktioner behöver ett redan fungerande spel för att vara relevanta.

En funktion som kan förbättras är “zombiernas” kollision med varandra och vart de väljer att gå. Det fungerar utan problem överlag men när man överlevt tillräckligt länge och fienderna ökar i antal kan det skapa en viss “förvirring” och de börjar stundvis bete sig ologiskt.

En annan funktion som kan förbättras är hur IP adressen konfigureras. Just nu är klienten hårdkodad att kommunicera med en server som ligger på samma maskin (mer

om detta senare i rapporten). Detta är en suboptimal lösning som vi inte hunnit lösa på ett bättre sätt.

Under projektets gång har vi lagt ner mycket tid på att ständigt utveckla och uppdatera våra funktioner så att det fortfarande fungerar med de nya användarberättelserna. Detta är något som vi inte förutsåg i början av kursen. En utvecklingsmöjlighet till nästa års projekt är alltså att designa funktionerna så långt det går efter de specifikationer de kommer behöva i framtiden. Det hade sparat oss mycket tid.

6.0 Referenser

[1] https://www.getpaint.net/index.html

[2] https://www.pixilart.com/draw#

[3] https://gits-15.sys.kth.se/antoncaj/Projektkurs

7.0 Appendix

7.1 Drifthandbok

7.1.1 Installera spelet

Detta spel är byggt på SDL biblioteket. För att starta detta spel, så krävs det därför att användaren installerar följande tillägg.

- SDL2: https://www.libsdl.org/download-2.0.php
- SDL_net: https://www.libsdl.org/projects/SDL_net/
- SDL_image: https://www.libsdl.org/projects/SDL_image/
- SDL_mixer: https://www.libsdl.org/projects/SDL_mixer/
- SDL_ttf: https://www.libsdl.org/projects/SDL_ttf/

SDL2, samt dess tillägg installeras alla på samma sätt. Under utvecklingsprocessen har kompilatorn MinGw använts för att kompilera. I detta fall är då bara att kopiera över mapparna “bin”,”include”, samt “lib” och dess innehåll i motsvarande mappar (med samma namn) i MinGw installations-mappen.

7.1.2 Starta servern

Innan spelet kan startas, är det viktigt att en server redan är uppstartad. Servern startas genom att starta server.exe i spelets map. För att multiplayern ska fungera utanför LAN:et så krävs det också att serverns router har en “port-forward” på port 2000, till serverns lokala ip-adress.

7.1.3 Konfigurera IP

Innan klienten kan starta spelet så måste rätt IP konfigureras. Klienten hårdkodad att kommunicera med en server som ligger på samma maskin, den använder sig därför av loopback-adressen 127.0.0.1. För att ansluta sig till en annan server, måste därför användaren manuellt ändra “127.0.0.1” på alla ställen i koden, till valfri IP. Detta är en dålig lösning som inte hunnit lösas på ett finare sätt. Programmet måste kompileras igen, efter att IP:n har ändrats.

7.1.4 Starta spelet

Spelaren startar spelklienten genom att starta den exekverbara filen “pixelzombie” och möts då av av en huvudmeny. Proceduren för att starta spelet skiljer sig något om användaren är första spelare att ansluta till servern, eller om användaren är spelare nr2-nr4

Spelare nr1

1. Från huvudmenyn så klicka på knappen “Host Game”
2. Tryck sedan på tangentbordet nr1-4 för att välja antalet spelare som ska vara med i lobbyn.
3. När samtliga spelare är anslutna till servern, så används knappen “Start Game” för att starta spelet. Spelet kommer då att starta för samtliga spelare.

Spelare nr2-4

1. Från huvudmenyn så klicka på knappen “Find Game”
2. Förutsatt att rätt ip har angetts, och servern är igång. Så kommer spelet att starta för “dig”, och de andra spelarna efter att spelare nr:1 har tryckt på “Start Game”

7.2.1 Spelkontroller

För att förflytta spelaren så används de klassiska knapparna **W** (upp), **A** (vänster), **S** (ner), och **D**-tangenterna (höger).

Attackera zombies

För att tilldela skada till en zombie så används pistolen. Denna pistol avfyras genom att trycka ner **LCTRL**-tangenten (CTRL på vänster sida av tangentbordet. Pistolen kommer då att avfyra en projektil i samma riktning som spelaren.

7.3.1 Spelets mål

Efter spelets start så kommer 1-4 spelare att spawnas in på en bana. Det kommer sedan att börja komma zombies mot spelarna. Målet med spelet är att skjuta dessa zombies, och överleva så många “waves” som möjligt.

Zombies kommer att följa den närmaste spelaren, från deras perspektiv. Zombien kommer att orsaka skada på spelaren om den står för nära.

7.4.1 Avsluta spelet

För att avsluta spelet så används **KRYSSET** längst upp i högra hörnet av spelfönstret.