Rasmus Bergström

Retro FPS -prototyyppi nykypelimoottorilla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

15.4.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Rasmus Bergström Retro FPS -prototyyppi nykypelimoottorilla 37 sivua 15.4.2018
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja	lehtori Antti Laiho

Insinöörityön tarkoituksena oli luoda vanhojen FPS-pelien hengessä niiden suunnittelua, ja toteutustapoja mukaileva FPS-pelin (First Person Shooter, ensimmäisen persoonan ammuskelupeli) prototyyppi, joka toimisi pohjana oman idean jatkokehittelyä varten.

Vanhojen FPS-pelien teknistä toteutusta ja peli- ja kenttäsuunnittelua tutkiessa selvisi, että verrattaessa uudempiin vastaaviin peleihin suurimmat erot löytyivät 2D-spritejen käytössä ja kenttäsuunnittelussa.

Prototyyppi suunniteltiin toteutettavaksi modernilla pelimoottorilla, ja siihen tehtiin kaikki tarvittavat komponentit itse käyttämällä apuna 3D-mallinnukseen ja 2D-grafiikkaan soveltuvia ohjelmistoja sekä C#-ohjelmointikieltä pelin ominaisuuksien ohjelmointiin. Prototyypille määriteltiin projektin onnistumista mittaavat kriteerit ja listattiin kaikki toimivaa prototyyppiä varten tarvittavat komponentit.

Prototyypissä toteutettiin pelin kentät käyttäen niiden rakentamiseen 3D-malleja. Viholliset ja pelistä löytyvät tavarat toteutettiin käyttämällä 2D-spritejä niiden kuvastamiseen. Kaikki pelin toiminnallisuudet ohjelmoitiin C#-ohjelmointikielellä.

Valmis prototyyppi täytti kaikki sille määritellyt kriteerit, vaikka jotkin toiminnallisuudet eivät olleet aivan niin optimoituja, kuin olisivat voineet olla. Tuloksena oli kaksi pelattavaa kenttää käsittävä FPS-peli, joka näytti ja tuntui samalta kuin esikuvansa.

Prototyypin pohjalta peli-ideaa päätettiin jatkaa harrasteprojektina, sillä itse kaikki komponentit tekemällä valmiin pelin vaatima aika ja vaiva olisi liian iso.

Avainsanat	peli, C#, FPS, Retro, ensimmäisen persoonan ammuskelupeli

Author Title Number of Pages Date	Rasmus Bergström Retro FPS prototype on a modern game engine 37 pages 15 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation Option	Software Engineering
Instructor	Antti Laiho, Senior Lecturer

The aim of this final year project was to create a working FPS game prototype in the spirit of retro FPS games, and mimic the style of their level design and gameplay design. This prototype would work as a foundation for a game idea from the student.

By analyzing the technical implementation in addition to both level and game design of retro FPS games it was discovered by comparing to newer titles in the same genre that the greatest differences were in the use of 2D sprites and in the way levels were designed.

The prototype was planned to be implemented with a modern game engine and by doing all the game assets by hand. The assets were done by using a 3D modeling software, a graphics editor for creating 2D art and C# for programming. The criteria for the prototypes success were defined, and a list was made of assets needed to successfully complete a working prototype.

The levels were implemented in this prototype by using 3D models to create them. The enemies and items found in the game were done by using 2D sprites to represent them and all the functionalities of the game were written in C#.

The finished prototype passed all the criteria that was defined at the start of the project, and was considered a successful prototype. Some functions in the game could have been more optimized. The result is a prototype with two playable levels that feels and looks just like its predecessors.

Based on the time spent making this prototype the development of this game will be continued as a hobby project, as the time and effort spent making assets by hand would be too much.

Keywords	Retro, FPS, C#, Game, First Person Shooter

Författare Titel Sidantal Datum	Rasmus Bergström Retro FPS -prototyp på en modern spelmotor 37 sidor 15 April 2018
Utbildningsprogram	Ingenjör (Yrkeshögskola)
Inriktningsalternativ/Fördjupning	Informationsteknik
Yrkesinriktad huvudämne	Programvaruteknik
Handledare	Antti Laiho, Lektor

Syftet med detta examensarbete var att utveckla ett prototypspel som grundade sig på gamla förstapersonsskjutare från 1990-talets början som skulle efterlikna både deras anda, och teknisk implementering. Prototypen används som grund för studentens eget spelidé.

Gamla förstapersonsskjutares tekniska implementering samt deras nivå- och spelplanering studerades och jämfördes med nutida liknande spel. Det upptäcktes att de största skillnaderna var i användandet av 2D-bilder både i spel och i nivåplaneringen.

På grund av informationen planerades prototypens implementering med hjälp av en modern spelmotor och alla spelets komponenter gjordes för hand. Komponenterna gjordes med hjälp av 3D-simulerings- och 2D-bildbehandlingsprogram. Kriterium för prototypens succé bestämdes och alla komponenter man skulle behöva för en fungerande version av prototypen listades.

Nivåerna i prototypen implementerades med hjälp av 3D-modeller. Fienden och övriga komponenter i nivåerna presenterades genom att använda 2D-bilder. Alla funktioner i prototypen gjordes med hjälp av C#-programmeringspråk.

Den färdiga prototypen uppfyllde alla krav som ställdes på ett lyckat spel. Medan några funktioner i spelet behöver optimeras var slutresultatet en fungerande spel som omfattade två olika spelnivåer. Spelet såg ut och fungerade precis som sina 1990-talets förebilder.

Baserat på tiden och insatsen för att utveckla prototypen kommer utvecklingen av prototype att fortsätta som ett hobbyprojekt. För att färdigställa spelet skulle det ta för mycket tid om man gjorde alla spelets komponenter för hand.

Nyckelord	retro, förstapersonsskjutare, C#, spel, FPS

Sisällys

Lyhenteet

FPS	First Person Shooter. Ensimmäisestä persoonasta kuvattu ammuntapeli.
MIDI	Musical Instrument Digital Interface (musiikkisoittimien digitaalinen liitäntä tai rajapinta).
LO FI	Low Fidelity (äänitallenne, jonka äänenlaatu on tavallista matalampi).
BSP	Binary Space Partitioning (binäärisen avaruuden osiointi). Tietokonegrafiikassa käytettävä menetelmä, jossa avaruus jaetaan konvekseihin sektoreihin.
Ray casting	Säteensuuntaus. Tietokonegrafiikassa käytetty renderöintimenetelmä.

1 Johdanto

Insinöörityön tarkoituksena oli toteuttaa vanhojen FPS-pelien (First Person Shooter eli ensimmäisen persoonan ammuskelupelien) henkinen ja niiden kenttä- ja pelisuunnittelua mukaileva prototyyppi nykyaikaista pelimoottoria hyödyntäen ja käyttäen ohjelmointiin C#-ohjelmointikieltä. Prototyypin on määrä toimia pohjana mahdollista pelin jatkokehitystä varten. Projekti toimi myös opiskelijalle oppimisympäristönä, sillä aiempaa kokemusta peliohjelmoinnista 3D-mallinnuksesta tai 2D-grafiikan tuottamisesta ei ollut.

Toiveena myös on, että insinöörityö toimisi apuvälineenä muille, jotka haluavat itse toteuttaa oman FPS-pelinsä, sillä projektin alussa löytyi hyvin vähän informaatiota tai apua siitä, kuinka 2.5D-peliä kannattaisi lähteä toteuttamaan nykypelimoottorilla.

Insinöörityöraportti koostuu viidestä luvusta, joista ensimmäisessä käydään läpi yleisesti, mitä vanhoilla FPS-peleillä tarkoitetaan, niiden käyttämiä pelimoottoreita ja niiden rajoituksia sekä teknistä toteutusta, jotta ymmärrettäisiin ratkaisuja, joihin pelien suunnittelijat päätyivät pelejä suunnitellessaan. Lisäksi käydään läpi, millaista kenttäsuunnittelua vanhat FPS-pelit edustivat ja millaista pelisuunnittelua niissä käytettiin, minkä jälkeen käydään läpi opiskelijan omia taustoja projektin suhteen sekä esitellään peli-idea.

Toisessa luvussa määritellään projektin tavoitteet, projektia varten tarvittavat komponentit ja miten ne kannattaisi toteuttaa sekä käydään läpi projektin toteutusta varten vaadittavia työkaluja.

Kolmannessa luvussa käydään läpi, mitä ehdittiin projektin aikana toteuttaa, ja hieman niiden toteutustavasta. Sen enempää ei syvennytä siihen, miten koodi toimii, vaan tarkastellaan sen sijaan tehtyihin toteutuksiin ja toiminnallisuuksiin käytettyjä metodeja.

Neljännessä luvussa tarkastellaan valmista prototyyppiä ja sitä, vastaako se projektin alussa sille asetettuja kriteerejä, sekä käydään läpi mahdollisia ongelmatilanteita, jotka projektin aikana nousivat esille ja ratkaisuja niihin. Prototyyppiä myös verrataan esikuviinsa.

Viimeisessä luvussa käydään läpi projektin aikana opitut asiat ja mietitään projektin jatkoa.

2 Vanhat FPS-pelit

2.1 Yleistä

Niin sanotuilla vanhoilla FPS-peleillä tarkoitetaan yleensä nimellä 2.5D FPS kutsuttuja sprite-pohjaisia ensimmäisen persoonan ammuskelupelejä 1990-luvun alkupuolelta (mm. Wolfenstein 3D, Doom & Doom2, Hexen, Heretic, Shadow Warrior, Duke Nukem 3D ja Rise of the Triad), mutta myös hieman myöhemmin tulleita kolmiulotteisia ensimmäisen persoonan ammuskelupelejä, joissa käytettiin polygoneja spritejen sijaan (Quake, Half-Life, Soldier of Fortune, jne.). [28.]

Näissä peleissä pelaaja etenee yleensä sokkeloita täynnä olevien kenttien läpi yksi kerrallaan kohdaten pelin edetessä uusia vihollisia ja löytäen uusia aseita. Juonta kuljetetaan yleensä kenttien välissä, mutta pääpaino on aina nopeatempoisella pelattavuudella.

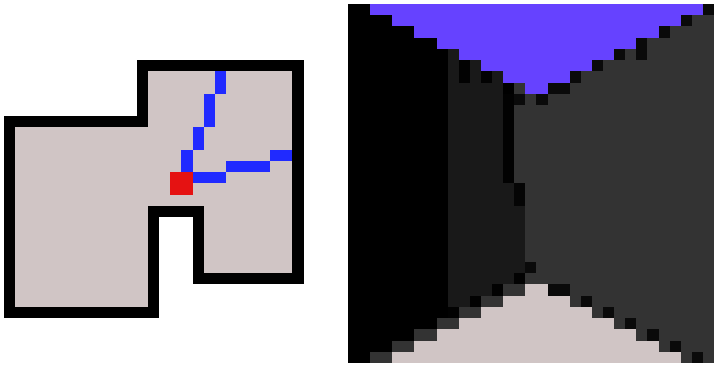
2.2 Vanhojen FPS-pelien tekninen toteutus

Seuraavassa tarkastellaan kolmea 1990-luvun alkupuolen tunnetuinta FPS-pelien pelimoottoria ja käydään pintapuolisesti läpi niiden teknistä puolta. Tarkasteltavat pelimoottorit ovat Wolfenstein 3D engine, id Tech 1 (tunnetaan myös nimellä Doom Engine) ja Ken Silvermanin Build Engine. Näiden moottoreiden historia kiteyttää melko hyvin koko 2.5D FPS-pelien kehityskaaren.

Wolfenstein 3D Engine

Wolfenstein 3D Enginenä tunnetuksi tullut pelimoottori on pääosin John Carmackin luoma pelimoottori, joka tunnetaan parhaiten id Softwaren kehittämästä nimikkopelistään Wolfenstein 3D (1992), mutta Carmack kokeili siinä käytettävää Ray casting -tekniikkaa (säteensuuntaus, tietokonegrafiikassa käytetty renderöintimenetelmä) jo aikaisemmin id Softwaren tekemissä Hovertank- (1991 ja Catacomb 3-D (1991) -peleissä. Wolfenstein 3D -pelimoottorin suurin vahvuus senaikaisiin kilpailijoihin nähden oli nopeus jolla pelimoottori renderöi kuvaruudulle kuvan, minkä ansiosta se toimi nopeasti myös hieman vanhemmilla koneilla.

Pelimoottori piirtää kuvaruudulle kolmiulotteisen ympäristön litteän kaksiulotteisen ruudukon pohjalta, jossa pelaaja liikkuu. Ray casting toimii niin, että pelissä pelaajahahmon katsomaan suuntaan ammutaan horisontaalisesti vasemmalta oikealle eteenpäin säde pelaajan kuvaruudun jokaista pystysuoraa riviä kohden. Säteen osuessa seinään pelimoottori laskee etäisyyden pelaajan ja seinän välillä. Mitä pidempi välimatka, sitä matalammaksi seinä piirretään kuvaruudulla, ja toisinpäin. [1; 2.] Kuvassa 1 havainnollistettuna, kuinka ray casting toimii



Kuva 1. Vasemmalla pelaaja 2D-kartalla ja oikealla pelaajan näkökulma kuvaruudulla [1; 2].

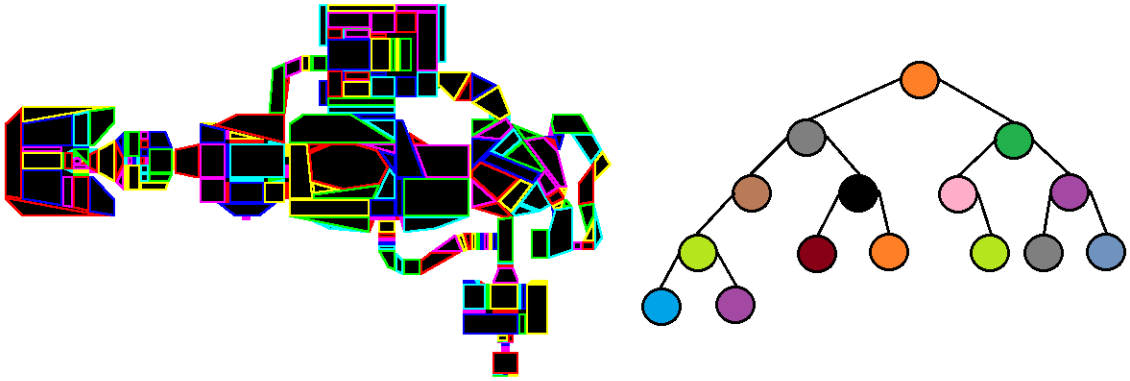
Pelimoottorilla oli kuitenkin rajoituksensa. Korkeuseroja ei ollut, kulmien piti aina olla 90 astetta, ylös tai alas ei voinut katsoa, eikä esimerkiksi hyppimistä voinut toteuttaa.

id Tech 1 (Doom Engine)

Doomia (1993) varten kehitetty id Tech 1 oli seuraava kehitysaskel John Carmackin pelimoottoriteknologiassa. Pelimoottori mahdollisti viimein korkeuserot pelissä ja eriasteiset kulmat seinissä. Pelikuva renderöitiin edelleen kaksiulotteiselle tasolle piirretystä kartasta, ja jakamalla kartta sektoreihin pystyttiin eri sektoreille määrittelemään omat korkeutensa. Pelimoottorissa hyödynnettiin 3D:n edellyttämää Z-akselia, mutta se oli melko rajoittunut ja päti lähinnä vain projektiileihin, hitscan-hyökkäyksiin, sekä vihollisiin mahdollistaakseen pääsyn sektoriin. Kiinteät objektit eivät myöskään voineet olla päällekkäin (esimerkiksi pelaaja ja vihollinen), sillä pelimoottori laajalti jätti korkeuden ja Z-koordinaatit huomiotta. Samalla moottorilla tehdyissä Heretic- ja Hexen-peleissä Z-akselin toimivuutta paranneltiin. [4.]

Carmack hyödynsi pelimoottoria luodessaan apuna Gordonin ja Chenin vuonna 1991 esittelemää renderöintimenetelmää, jossa apuna käytettiin BSP:tä (Binary Space Partitioning, suomeksi binäärinen avaruuden osiointi), josta oli kirjoitettu jo vuonna 1969. [23; 24.].

BSP:tä hyödyntäen pelimoottori ensin prosessoi kentät ennalta ja jakaa kaksiulotteisen kartan kahteen sektoriin. Nuo kaksi sektoria jaetaan edelleen kahtia, ja samaa menetelmää toistetaan, kunnes kaikki on jaettu lukuisiin konvekseihin alisektoreihin. Näin saadaan luotua hierarkkinen binääripuu täynnä solmukohtia, jossa jokainen solmukohta edustaa yhtä sektoria. Kuvassa 2 havainnollistetaan, kuinka kartta on jaettu konvekseihin alisektoreihin, sekä niistä syntyvää BSP-puuta.



Kuva 2. Binäärinen avaruuden osiointi havainnollistettuna [5].

Moottori piirtää ensin siitä sektorista, jossa pelaaja seisoo, kaikki näkyvät seinät. Moottori muistaa mihin kohtiin ruutua se on jo piirtänyt, ja käy läpi kaikki viereisetkin sektorit aloittaen aina lähimmistä ensin, jotta vältyttäisiin turhalta renderöimiseltä. Sen jälkeen, kun moottori on saanut piirrettyä kaikki tarpeelliset seinät, sekä lattian ja katon, ovat vuorossa muut asiat, kuten spritet ja läpikuultavat seinät.

Siinä, missä seinät renderöidään järjestyksessä lähimmästä kauimmaiseen, piirretään spritet kuvaruudulle aloittaen kauimmaisista potentiaalisten läpikuultavuuksien takia. Tämän takia, jos edessä on useampi vihollinen, ne kaikki renderöidään, vaikka osa niistä olisikin toisten takana. Aivan viimeiseksi piirretään kuvaruudulle UI-elementit, pelaajan terveys, ammukset, käytössä oleva ase ja niin edelleen.

Vaikka id Tech 1 olikin aikaansa nähden erittäin tehokas pelimoottori ja toimi aikansa kotitietokoneissa hyvin, niin kuin Wolfenstein 3D Engine, oli myös id Tech 1:llä rajansa,

sillä edelleenkään pelissä ei voinut hyppiä, ylös tai alas ei voinut katsoa, eikä pelimoottori tukenut kaltevia, tai kaarevia pintoja.

Doomin pelimoottori ei myöskään tukenut päällekkäisiä huoneita, sillä pelin kartta oli vain kaksiulotteinen ruudukko, eikä yhteen tasoon saanut päällekkäin huoneita. Kenttien ennalta prosessoimisessa sektoreihin oli myös se huono puoli, että se piti tehdä ennalta, ja se oli aikaa vievää. Tämä nopeutti huomattavasti pelimoottoria, mutta hintana oli, että sen vuoksi myös seinät olivat staattisia, eikä niitä voinut liikuttaa reaaliaikaisesti. [3; 5.]

Build Engine

Build Engine on Ken Silvermanin 3D Realmsille (Apogee) kehittämä pelimoottori, joka lienee 1990-luvun 2.5D-pelimoottoreista kaikkein kehittynein. Ken Silverman kehitti moottoria vuosien 1993 ja 1996 välillä, ja hän lisäsi paljon uusia ominaisuuksia, kuten ylös ja alas katsomisen, kaltevat tasot, päällekkäiset sektorit, peilit, hajoavan maaston, seinän ja lattian spritet, hyppimisen, lentämisen ja mahdollisuuden mennä veden alle. Pelin kartat piirrettiin edelleen kuvaruudulle kaksiulotteisesta kenttäruudukosta. [6; 15.]

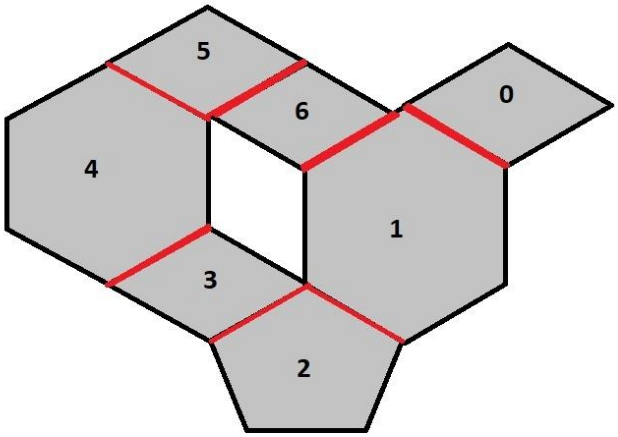
Build Enginessä renderöinti toimi aika lailla samalla tavalla kuin id Tech 1 -pelimoottorissa, mutta suurin ero oli se, että Build Engine ei käyttänyt BSP:tä seinien piirtämiseen. Sekin kuitenkin turvautui sektoreihin renderöidessään pelimaailmaa. BSP:n sijaan Build-pelimoottori käytti hyväkseen portaaleita, jotka yhdistivät sektoreita toisiinsa.

Peli renderöi ensin pelaajan näkökentässä sektorin, jossa pelaaja sillä hetkellä seisoi, ja sen jälkeen jatkoi renderöimällä mahdolliset naapurisektorit, jotka on kytketty toisiinsa portaalien avulla. Päällekkäiset sektorit olivat Build Enginellä vihdoinkin mahdollisia, kunhan molemmat sektorit eivät olleet näkyvillä pelaajalle samaan aikaan. Kuvassa 3 on havainnollistettuna, kuinka Build Enginen käyttämät portaalit (punaisella) yhdistivät eri numeroidut sektorit.

Toisin kuin id Tech 1, Build Engine ei prosessoinut karttojaan ennalta alisektoreihin, vaan hoiti kaiken reaaliaikaisesti. Tämä poisti pelimoottorilta rajoitteen liikutella seiniä ja mahdollisti tuhoutuvan maaston.

Tunnetuin Build-pelimoottorilla tehty FPS-peli on Duke Nukem 3D, joka julkaistiin vuonna 1996, ja se toi id Tech 1 -moottoriin verrattuna paljon uusia innovatiivisia ominaisuuksia. Uutta valtakautta kuitenkin kesti vain muutaman kuukauden, sillä toukokuussa 1996 id

Software julkaisi id Tech 2 -pelimoottorin tukeman Quake-pelinsä, jossa oli yksi aikansa ensimmäisistä aidosti kolmiulotteisista FPS-pelimoottoreista. [6; 15.]



Kuva 3. Build Engine käytti portaaleita BSP:n sijaan [6].

Build Enginen avulla tehtiin kuitenkin vielä tulevina vuosina monta suurta suosiota saanutta FPS-peliä, mm. Shadow Warrior (1997), Blood (1997) ja Redneck Rampage (1997).

Tämän työn kirjoitushetkellä 3D Realms kehittää uutta 2.5D-peliään, joka on määrä julkaista vuonna 2018. Peliä tehdään EDuke32:lla, joka on Ken Silvermanin Build Enginen lähdekoodeihin perustuva pelimoottori, joka tukee nykyaikaisia käyttöjärjestelmiä ja johon on lisätty uusia ominaisuuksia (mm. 3D-mallit). [13.]

Ominaisuuksien kehittyminen:

Seuraavassa on listattuna 2.5D -pelien ominaisuuksien kehityskaarta pelijulkaisujen aikajanan avulla. Listalla olevalla Ultima Underworldillä oli teknisesti kehittynyt moottori, mutta se vaati PC:ltä paljon tehoa eikä kyennyt näin ollen kilpailemaan id Softwaren moottoreiden kanssa.

- **Maze War** (1973, ensimmäisen persoonan näkökulma)

- **Hovertank 3D** (Softdisk 1991, ray casting, Wolfensteinin prototyyppimoottori)

- **Catacomb 3-D** (id Software 1991, teksturoidut seinät, Wolfensteinin prototyyppimoottori)

- **Wolfenstein 3D** (id Software 1992, nopeampi renderöinti)

- **Ultima Underworld: The Stygian Abyss** (Blue Sky Productions 1992, teksturoidut seinät, lattiat ja katot, 45 asteen kulmat seinissä, korkeuserot, valaistus, ylös ja alas katsominen, hyppääminen)

- **Shadowcaster** (Raven Software 1993, kaltevat lattiat, teksturoidut lattiat ja katot. pelaajasta poispäin heikentyvä valaistus, Shadowcasterin moottori oli John Carmackin kehittämä Wolfensteinin ja Doomin välimuoto)

- **Doom** (id Software 1993)

- **Heretic** (Raven Software 1994, ylös ja alas katsominen ja inventaariosysteemi)

- **Dark Forces** (LucasArts 1995, kyykkyyn meneminen, hyppiminen, uiminen, päällekkäiset huoneet, pelimoottori hyvin samankaltainen kuin Build Engine)

- **Hexen** (Raven Software 1995, hahmoluokat, teki suosituksi hubimaisen kenttärakenteen FPS-peleissä)

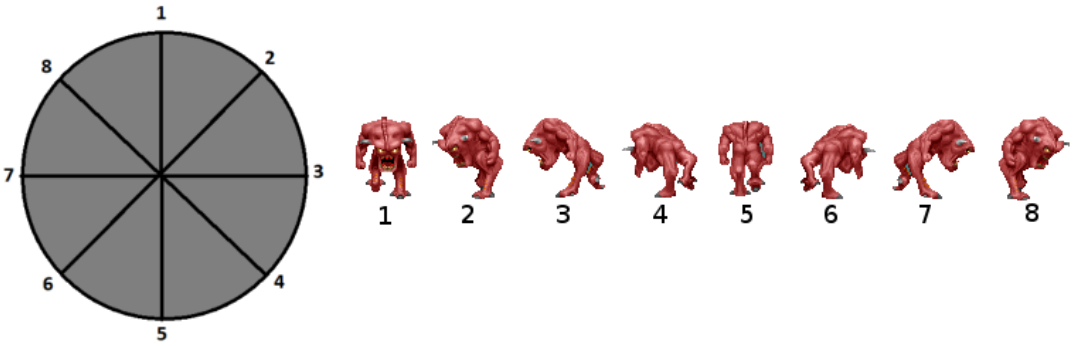
- **Duke Nukem 3D** (3D Realms 1996, tuhoutuva maasto, päällekkäiset huoneet)

- **Blood** (Monolith + 3D Realms 1997, Voxeli-objektit pelissä)

Spritet: Suurin osa Doomin jälkeisistä 2.5D peleistä käytti pelissä spritejä hyväkseen samalla tavalla, joten tässä osiossa tarkastellaan, miten sprite-grafiikat toteutettiin Doomissa.

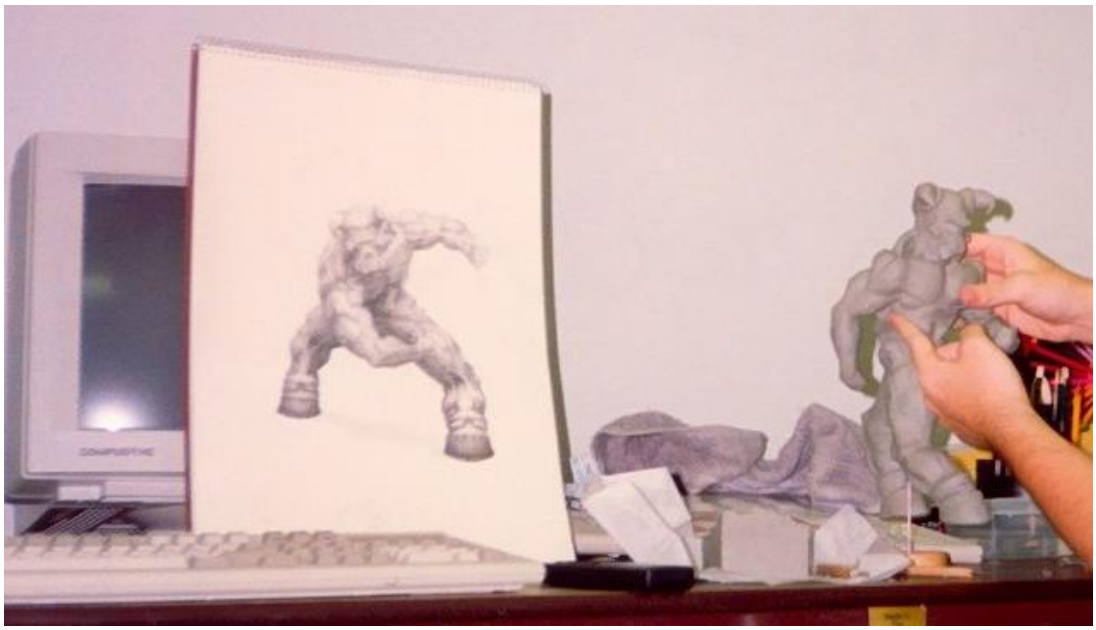
Yksinkertaisemmille spriteille, kuten maassa lojuville tavaroille ja aseille, ei ollut olemassa kuin yksi sprite. Tämän takia pelimaailmassa näitä tarkasteltaessa ne näyttivät aina samalta jokaisesta kulmasta, josta pelaaja niitä katsoi. Monimutkaisemmille asioille, kuten vihollisille ja moninpelissä toisille pelaajahahmoille, oli olemassa lukuisia spritejä.[18]

Viholliset: Pelissä vihollisten ja pelaajan välinen kulma laskettiin ja tulos jaettiin 45:llä, jolloin saatiin kahdeksan numeroa. Jokainen numero edusti 45 asteen kulmaa, jossa numero 1 edusti kulmaa, jossa pelaaja ja vihollinen katsoivat suoraan toisiaan. Tästä numeron kasvaessa kulma kääntyi aina 45 astetta myötäpäivään. Tällä tavoin saatiin luotua vaikutelma kolmiulotteisesta vihollisesta [18.]. Kuvassa 4 on havainnollistettuna, kuinka animaatiot on jaettu kahdeksaan eri kulmaan.



Kuva 4. Pelaajan ja vihollisen välisen kulman avulla valitaan animaatio viholliselle [18].

Poikkeuksiakin toki oli, sillä vihollisten kuollessa animaatio oli aina kohdistettu pelaajaa kohden, ja vihollisten ruumiit olivat yksittäisiä spritejä, eli ne olivat samannäköiset jokaisesta kulmasta tarkasteltuna. Osa pelin vihollisista piirrettiin käsin, mutta muutamista pelin sisältämistä vihollisista luotiin savesta veistos ja siitä otettiin eri kulmista kuvia, asettaen raajoja eri asentoihin, jotta veistos saataisiin animoitua pelissä. Tämän jälkeen kuvat digitalisoitiin ja siirrettiin peliin. [9; 21, kappale 8.] Kuvassa 5 Adrian Carmack muotoilee savesta vihollista vuoden 1993 Doom -peliä varten.



Kuva 5. Vahasta muovailtu vihollinen [22].

Aseet: Ensimmäisen persoonan ammuskelupeleissä on yleensä kahta eri metodilla toteutettua asetta. Kaikkein käytetyin on niin kutsuttu hitscan-hyökkäys, joka toimii ampumalla ray casting (säteensuuntaus suomeksi) avulla säde määrättyyn suuntaan. Säteen osuessa johonkin peli tarkastaa, mihin se osui, ja esimerkiksi tapauksessa, jossa pelaaja ampuisi vihollista, peli sen jälkeen katsoo vihollisen saaneen vahinkoa aseesta. Tämäntyyppistä hyökkäystä ei ole mahdollista väistää.

Toinen usein käytetty hyökkäystyyppi on projektiiliase. Hyökkäys toimii niin, että peliin luodaan objekti, joka matkustaa haluttuun suuntaan sille määrätyllä nopeudella. Osuessaan johonkin objekti suorittaa sille määrätyn efektin. Esimerkiksi tulipallon tapauksessa se räjähtää aiheuttaen vahinkoa.

Pelatessa kuvaruudulla näkyvät aseet toteutettiin nekin spriteillä ja animoitiin. Toteutustapa pelin aseissa oli hyvin samanlainen kuin pelin vihollisilla, jotka luotiin savesta ja kuvattiin. id Softwaren työntekijät ostivat läheisen marketin leluosastolta leikkipyssyjä, pitelivät niitä kädessään ja ottivat valokuvia. Valokuvat digitalisoitiin ja muunneltiin vastaamaan Doomin käyttämää väripalettia ja resoluutiota. Koska kuva oli digitaalisessa muodossa, siihen voitiin piirtää tarvittaessa lisää asioita. [9; 21, kappale 8.]

2.3 Vanhojen FPS-pelien kenttä- ja pelisuunnittelu

Kenttäsuunnittelu oli aiemmissa FPS-peleissä erilainen kuin nykyajan vastaavissa peleissä. Kentät olivat yleensä itsessään aina pientä pulmanratkontaa, kun eteneminen ei aina ollut täysin suoraviivaista, vaan kenttiin oli ripoteltu kekseliäitä salahuoneita ja reittejä, jotka saattoivat avata kentässä uuden tutkittavan reitin tai antaa pelaajalle jonkin uuden tehokkaamman aseen. Pelaajaa kannustettiin tällä tavoin tutkimaan kenttää ja koluamaan se läpikotaisin löytääkseen kaiken tarvittavan avun vihollisia vastaan. Aseita, ammuksia ja pelaajan terveyttä kohentavia terveyspakkauksia ripoteltiin kenttiin yleensä juuri sellainen määrä, että pelaaja selvisi niiden avulla eteenpäin, mutta ei niin paljoa, että peli olisi liian helppo. Tasapainon löytäminen vaatikin lukuisia iteraatioita ja vei paljon suunnittelijan aikaa.

John Romero myös käytti kenttiä suunnitellessaan ”hevosenkenkämallia”, kuten hän itse sitä kutsuu. Siinä pelaaja näki usein kentän alkupuolella määränpään, johon kuului mennä, mutta joutui sinne päästäkseen kiertämään muun kentän läpi ensin. Tämä teki

kentästä paljon mielenkiintoisemman, kuin jos pelaaja olisi vain joutunut kulkemaan suorassa linjassa loppuun asti. [17.]

Juonenkuljetus ei koskaan katkaissut nopeatempoista toimintaa, vaan juonta yleensä kuljetettiin eteenpäin ainoastaan kenttien välissä, joko tekstillä tai välianimaatiolla. id Softwaren kehittäessä Doom-peliä vuosien 1991 ja 1993 välillä, oli pelille kirjoitettu taustatarina ja yksityiskohtainen juoni jo valmiiksi, mutta näistä päätettiin luopua John Carmackin vastustaessa ideaa. Carmackin mielessä oli yksinkertaisempi ja toimintaa painottava peli. Hänen mielestään ”peleissä oletetaan olevan tarina, mutta se ei ole kovinkaan tärkeä”. Doomin julkaisua seuranneina vuosina valtaosa FPS-peleistä noudatti tätä ohjenuoraa. [10; 11; 21, kappale 8.]

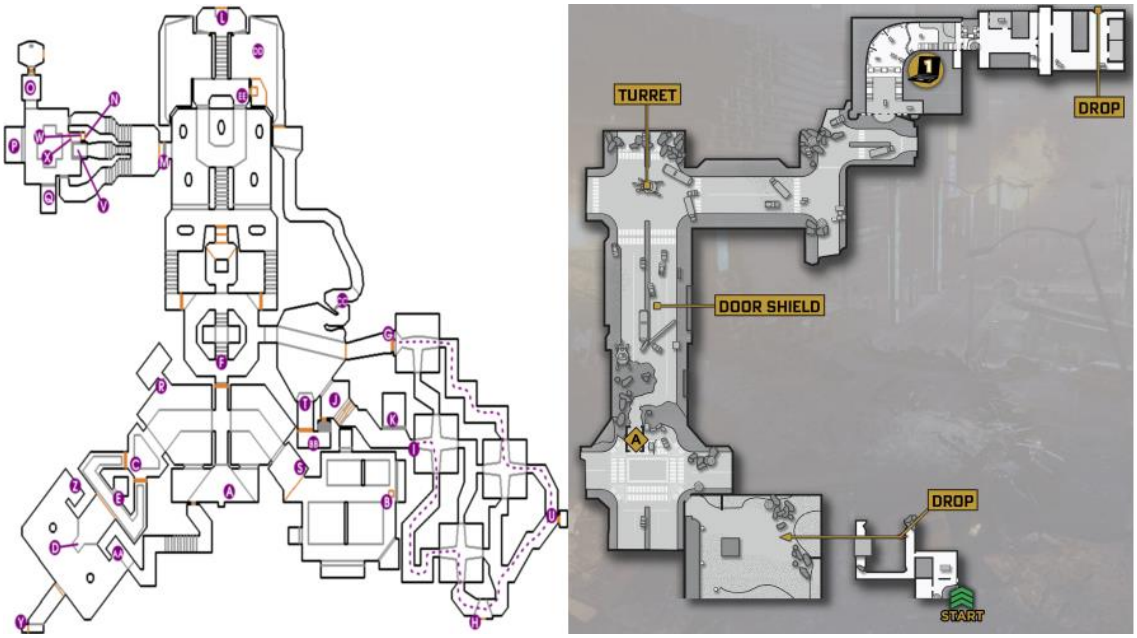
Doomissa pelimoottori oli kirjoitettu nopeus mielessä, ja id Softwaren jätettyä ylimääräiset juonielementit pois pelistä jäi jäljelle nopeatempoinen räiskintäpeli, jota vahvistettiin edelleen Robert ”Bobby” Princen säveltämiltä nopeilla trashmetallista vaikutteita ammentavilla Lo Fi midi -tulkinnoilla. [25.]

Jotkin aikaisista ensimmäisen persoonan ammuskelupeleistä sisälsivät lukuisia puzzleja eli eräänlaisia aivopähkinöitä, joita pelaaja joutui selvittämään edetäkseen pelissä tai päästäkseen alueelle, jonne ei muuten päässyt. Ensimmäisiä kertoja näitä nähtiin nopeatempoisissa ensimmäisen persoonan räiskintäpeleissä Raven Softwaren Doom Enginellä kehittämässä Hexen (1995) -pelissä.

2.4 Erot vanhojen ja nykyaikaisten FPS-pelien välillä

Kenties suurin ero nykyisten FPS-pelien yksinpelissä verrattuna edeltäjiinsä on kenttäsuunnittelu sekä juonen ja juonenkuljetuksen yhä isompi rooli uudemmissa peleissä. Kenttäsuunnittelu on muuttunut valtaosassa peleistä paljon suoraviivaisemmaksi. Kentissä edetään usein hyvin lineaarisesti alusta loppuun, eikä peli varsinaisesti rohkaise tutkimaan ympäristöä, vaan pikemminkin koettaa hillitä pelaajaa ohjaamalla häntä kädestä pitäen eteenpäin. Juonta edistetään kenttien aikana valmiiksi käsikirjoitettujen tapahtumien avulla, jolloin peli yleensä pysähtyy tapahtuman tai dialogin ajaksi, kun taas vanhemmissa FPS-peleissä hengähdystaukoja ei juuri kenttien aikana tarjottu, vaan vasta kentän lopussa toiminta pysähtyi hetkeksi ennen seuraavaa etappia. [14.]

Ensimmäinen pitkä askel tähän suuntaan nähtiin vuonna 1998, kun Half-Life ilmestyi. Half-Life tarjosi lineaarisempaa kenttärakennetta säilyttäen kuitenkin lukuisia eri salaisuuksia ja piilotettuja huoneita ja kannustaen pelaajaa tutkimaan matkallansa mahdollisimman paljon. Juonenkuljetusta painotettiin entistä enemmän, ja pelaajalle kerrottiin taustatarinaa ja edistettiin juonta kentän edetessä. Half-Life löysi sopivan tasapainon näiden kahden välillä, mutta tästä eteenpäin FPS-peleissä on hiljalleen näkynyt, kuinka kenttäsuunnittelu on muuttunut vuosi vuodelta yksinkertaisemmaksi välinäytösten ja juonenkuljetuksen ottaessa yhä vahvempaa otetta pelistä. Välillä jotkin moderneista FPS-peleistä tuntuvatkin siltä, kuin niissä edettäisiin ikään kuin raiteilla ja pelaaja on lähinnä matkustajan roolissa. Kuvassa 6 on verrattuna rinnakkain uutta ja vanhaa kenttäsuunnittelua. Vasemmalla vuoden 1993 Doom-pelistä E1M6-kenttä ja oikealla vuoden 2014 Call of Duty: Advanced Warfare -pelistä Induction-kenttä. [8; 14.]



Kuva 6. Vertausta uuden ja vanhan kenttäsuunnittelun välillä [26; 27].

Poikkeuksiakin toki on, sillä esimerkiksi vuonna 2016 julkaistu neljäs Doom -peli palasi juurilleen ja tarjosi pelaajalle paluuta monimutkaisempaan kenttäsuunnitteluun laittaen samalla juonen takasijalle ja nostamalla pelaamisen etusijalle. Pelin yksinpeli sai kovasti kehuja arvosteluissa ja näytti vanhanmalliselle FPS-pelille olevan vielä kysyntää markkinoilla. [7.]

2.5 Peli-idea

Olen pelannut paljon FPS-pelejä nuoruudessani, ja peli-idea, jonka pohjaksi prototyyppi tehtäisiin, syntyi eräänä päivänä kun rupesin miettimään, miksi kukaan ei enää tee spriteihin pohjautuvia FPS-pelejä, vaikka niille todennäköisesti olisi vähintäänkin pieni markkinarako olemassa vanhojen FPS-pelaajien keskuudessa. Mielipidettäni vahvisti vielä entisestään vähän aikaa sitten (2014) julkaistun Strife: Veteran Editionin (alun perin vuonna 1996 julkaistun id Tech 1 -pelimoottoria käyttävän Strife-pelin päivitetty uudelleenjulkaisu) suhteellisen hyvät myyntiluvut (noin 65 000 myytyä kopiota Steamin kautta) ja silloin suuressa suosiossa ollut Brutal Doom -modi (ohjelmiston muokkaaminen uusien ominaisuuksien lisäämiseksi) alkuperäiselle Doomille. Se lisää peliin lähinnä uusia kosmeettisia asioita, kuten uusia efektejä animaatioita, ja aseita. [16.]

Tästä rohkaistuneena mietin, miksen yrittäisi itse tehdä omaa retro-FPS-peliä, vaikka minulla ei ollut aikaisempaa kokemusta pelinkehittämisestä, 3D-mallintamisesta, spritejen piirtämisestä tai äänien ja musiikin tekemisestä.

Koska ideani kumpusi vanhoista FPS-peleistä, oli pelini määrä mukailla esikuviaan graafisesti, pelillisesti ja myös kenttäsuunnittelun kannalta. Halusin, että pelissä olisi myös ripoteltu hieman ongelmanratkontaa ja tasohyppelyä, jotta peliä saataisiin vähän rytmitettyä eikä pelaaja tuudittautuisi pelkkään toiminnalliseen räiskintään.

Ottaen huomioon, että peli olisi kuitenkin toimintapeli, en halunnut, että ongelmanratkonta tai tasohyppelyosuudet kuitenkaan kokonaan pysäyttäisivät pelin etenemistä, joten näiden osuuksien pitäisi olla tarpeeksi haastavia, mutta ei liian rankaisevia, mikäli pelaaja ei niitä läpäisisi.

Pelin viholliset, ja objektit toteutettaisiin käyttäen hyödyksi spritejä, sillä koin taidottomuuteni 3D-mallintamisessa tekevän työstä erittäin raskasta ja hidasta, kun taas 2D-spritejen piirtäminen alhaisilla resoluutioilla on siihen verrattuna helpompaa toteuttaa, joskin työlästä tarvittavien spritejen määrän takia.

Pelin alkuasetelmaksi ajattelin seuraavaa: Muinainen pahuus olisi hiljalleen yhdistämässä aika-avaruutta ja vaihtoehtoisia muita todellisuuksia yhteen pisteeseen singulariteetiksi, ja pelaajan saapuessa paikalle olisivat välimatkat, aika ja todellisuus vääristyneet jo niin paljon, että muinaisen pahuuden linnan lisäksi olisi neljä pelattavaa aluetta, jotka eroaisivat temaattisesti toisistaan.

Tärkeää tässä ei olisi itse taustatarina, vaan se, kuinka sen avulla olisi helppo oikeuttaa eri teemojen käyttö pelin kentissä, kuten vaikkapa western-, fantasia- ja sci-fi-teemat, eikä se vaikuttaisi pelimaailmassa oudolta.

Peli olisi rakenteeltaan hubimainen, eli pelissä olisi eräänlainen solmukohta, josta pääsisi etenemään halutussa järjestyksessä kenttäkokonaisuuksia palaten aina solmuohtaan takaisin. Keskiössä hubina toimisi muinaisen pahuuden linna. Itse viimeiselle loppuvastukselle johtava ovi olisi lukossa, ja avatakseen oven pelaajan tulisi ensin selvittää neljä eriteemaista kappaletta, jotka voitaisiin valita vapaassa järjestyksessä pelin hubista käsin ja jotka koostuisivat useammasta kentästä, jotka käytäisiin läpi lineaarisesti, ennen kuin pelaaja palaisi viimeisen kentän jälkeen takaisin keskusmaailmana toimivaan hubiin. Jokaisen kappaleen lopussa pelaaja kohtaisi yhden muinaisen pahuuden lakeijoista, joilla jokaisella olisi hallussaan yksi avaimen osa, jolla keskusmaailmassa sijaitseva viimeinen ovi avattaisiin.

Jotta järjestys, jossa kentät halutessaan voisi mennä läpi, ei olisi samantekevä, eri kentistä saisi teemaansa sopivasti erinäisiä aseita, jotka tekisivät erityyppistä vahinkoa (esimerkiksi läpäisevää vahinkoa tai tulivahinkoa). Jotkin viholliset pelissä olisivat kestävämpiä yhtä vahinkotyyppiä vastaan ja heikompia toista vastaan. Samalla pelaaja saisi syyn käyttää monipuolisesti aseita kohtaamiensa vihollistyyppien mukaan eikä ainoastaan sen takia, mitä ammustyyppiä olisi milloinkin tarjolla kentissä.

3 Pelin prototyyppi

3.1 Prototyypin tavoitteet

Insinöörityön päätavoitteena oli tehdä onnistuneesti ensimmäisen persoonan ammuskelupelin prototyyppi, joka tuntuu ja näyttää samalta, kuin 1990 -luvun esikuvansa, niin visuaalisesti kuin suunnittelunkin kannalta. Prototyypin oli myös onnistuttava luomaan perusta, josta omaa peliä on mahdollista jatkokehittää eteenpäin, mikäli prototyyppi vaikuttaa lupaavalta.

Prototyypissä oli hyvä olla ainakin pari kenttää, jotta voi testata, että pelaajahahmon tallennetut tiedot (terveys, ammustilanne, kerätyt aseet) siirtyvät seuraavaan kenttään onnistuneesti.

Aseita tuli olla vähintään kaksi erilaista, yksi hitscan-ase ja yksi projektiiliase, jotta näille eri hyökkäystyypeille oli valmiiksi toteutus jatkoa ajatellen. Myös jonkinlainen lähitaisteluase oli hyvä toteuttaa prototyypissä.

Koska peli toteutettiin käyttäen spritejä kuvastamaan vihollisia ja useimpia objekteja pelimaailmassa, oli prototyypin tärkein tavoite saada aikaiseksi vihollisille toimiva skripti, joka kohdistaa aina vihollista kuvastavan kaksiulotteisen spriten pelaajaa kohti ja laskee kulmien avulla oikean animaation viholliselle, kun pelaaja kiertää tätä ympäri. Viholliselle tuli olla myös jokaista ilmansuuntaa ja väli-ilmansuuntaa varten animaatiot, jotta toteutusta voidaan arvioida.

Prototyyppi toteutettiin käyttäen apuna pelkästään itse tehtyjä komponentteja, jotta pelin kehitystä mahdollisesti jatkettaessa kaikki oikeudet materiaaleihin olisivat itsellä.

Mikä tekee pelistä retron

Retro-sana itsessään on määritelty esimerkiksi Cambridgen yliopiston sanakirjassa asiaksi, joka on samankaltainen kuin menneisyyden esikuvansa. Tässä tapauksessa siis ollakseen retro pelin tuli jäljitellä esikuvansa tyyliä niin, että vanhan pelin tyylin pitäisi olla heti tunnistettavissa. Koska tässä projektissa oltiin toteuttamassa 1990-luvun FPS-pelejä mukailevaa retropeliä, alla on listattuna, millä tavoin tätä koetettiin tavoitella.

- Viholliset ja tavarat 2D-spriteinä.
- Koetettiin pysyä esikuville uskollisena kenttäsuunnittelussa, eli moniulotteisia karttoja, jotka eivät ole pelkästään suoria putkia pisteestä A pisteeseen B.
- Koetettiin käyttää pientä resoluutiota spriteissä ja tekstuureissa.
- Pyrittiin jäljittelemään samanlaiset toiminnallisuudet peliin kuin esikuvissakin oli.

3.2 Prototyyppiä varten tarvittavat komponentit

3D-mallit testikentälle ja pelin kaksi ensimmäistä kenttää: Testikenttä helpottaa peliin toteutettavien ominaisuuksien testaamista huomattavasti, ja siellä kannattaa rakentaa valmiiksi pelin tärkeimmät ominaisuudet, ennen kuin alkaa ensimmäistä kenttää

suunnitella. Kenttiä suunnitellessa pitää muistaa tehdä niistä hieman epälineaarisia, jotta pelaajan mielenkiinto pysyy yllä ja yrittää antaa jokaiselle kentän sisältämälle jokin tarkoitus, miksi pelaaja sinne menisi.

Kentät toteutetaan 3D-malleina, sillä nykyajan 3D-mallinnusohjelmistoilla niistä saa helposti tehtyä juuri sellaisia, kuin itse haluaa, ja nykyajan pelimoottorit ja tietokoneet eivät tämän kokoluokan pelissä tarvitse graafisia kikkailuja toimiakseen nopeasti.

Pelaajakontrolleri: Jotta maailmassa voi liikkua, tarvitaan pelimaailmaan objekti edustamaan pelaajahahmoa. Hahmoon tulee olla kiinnitettynä pelin pääkamera, sillä peliä kuvataan ensimmäisestä persoonasta ja liikkuminen hoidetaan pelaajakontrolleri-skriptin kautta käyttäen syötteinä näppäimistöä ja hiirtä.

Viholliset: Vihollisia varten tarvitaan useita eri komponentteja. Viholliset tarvitsevat vähintään jonkin alkeellisen tekoälyn, mutta myös näkymättömiä välietappeja (waypoint), jonka avulla ne voivat liikkua ympäriinsä pelimaailmassa. Vihollisia varten tarvitaan myös spritejä animaatiota varten. Spritejä tarvitaan vähintään kaksi jokaista kahdeksaa ilmansuuntaa kohden. Hyökkäys- ja kuolinanimaation voi toteuttaa, jos aikaa jää yli, mutta ne eivät ole projektin kannalta tarpeellisia

Käyttöliittymä: Alkuvalikko on hyvä olla, mutta se ei ole prioriteettilistalla kovinkaan korkealla. Tärkeämpiä ovat pelin aikaiset UI-elementit, kuten terveysmittari, ammusmittari, pelaajan ase ja aseen animaatiot (ampuminen, aseen heilahtelu pelaajan liikkuessa, jne.). Kenttien välillä olevaa lopetusruutua varten pitää kentän alussa laskea viholliset, löydetyt salaisuudet ja aika, jotta kenttien välissä voidaan antaa pelaajalle tietoa siitä, miten kenttä meni.

Sekalaiset: Tänne tulevat kaikki pienemmät toteutettavat asiat, jotka ovat kuitenkin prototyypin toteutuksen kannalta tarpeellisia, kuten valaistus, ovet, mahdolliset hissit, sekä liikkuvat tasot, vipuja, näkymättömiä triggereitä, tavaroita (ammuslippaat, terveyspakkaukset, kerättävät avaimet) sekä pelaajan aloituspaikka.

Tekstuurit ja spritet: Kuten aikaisemmin vihollisten kohdalla mainittiin, tarvitaan vihollisille lukuisia spritejä, jotta ne saadaan animoitua. Mutta niiden lisäksi myös pelin tavarat, aseet ja mahdolliset lamput, kynttilät, napit, vivut, jne. tarvitsevat spritejä. Tekstuureja tarvitaan 3D-mallien seinien, ovien, lattioiden ja katon pintoihin.

Äänet: Äänet ja musiikki ovat olleet suuri tekijä FPS-peleissä tunnelman ja pelaajakokemuksen luomisen suhteen. Mutta jottei projekti paisuisi liian isoksi, päätin olla toteuttamatta musiikkia omassa prototyypissäni, mutta tekeväni perusasioille äänet (esimerkiksi: ampuminen, vahingon ottaminen, ovien avaus).

3.3 Toteutukseen tarvittavat työkalut

Tämäntyyppisen prototyypin voi luoda monella eri ohjelmalla, mutta asetin kriteeriksi sen, että käytettävä pelimoottori olisi mahdollisimman käyttäjäystävällinen ja ilmainen. Koska projektissa tulee käyttää vain omia komponentteja, tarvitsi pelimoottorin lisäksi myös keinon tehdä 3D-malleja, 2D-grafiikkaa ja ohjelman, jolla äänittää ääniefektejä peliin.

Pelimoottori: Pelimoottoria valitessa valintaprosessi johti valintaan Unreal Enginen ja Unity3D:n välillä. Lopulta päädyttiin valitsemaan alustaksi Unity3D, sillä sille on tarjolla uusille käyttäjille isot määrät tutoriaaleja alkuun pääsemisen helpottamiseksi, ja sen ilmaisen version katsottiin riittävän mainiosti saavuttamaan prototyypin tavoitteet siitä huolimatta, että osa Unityn ominaisuuksista on lukittu lisenssin taakse. Unityssä on myös haluttaessa mahdollista käyttää Unityn omaa Asset Storea, jonka kautta voi ostaa tarvittavia komponentteja ja toiminnallisuuksia ilman, että ne pitäisi itse tehdä.

3D-mallinnus: 3D-mallinnuksessa käytettäväksi ohjelmaksi valikoitui Blender. Sen puolestapuhujiin kuului se, että se on ilmainen, sillä on vankka käyttäjäkunta, sitä päivitetään tasaisin väliajoin ja siihen on helppo löytää apua sekä tutoriaaleja tarvittaessa. Myös 3D-mallit, joita prototyyppi tarvitsee, ovat sen verran yksinkertaisia, että ne onnistuvat Blenderillä helposti ilman valtavaa tietotaitoa. Blenderistä on myös helppo tuoda Unityssä sijaitsevaan peliprojektiin suoraan tiedostoja, sillä ne päivittyvät reaaliaikaisesti komponentteihin, kunhan käyttää jotain muuta tiedostomuotoa kuin Blenderin omaa ”.blend”-tiedostoa. Tässä projektissa käytettiin tiedostomuotona ”.obj”-formaattia.

Tekstuurit: Tekstuureja ja spritejä varten työkaluksi valittiin aikaisempien kokemuksieni perusteella GIMP-ohjelmisto. Se on yksinkertainen, selkeä ja sillä saa helposti vietyä eri tiedostomuotona kuvatiedostot haluttuun paikkaan. Myös GIMP-ohjelmistolla saa suoraan vietyä tiedostot Unityyn, kunhan muistaa vaihtaa tiedostomuodon Gimpin omasta ”.xcf”-muodosta toiseen. Tämän projektin aikana tiedostot tallennettiin ”.png”-formaattiin.

Äänet: Ääniä varten tarvittiin ohjelma, jolla nauhoittaa ja miksata ääniä peliä varten. Tähän tarkoitukseen löytyi ilmaisten ohjelmien seasta Audacity-sovellus, jolla saisi pienellä vaivalla riittävän hyviä ääniä aikaiseksi.

4 Prototyypin toteutus

4.1 Prototyypin komponentit

4.1.1 Pelaajakontrolleri

Insinöörityössä tehty pelaajahahmo toteutettiin kiinnittämällä tyhjään objektiin Unityn oma character controller -luokka, oma pelaajakontrolleri-Scripti ja myöhemmin rigidbody (rigidbody mahdollistaa objektin liikuttamisen Unityn fysiikkamoottorin kautta) [19.] niitä hetkiä varten, kun pelaajahahmo pitää asettaa kinemaattiseksi (liikkuvien tasojen päällä ollessaan). Character Controller -luokan avulla voidaan helposti ohjata pelihahmoa syöttämällä PlayerController-skriptin kautta suunta ja nopeus Character Controllerille ilman, että pelaajahahmoa tarvitsee erikseen liikutella rigidbodyn avustuksella. PlayerController-skripti ottaa vastaan näppäimistöltä syötteitä ja antaa niiden perusteella tietoa eteenpäin Character Controllerille.

Pelaajahahmon objektiin on myös kiinnitettynä pelin pääkamera ja Weaponswitcher-skripti, jonka kautta tieto etenee pelin käyttöliittymälle sillä hetkellä käytössä olevista aseista, jotka sitten animoidaan kuvaruudulla.

Pääkameraa ohjataan PlayerController-skriptin kautta, joka tulkitsee hiiren liikkeitä pysty- ja vaakasuunnassa ja kääntää kameran oikeaan suuntaan pelimaailmassa. Kameralla on oletusasento 90 asteen kulmassa, ja skripti rajoittaa kameran kääntämistä niin, ettei pelaaja voi kääntää päätä oletusasennosta 90:tä astetta enempää ylös tai alaspäin.

Pelaajakontrolleriin on myös varastoituna pelaajan tietoja, kuten senhetkinen terveystila, ammukset, käytössä oleva ase, hiiren herkkyys, liikkumisnopeus, hyppynopeus ja aseiden cooldownit (odotusaika, ennen kuin pelaaja pystyy laukaisemaan aseen uudelleen).

Myös aseiden funktiot löytyvät pelaajakontrollerin alta, ja pelaajan painaessa hyökkäysnappia tulkitsee skripti syötteen ja lähettää sillä hetkellä aktiivisena olevan aseen funktiolle hyökkäyskäskyn.

4.1.2 Viholliset ja tekoäly

Viholliset koostuvat kahdesta eri objektista. Itse vihollisobjekti, jossa on rigidbody, boxcollider (näkymätön laatikko, joka toimii tunnistamalla siihen kohdistuvat törmäykset), vihollisen skripti ja äänilähde, jotta voidaan vihollishahmon sijainnin kohdalta soittaa ääniefektejä, sekä vihollisobjektista periytyvä cultistSprite-objekti, jossa on vihollisten 2D-spritet aina pelaajaa kohti kääntävä skripti ja vihollisen animaatiot.

Pelin vihollisille toteutettiin yksinkertainen tekoäly. Viholliset ovat ensisijaisesti joko aktiivisia tai passiivisia. Vihollisilla on 180 asteen näkökenttä eteenpäin, ja passiivisena vihollinen jatkuvasti piirtää säteen (ray cast) itsestään pelaajahahmoa kohti ja määrittelee sen perusteella, onko pelaajan ja vihollisen välissä jokin näköeste. Viholliset seisovat paikallaan odottaen, että pelaajahahmo joko kävelee näkökenttään tai pelaaja esimerkiksi laukaisee aseensa samassa huoneessa, jolloin viholliset reagoivat ärsykkeeseen ja aktivoituvat.

Ollessaan aktiivisia viholliset edelleen katsovat, onko pelaajaan näköyhteys, ja jos pelaajan ja vihollisen välissä ei ole mitään estettä, vihollinen lähtee etenemään suoraan pelaajaa kohti. Samaan aikaan vihollinen myös hyökkää, mikäli sattuu olemaan hyökkäysetäisyydellä. Vihollisilla on oma minimietäisyytensä, jolloin ne pysähtyvät pelaajan eteen ja jatkavat hyökkäämistä. Aktiivisena ollessaan ja kadottaessaan näköyhteyden pelaajaan viholliset lähtevät vaeltamaan kohti lähintä välietappia (waypoint). Saavutettuaan välietapin skripti katsoo, mitkä ovat lähimmät välietapit, jotka on linkitetty juuri saavutettuun välietappiin, ja arpoo näistä yhden, jonne jatkaa matkaa. Välietapit on manuaalisesti sijoitettu kenttään, jotta viholliset osaisivat liikkua pelimaailmassa osumatta seiniin.

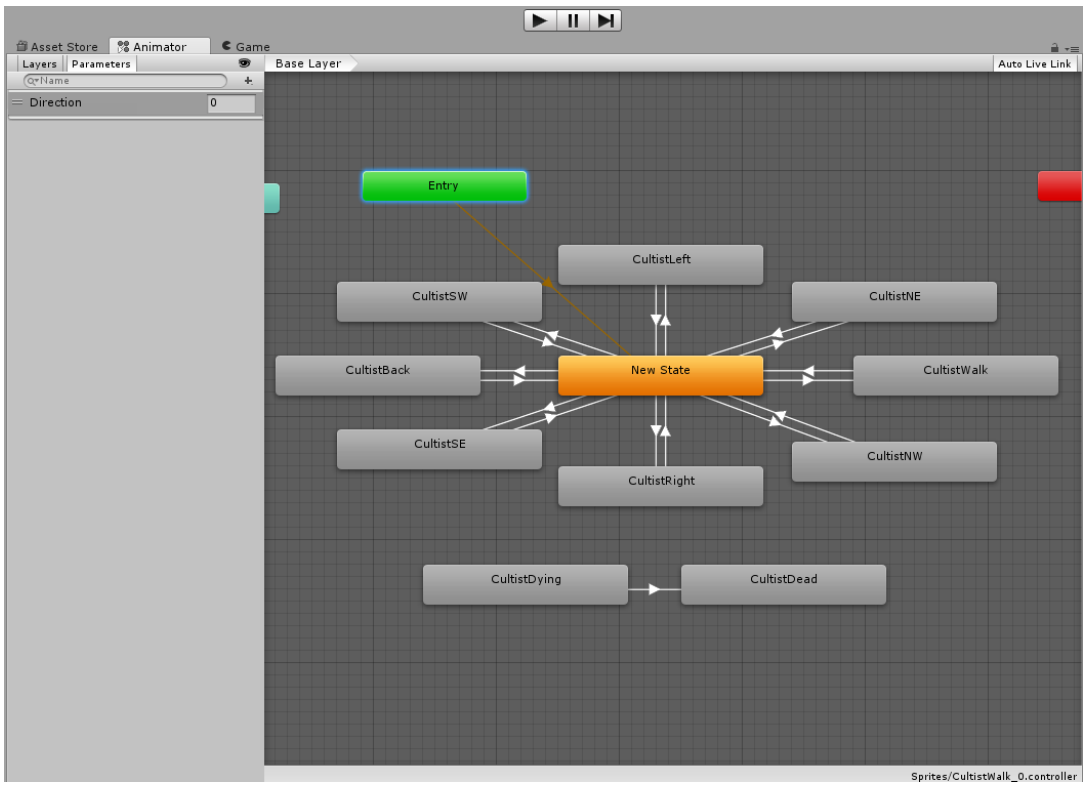
Mikäli vihollinen saa taas näköyhteyden pelaajaan, ne käyvät aggressiivisiksi ja kävelevät pelaajaa kohti samalla hyökäten.

Vihollisia on pelissä kahta eri vihollistyyppiä: pitkältä matkalta projektiileja ampuvia kultisteja ja lähelle pyrkiviä ja lähitaistelussa vahinkoa tekeviä varjo-olentoja.

Kultisti: Pelin ensimmäinen vihollinen on tulipalloja heittelevä kultisti. Kultisti on melko heikko tuliaseita vastaan, mutta kestää tulivahinkoa hieman muita vihollisia paremmin. Kultisti on melko hidas, mutta sillä on hieman enemmän terveyspisteitä kuin pelin toisella vihollisella. Kultistin hyökkäyksenä toimiva tulipallo on toteutettu projektiili. Hyökätessään kultisti luo eteensä tulipallon ja antaa sille eteenpäin suuntaavan voiman, jonka avulla tulipallo liitää ilman halki siihen suuntaan, missä pelaaja oli hyökkäyksen alkaessa. Tämä hyökkäys on kuitenkin väistettävissä, sillä tulipallolla menee aikaa saavuttaa kohteensa.

Varjo-olento: Varjo-olento on pelin toinen vihollinen, jonka hyökkäys toimii vain lyhyeltä kantamalta, minkä takia varjo-olento pyrkii pääsemään mahdollisimman lähelle pelaajaa. Varjo-olento on paljon nopeampi kuin Kultisti ja tekee enemmän vahinkoa, mutta sillä on paljon vähemmän terveyspisteitä. Varjo-olento myös ottaa vähemmän vahinkoa pistoolista, mutta on heikko tulipalloja vastaan.

Graafinen toteutus: Viholliset toteutettin vanhoille FPS peleille uskollisesti spriteinä eli tekemällä sarjan 2D-animaatioita, jotka olivat aina kohdistettuna pelaajaa kohden. Pyörivä animaatio määriteltiin laskemalla kulma sen mukaan, mihin suuntaan pelaaja katsoo, ja sitä verrattiin vihollisen katsomasuuntaan. Tästä saatu kulma jaetaan 45:llä, jolloin 360 asteesta saadaan 8 kokonaislukua. Yksi kokonaisluku vastaa 45 asteen kulmaa, ja se syötetään suoraan Unityn omaan animaattoriin, jossa on erikseen tehtynä ennalta viholliselle erilaiset suunta-animaatiot, joista kukin vastaa yhtä kokonaislukua väliltä 0–7. Kuvassa 7 on näkymä Unity3D:n animator-välilehdeltä, jossa näkyvillä ovat kultistin animaatiot ja niiden keskellä sijaitseva tyhjä animaatio, jonka kautta siirrytään seuraavaan animaatioon.



Kuva 7. Unity3D:n animator-välilehden näkymä.

Kultistille piirrettiin ja animoitiin kaksi erilaista spriteä jokaista kahdeksaa kulmaa kohti. Spritejä tuli kokonaisuudessaan kävelyanimaatioille 16. Kuolinanimaatio vei 4 spriteä, joten yhteensä kultistin animaatioihin meni 20 spriteä. Kuvassa 8 ovat esiteltynä kultistin kuolinanimaation kaikki neljä spriteä.

Kuva 8. Vasemmalta oikealle kultistin kuolinanimaatio, johon meni 4 spriteä.

4.1.3 Aseet ja tavarat

Aseet toteutetaan sekä pelaajakontrollerissa että pelin käyttöliittymässä. Pelaajakontrollerissa lasketaan kaikki hyökkäykseen liittyvä, ja käyttöliittymään animoidaan tilanteeseen sopiva animaatio hyökkäyksen tapahtuessa. Käyttöliittymässä aseisiin on myös lisätty skripti, joka animoi niitä liikkumaan hieman pysty- ja vaakasuunnassa pelaajan liikkuessa eteenpäin, mikä luo illuusion liikkeestä.

Kaikki pelin aseet toteutettiin ottamalla valokuva pitämällä eri objekteja kädessä, minkä jälkeen valokuvat prosessoitiin GIMP-ohjelmiston läpi. Ensin poistettiin halutun grafiikan ympäriltä kaikki ylimääräinen, minkä jälkeen kuvan resoluutiota alennettiin, ja sitten muutin se käyttämään Doomin rajallisempaa väripalettia. Tämän jälkeen muutettiin vielä hieman valotusta, jotta korkeat kohdat erottuisivat paremmin varjossa olevista yksityiskohdista.

Sakset: Sakset ovat lisähyökkäys, jota voi käyttää silloin, kun pelaajalla on vain yksi käsi käytössä (esimerkiksi pistoolia käyttäessä). Ase toimii hitscannina (hyökkäys toimii ray castilla) lyhyeltä matkalta ja tekee vain vähän vahinkoa, mutta samaan aikaan on mahdollista ampua pistoolilla. Kuvassa 9 on saksihyökkäyksen animaatio.

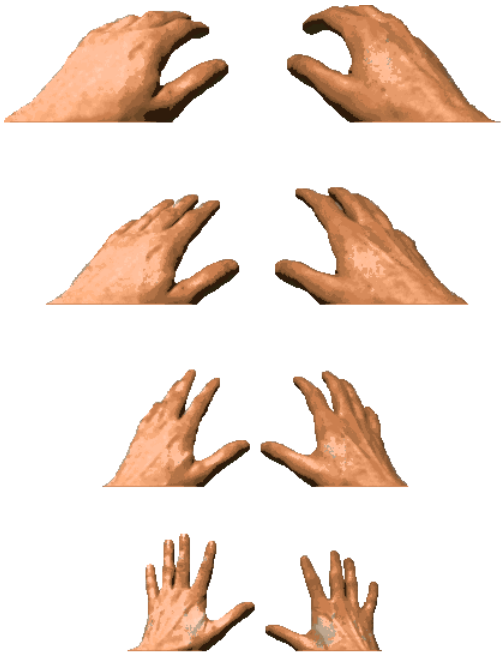
Kuva 9. Saksihyökkäyksen animaatio.

Pistooli: Pistooli on pelin ensimmäinen ase, joka ampumalla säteen (ray cast) suoraan pääkameran keskikohdasta eteenpäin määrittää, osuuko säde vastustajaan. Esikuvilleen uskollisena pistoolilla ei ole erikseen lipaskokoa, eikä sitä näin ollen tarvitse koskaan ladata, vaan sillä voi ampua niin kauan, kuin panoksia riittää. Pistoolilla on kuitenkin

pieni viive aina, ennen kuin sillä voi ampua uudelleen. Kuvassa 10 ovat pistoolin animaation käyttämät spritet.

Kuva 10. Pistoolin animaatio.

Tulipallo: Tulipallo on pelin ensimmäinen saatavilla oleva uusi ase, ja se on projektiili. Pelaaja ampuu käsistään tulipallon, joka etenee pelimaailmassa, kunnes osuu johonkin kiinteään objektiin. Osuessaan tulipallo aiheuttaa vahinkoa pienelle alueelle ympärillään osumahetkellä. Kuten pistooli, myös tulipallo lentää keskeltä kameraa suoraan eteenpäin, mutta siinä missä pistoolin ampuminen toimii säteen avulla, generoidaan tulipallo-objekti pelaajan eteen, ja annetaan sille eteenpäin suuntaava voima. Kuvassa 11 on animaatio tulipallohyökkäykseen, jossa kuvaruudulla näkyvät kädet taikovat tulipallon.

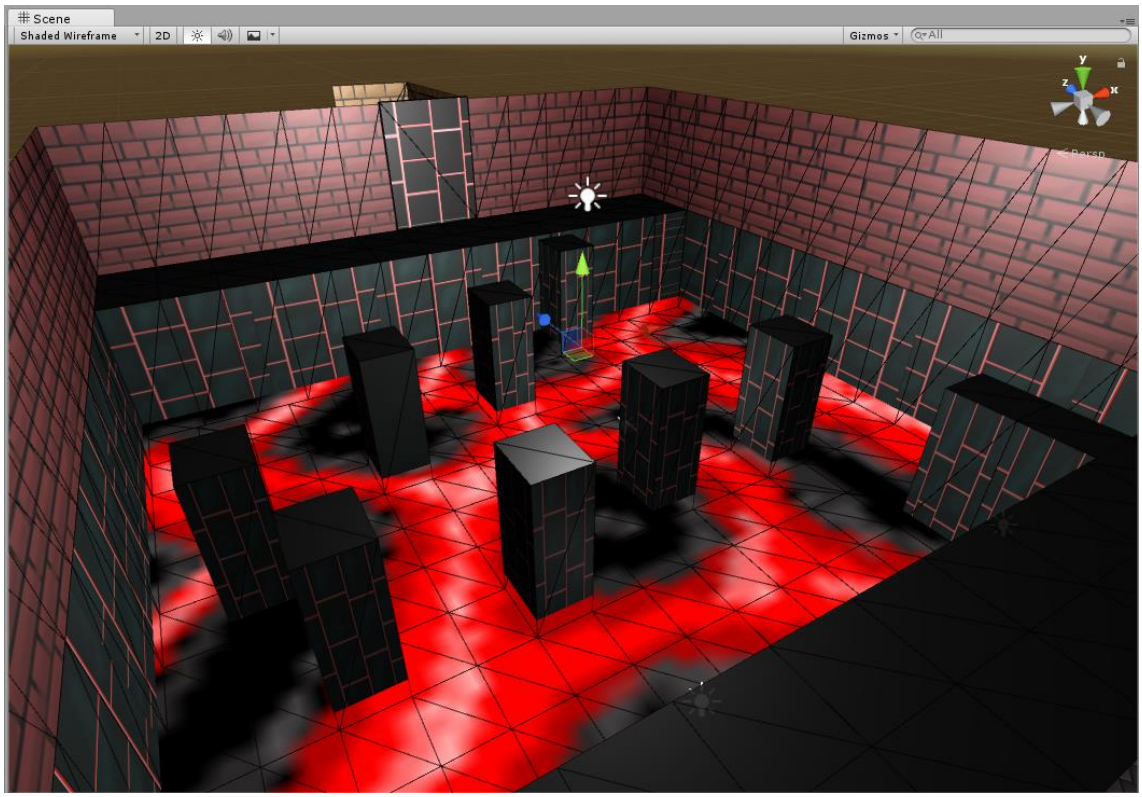


Kuva 11. Tulipallotaian animaatio.

4.1.4 Puzzlet

Ensimmäiseen kenttään tehtiin kaksi puzzlea, joista yksi oli vapaaehtoinen ja piilotettu salaisen käytävän taakse, ja toinen toimii pienenä aivopähkinänä ennen ensimmäisen kentän loppua.

Piilotettu puzzle: Salahuoneessa sijaitsevaan puzzleen lisättiin hieman tasohyppelyä. Kyseessä on pilkkopimeä suuri huone, jonka keskellä on kuilu ja alhaalla laavaa, jossa seistessään pelaaja hitaasti menettää terveyttä. Kuilussa on pilareita lattian tasolle asti, mutta ne ovat piilossa, sillä huone on pilkkopimeä. Kuvassa 12 näkyy ensimmäisen kentän salahuone, jossa valot ovat päällä.



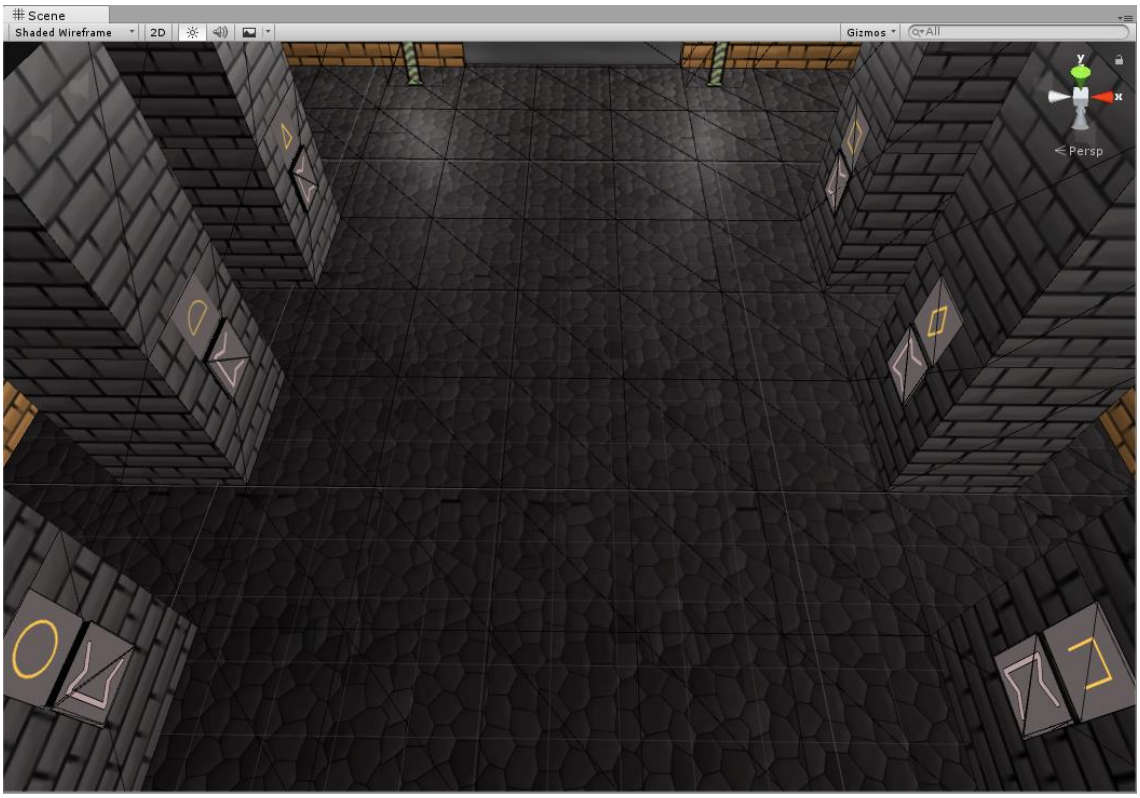
Kuva 12. Ensimmäisen kentän salahuone.

Katossa on himmeitä valoja, joita on jokaisella ruudulta huoneessa, josta löytyy lattiatasolla oleva pinta. Huoneen alussa on pelaajia varten jätetty viesti, joka neuvoo seuraamaan tähtiä, mikäli kadottaa tiensä. Sisääntulon puoleisella kuilun seinämällä on kummassakin kulmassa hissi, josta pelaaja pääsee ylös, mikäli sattuu putoamaan alas pyl-

väiltä. Mikäli pelaaja pääsee loppuun asti, on toisella puolella triggeri, joka poistaa huoneesta pimeyden ja aktivoi valolähteen samalla avaten oven, jonka takana on pelin ensimmäinen saatavilla oleva lisäase, tulipallo.

Ensimmäisen kentän puzzle: Ensimmäisen kentän lopussa on pientä ongelmanratkontaa kuuden napin muodossa, jossa kunkin napin yllä on erimuotoinen kuvio. Ideana on, että oikeat napit ovat ne, joiden kuviot löytyvät neljästä aikaisemmasta huoneesta, joihin pelaajalla on ollut pääsy ennen saapumista viimeiseen huoneeseen. Kuviot ovat myös samanmuotoisia kuin huoneet, joista ne löytyvät.

Mikäli pelaaja painaa onnistuneesti oikeita nappeja, avautuu ovi kentän lopettavalle napille. Mikäli pelaaja painaa väärää nappia, avautuvat viimeisessä huoneessa olevat piilohuoneet, joista tulee ulos uusia vihollisia. Pelaajaa ei kuitenkaan rankaista sen enempää väärästä vastauksesta, vaan hänen on mahdollista painaa kaikki napit pohjaan, jolloin ovi viimein aukeaa. Puzzle toteutettiin tällä tavalla, koska ei haluttu, että pelaaja turhautuu, mikäli peli pysähtyisi pidemmäksi aikaa tähän huoneeseen, vaan pelaaja pääsee jatkamaan eteenpäin ilman suurempaa hidastetta, mikäli niin haluaa. Kuvassa 13 näkyy ensimmäisen kentän viimeinen puzzle, jonka avulla kentän loppuun johtava ovi avataan.



Kuva 13. Ensimmäisen kentän viimeinen puzzle.

Toisen kentän ”puzzle”: Toisessa kentässä oleva salahuone ei niinkään kätke puzzlea taakseen, vaan ennemminkin tasohyppelyä vaativan osuuden, jossa pelaajan on päästävä ykköskentästä tutun näköisen kuilun yli vaakatasossa liikkuvia tasoja pitkin. Toisella puolella taas on näkymätön triggeri paikoillaan, ja pelaajan päästessä toiselle puolelle aukeaa lukittu ovi.

Alun perin tarkoitus oli tehdä tästä huoneesta samantyyppinen kuin ensimmäisen kentän salahuoneesta, mutta koska ei saatu aikarajojen puitteissa pelin liikkuvia tasoja toimimaan toivotulla tavalla, vaan pelaaja ei täysin pysynyt niiden kyydissä ollessaan paikallaan, päätettiin luopua ideasta. Ylitse pääseminen oli jo tarpeeksi vaikeaa, ilman että pelaajan pitää turhautuneena ihmetellä pilkkopimeässä, miksi hän ei pääse kuilun yli. Mikäli peliä jatkokehitetään, Liikkuvat tasot viilataan sellaiseen kuntoon, että voidaan joko kakkoskentässä tai sitä seuraavassa toteuttaa alkuperäinen idea puzzlelle.

Liikkuvat tasot: Liikkuvat tasot toteutettiin luomalla peliin tyhjä objekti, johon liitettiin neliönmuotoinen objekti (platform) ja kaksi tyhjää objektia (startPosition ja endPosition). Platform-objektissa ovat mesh render, box collider, rigidbody ja block controller -skripti, ja platform-objektin alla on periytyvänä vielä yksi objekti, jolla on triggerinä toimiva box collider ja hold player -skripti. StartPosition ja endPosition -objekteja käytettiin merkitsemään sijainteja, joiden välillä taso liikkuu.

Skripti tunnistaa tagin perusteella, onko triggerin sisälle tullut objekti pelaajahahmo, ja asettaa pelihahmon rigidbody-komponentissa kinemaattisuuden päälle ja asettaa pelaajahahmon aliluokakseen (set as parent), jolloin pelaaja perii liikkuvalta tasolta sijainnin eikä taso vain liu’u pelaajan alta. Pelaajan liikkuessa pois triggerin vaikutusalueelta poistetaan pelaajahahmolta rigidbodystä taas kinemaattisuus ja periytyvyys liikkuvaan tasoon, jolloin pelaajahahmo voi vapaasti liikkua ilman ulkoisia vaikutteita omaan sijaintiinsa.

StartPosition- ja endPosition-objektit piirretään ruudulle editorinäkymässä käyttämällä hyväksi Unityn omaa Gizmos-luokkaa. Gizmos-luokan avulla saadaan editorinäkymässä piirrettyä liikkuvan tason ulottuvuudet omaavat laatikon ääriviivat, jotka värjätään, jotta voi erottaa, kumpi on aloitussijainti (vihreä) ja kumpi on lopetussijainti (punainen). Kuvassa 14 ovat toisen kentän salahuone ja liikkuvat tasot editorinäkymästä nähtynä. Tasot liikkuvat punaisten ja vihreiden laatikoiden välissä horisontaalisesti.

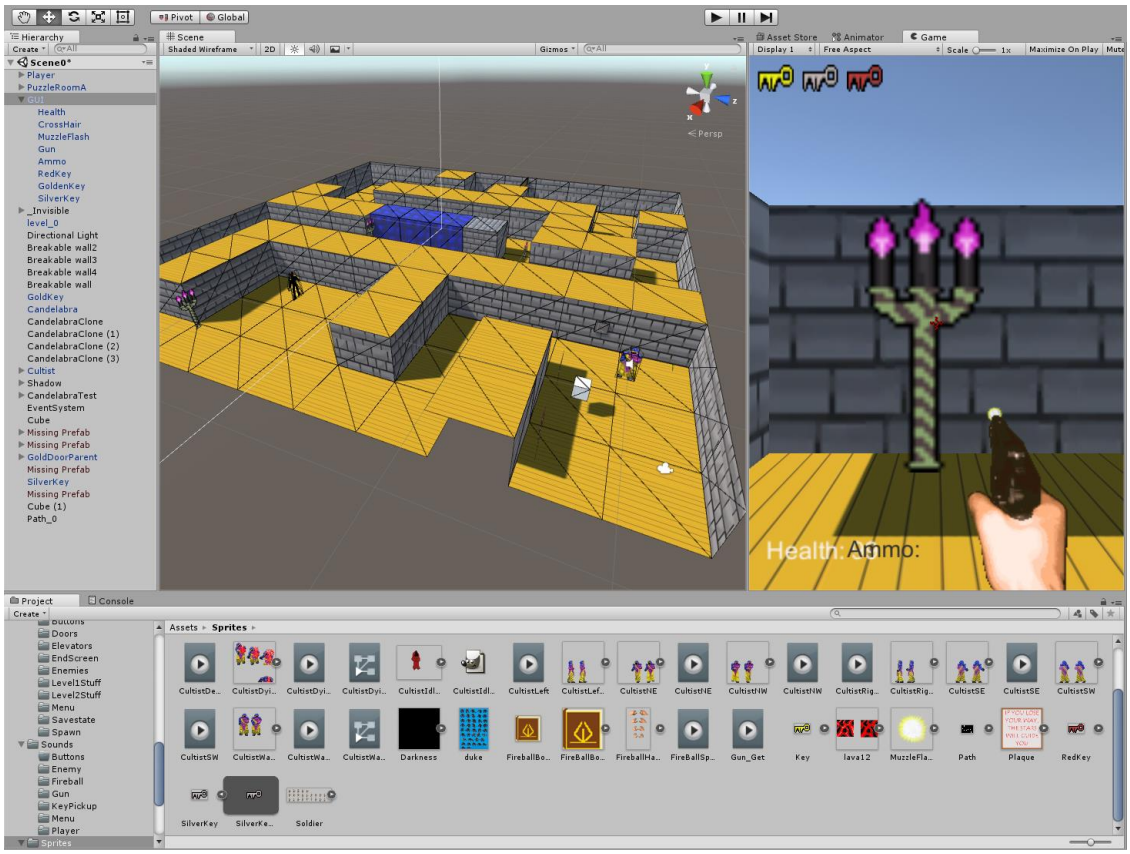
Kuva 14. Pelin toisen kentän salahuone.

Aloitus- ja lopetussijainteja voidaan vapaasti liikuttaa editorinäkymässä, ja taso liikkuu näiden kahden välillä taukoamatta. Se lähtee aina toisen sijainnin suuntaan saavuttaessaan edellisen. Näytöksen alkaessa arvotaan, mihin sijaintiin liikkuva taso lähtee ensimmäiseksi liikkumaan, ja nopeus, jolla liikkuva taso etenee.

4.1.5 Kentät ja tekstuurit

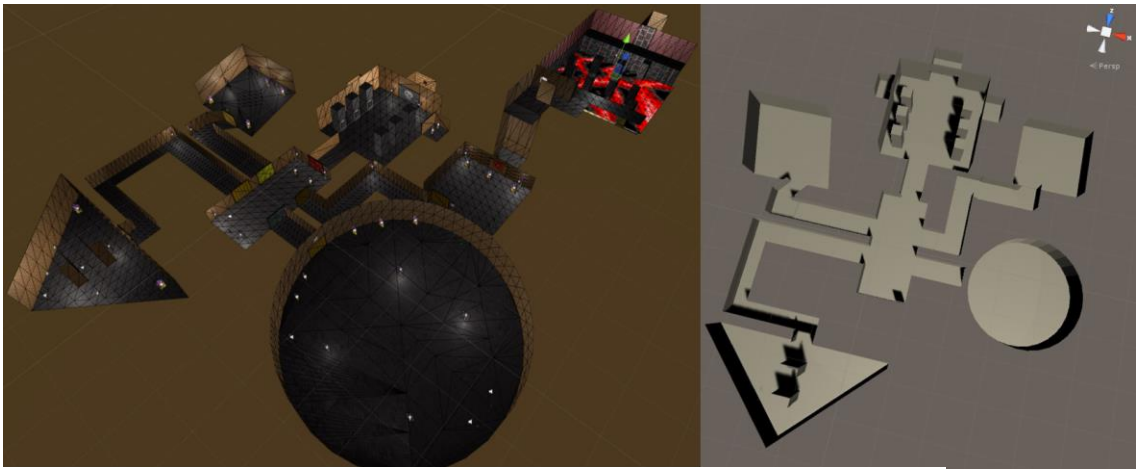
Kentät suunniteltiin ensin paperille, ja sen jälkeen niistä tehtiin sopivissa osissa 3D-mallit käyttäen hyväksi Blender-ohjelmistoa ja siirtämällä ne eteenpäin Unityyn. Teksturointi ja UV-kartat (käytetään teksturoimaan halutut 3D-mallin pinnat valituilla tekstuureilla heijastamalla ne 3D-pinnan päälle.) toteutettiin myös Blenderin puolella, mutta kätevästi muutokset siirtyivät reaaliaikaisesti Unityyn, kun objekti tallennettiin Unityn kansioon.

Testikenttä: Testikenttä oli ensimmäinen asia, joka peliä varten tehtiin. Blenderillä luotiin suuri neliön muotoinen litteä taso, joka jaettiin ruudukkoihin. Tämän jälkeen valittiin, mitkä ruuduista toimisivat seininä, ja ne nostettiin ylöspäin ohjelman avulla, jolloin jäljelle jääneet syvennykset toimivat käytävänä. Tällä tavoin sai helposti ja nopeasti luotua yksinkertaisen testiympäristön peliin toteutettaville toteutuksille. Kuvassa 15 on testikenttä editorinäkymästä.



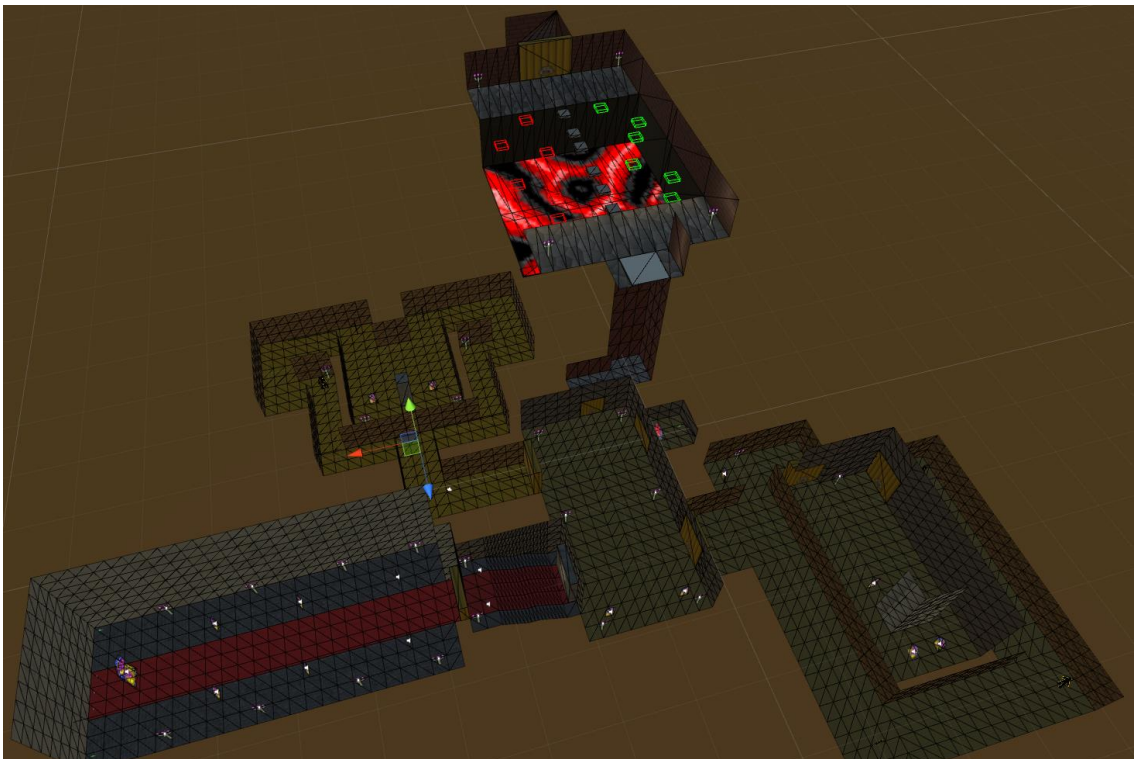
Kuva 15. Pelin testikenttä, jossa testattiin peliin lisättäviä toiminnallisuuksia.

Kenttä 1: Ensimmäisen kentän osat toteutettiin modulaarisina kokonaisuuksina, joille luotiin Blenderin puolella UV-kartat ja teksturoitiin. Tämän jälkeen kartan palaset siirrettiin Unityn komponentti-kansioon, ja ne liitettiin uudessa ensimmäistä kenttää varten luodussa kohtauksessa (scene) yhteen muodostaen ensimmäisen kentän. Ensimmäinen kenttä koostui 39:stä eri palasesta, jotka liitettiin yhteen. Kuvassa 16 vasemmalla näkyy valmis ensimmäinen kenttä ja oikealla vertailun vuoksi vielä teksturoimaton ensimmäinen kenttä ajalta, kun palasia vielä koottiin yhteen Unity3D:n editorinäkymässä.



Kuva 16. Ensimmäinen kenttä valmiina ja kokoamisvaiheessa.

Kenttä 2: Pelin toinen kenttä toteutettiin samalla tavalla, kuin ensimmäinen, mutta toisessa kentässä kiinnitettiin enemmän huomiota palasten mittasuhteisiin, jotta ne sopisivat pienemmällä vaivalla toisiinsa, ja keskityttiin hieman isompiin kokonaisuuksiin monien pienten palasten sijaan. Toisessa kentässä oli palasia 7, mikä nopeutti kentän kokoamista huomattavasti. Kuvassa 17 on esiteltynä valmis pelin toinen kenttä.



Kuva 17. Valmis pelin toinen kenttä Unity3D:n editorin kautta nähtynä.

4.1.6 Sekalaiset

Seuraavaksi käydään nopeasti läpi pelin tärkeimmät pienet komponentit, jotka prototyyppiä varten toteutettiin.

Tavarat: Kentissä on useita terveyspakkauksia, ammuslippaita ja uusia aseita. Ne kaikki koostuvat yhdestä spritestä, joka on kääntynyt aina pelaajaa kohti. Tavaroissa on kiinnitettynä triggerinä toimiva box collider, jonka sisälle astuessa tavara tuhotaan ja pelaajalle lisätään määrätty määrä terveyttä tai ammuksia.

Valaistus: Valaistusta varten tehtiin kynttelikköjä varten kaksi spriteä, jotka animoitiin Unityn animaattorissa. Kiinnitin Unityn oman valolähteen (point light) kynttelikköön, jotta valo ikään kuin lähtisi kynttelikköstä ympäristöön.

Ovet: Pelin ovet toteutettiin kiinnittämällä ovi-objektiin mallinnettu 3D -palikka, joka oli teksturoitu oven näköiseksi ja jolla on animaattori -komponentti. Ovessa on kiinni triggerinä toimiva box collider, joka tullessaan kosketuksiin pelaajan kanssa odottaa tietyn napin painallusta, jolloin oven skripti lähettää animaattorille viestin avata ovi. Ovet siis liikkuvat animaation avulla pelissä. Ovia on myös erilaisia: jotkin vaativat avaimen tai jonkin tietyn ehdon auetakseen.

Aloituspiste (spawnpoint): Aloituspiste asetetaan johonkin pisteeseen pelimaailmassa, ja pelin käynnistyessä aloituspisteen skripti etsii pelaajaksi merkittyjä objekteja ja lähettää pelaajakontrollerille käskyn siirtää itsensä samaan kohtaan.

Lisäefektit: Pelaajahahmon ollessa liikkeessä aseita liikuttavan skriptin lisäksi pelin kameraan on lisätty niin kutsuttu camerashaker-skripti, jota käytetään pelaajan ampuessa tai tulipallon osuessa johonkin objektiin. Tämä heiluttaa kameraa hieman ja luo enemmän immersiota, mikä elävöittää peliä. Myös pelin vihollisille lisättiin partikkeliefektejä, jotka aktivoituvat aina saadessa vahinkoa.

4.1.7 Äänet

Äänet tehtiin käyttäen Audacity-ohjelmistoa, jonka avulla äänitettiin sopivankuuloisia ääniä peliä varten ja muokattiin niitä hieman käyttäen avuksi Audacity-ohjelmiston tarjoamia työkaluja. Äänet tallennettiin ”.wav”-formaatissa Unity3D:n komponenttikansioon.

4.2 Projektin eteneminen

Prototyyppi toteutettiin kolmessa kokonaisuudessa. Ensimmäisessä luotiin testiympäristö, jossa lisättäviä toiminnallisuuksia ja toteutuksia voitaisiin testata ja korjailla. Sen jälkeen ne siirrettiin pelin ensimmäiseen kenttään, jossa jatkettiin toiminnallisuuksien optimoimista ja testausta, kunnes ne täyttivät niille asetetut kriteerit. Kolmannessa vaiheessa testattiin, toimivatko pelin toiminnallisuudet, kun ne siirretään uuteen näytökseen eli pelin toiseen kenttään, ja luotiin muutamia uusiakin. Testattiin myös pelaajahahmon parametrien säilyttämistä näytösten välissä (ammustilanne, terveyspisteet).

Testikenttä

Projekti aloitettiin tekemällä ensin Blender-ohjelmiston avulla testikenttä, jossa pystyisi kokeilemaan kaikkia toimintoja ennen niiden toteuttamista kentissä.

Tästä seuraava looginen askel oli toteuttaa pelaajahahmo ja pelaajahahmolle ohjain, jotta pelaajaa voisi liikutella pelimaailmassa. Unityssä luotiin pelaajahahmolle skripti, jonka avulla pelaaja pystyy liikkumaan ja hyppimään. Kiinnitin myös pelin pääkameran pelihahmon pään tasolle, ja lisäsin hiirellä ympärille katselemisen.

Seuraavaksi toteutettiin testikenttään muutama ovi, joita pelaaja pystyy aukomaan ja kokeiltiin pintojen teksturointia, sekä alettiin tekemään ensimmäistä vihollista ja asetta. Kaikista pelin komponenteista vihollisen ohjelmoimisessa olikin suurin työ, ja se oli kehityksessä lähes koko projektin ajan. Varsinkin vihollisen spriten kohdistamisen kanssa oli melkoisesti ongelmia, ja sitä optimoidessa kesti paljon suunniteltua kauemmin.

Kun vihdoin kuitenkin saatiin spriten kohdistus, pelin pistooli ja vihollisten tekoäly toimimaan hyväksyttävästi, alkoi ensimmäisen kentän suunnittelu.

Ensimmäinen kenttä

Ensimmäisen kenttä toteutettiin modulaarisina paloina, mutta hyvin pian selvisi, että kaikkia palojen mittasuhteita ei ollut saatu oikein, mikä johti projektin viivästymiseen, koska jouduttiin sovittelemaan palasia yhteen ja venyttämään niitä, jotta ne sopisivat yhteen.

Kun ensimmäinen kenttä oli koottuna, alkoi sen täyttäminen tavaroilla, ovilla ja teksturoimalla seiniä. Prototyyppiin peliruudulle lisättiin UI-elementtejä, ja peliin tehtiin efektejä sekä viilattiin vihollisten tekoäly kuntoon. Lisäksi toteutettiin välietapit vihollisia varten ja luotiin peliä varten ison määrän spritejä ja tekstuureja.

Kaikkia toteutuksia testattiin aina ensin testikentässä ennen niiden siirtämistä pelin ensimmäiseen kenttään. Lopulta oli aika pistää ensimmäinen kenttä kokoon toimivaksi paketiksi, ja sitä varten kenttää piti pelata ja testata paljon. Koodissa tai asetuksissa olevien virheiden metsästys ja pelitasapainon hiominen haluttuun pisteeseen vei lukuisia iteraatioita, ja kenttää tuli testausvaiheessa pelattua läpi niin monta kertaa, että siitä muovautui lopulta hieman vaikeampi, kuin oli alun perin suunniteltu.

Kun ensimmäinen kenttä oli lähes valmis, lisättiin peliin myös alkeellinen aloitus- ja lopetusruutu ja kentän loppuun ruutu, joka kertoi pelaajalle, montako vihollista tämä eliminoi, paljonko salaisuuksia löytyi ja kuinka kauan kentän suorittamiseen meni aikaa. Aloitus- ja lopetusruutu ovat omat näytöksensä projektissa, ja niistä siirtyminen toteutetaan lataamalla seuraava näytös. Kentän lopussa näkyvä ruutu on koko ajan kuvaruudulla ensimmäisenä näkyvä asia, mutta se aktivoidaan vasta, kun pelaaja painaa kentän lopettavaa lopetusnappia.

Arviolta noin 350 tuntia meni projektissa tämän pisteen saavuttamiseen. Toki suuri osa tästä ajasta meni ohjelmistojen opetteluun ja ensimmäisen kentän iterointiin.

Toinen kenttä: Pelin toinen kenttä valmistui hyvin paljon nopeammin kuin ensimmäinen. Suurin osa pelin peruspalikoista oli jo valmiiksi tehtynä, joten ei tarvinnut kuin siirtää valmiit objektit Unityyn luodusta valmiiden komponenttien kansiosta uuteen näytökseen.

Silti toiseenkin kenttään upposi aikaa. Kentän 3D-mallien suunnitteluun käytettiin huomattavasti enemmän aikaa, jotta aikaa saataisiin säästettyä myöhemmin kentän palasia yhteen liitettäessä. Myös joidenkin uusien ominaisuuksien lisääminen viivästytti, kuten liikkuvien tasojen ja uuden vihollistyypin lisääminen ja pelaajahahmon tietojen siirtäminen yhdestä näytöksestä toiseen.

Pelin toisessa kentässä kokeiltiin myös hieman suoraviivaisempaa etenemismallia. Toisessa kentässä ei enää käytetä erilaisia avaimia uusien ovien avaamiseen, vaan kentässä on kaksi erillistä huonetta, joissa olevien vipujen aktivoimisella viimeisen oven voi

avata. Kentän lopettavan ’exit’-napin sijasta pelaajan on päihitettävä viimeisessä huoneessa oleva tavallista isompi ja kestävämpi kultisti, minkä jälkeen kentän lopetusruutu tulee näkyviin.

Kaikkiaan toista kenttää tehdessä ja testatessa meni suunnilleen 80 tuntia.

5 Valmis prototyyppi

5.1 Projektin onnistuminen

Prototyypissä saatiin toteutettua kaikki tärkeimmät toiminnallisuudet, jotka sille asetettiin kriteereiksi projektin määrittelyissä. Prototyypissä on kaksi pelattavaa kenttää, ja kaikki tärkeimmät FPS-pelin toiminnallisuudet ovat toteutettuna. Tämän takia aikaa jäi myös ylimääräisten toiminnallisuuksien toteuttamiselle. Näillä mittareilla projektin voidaan siis katsoa onnistuneen.

Prototyyppiin jäi vielä pientä korjailtavaa, mutta puutteet ovat pieniä eivätkä häiritse tai keskeytä itse peliä. Näistä paras esimerkki ovat toisen kentän liikkuvat tasot, joita ei ehditty saada toimimaan halutulla tavalla projektin aikarajan puitteissa. Myös pelaajahahmon ohjattavuus kaipaisi hieman enemmän jouhevuutta, mikä vaatisi lisää optimoimista. Pelin toiselta viholliselta, varjo-olennolta ei myöskään ole piirrettynä kuin yksi sprite, eikä sitä näin ollen ole animoitu, mutta tämä toiminnallisuus esiteltiin jo kultistin kohdalla.

Projekti oli urakkana melko iso, ja se heijastuu valmiiseen peliin kuuluvien komponenttien määrässä. Peliä varten tehtiin seuraavat komponentit:

- skriptit: 65
- koodia: arviolta n. 1 700–1 900 riviä
- tekstuurit: 25
- 3D-mallit: 25
- äänet: 18

- spritet: 75
- näytökset: 4 (2 pelattavaa kenttää ja aloitus- ja loppuruutu) + 1 (testikenttä)
- vihollisia: 2 erilaista
- aseita: 2 (pistooli + tulipallo) + 1 (lähitaistelun saksihyökkäys)
- arvioitu peliaika: 10min

5.2 Ongelmatilanteet projektin aikana

Ohjelmoinnin puolella kaksi suurinta kompastuskiveä olivat vihollisten animoidut 2D-spritet, tai tarkemmin ottaen niiden kääntäminen pelaajaa kohti ja oikean animaation syöttäminen. Toinen ongelmia aiheuttanut asia oli pelin toisessa kentässä olevien liikkuvien tasojen saaminen toimimaan halutulla tavalla.

Spritejen kohdistaminen oli ensimmäinen iso ongelma, ja meni jonkin aikaa, ennen kuin päästiin sellaiseen ratkaisuun, joka oli tyydyttävä. Kulmien laskeminen vaati monta iteraatiota, ennen kuin se toimi oikein, ja animaatiosta toiseen siirtyminen ja oikean animaation syöttäminen koodin kautta oli oma projektinsa.

Selvisi, että viive animaatioiden välillä tai niiden puute johtui siitä, että animaattorin asetuksissa oli asetettu viiveajat animaatioille, jolloin animaattori aina suoritti ne loppuun ennen seuraavaan hyppäämistä. Tämä aiheutti ongelmia, sillä pelaajan pyöriessä lähellä vihollisen ympäri ei animaattori pysynyt enää mukana, koska animaatiot oli asetettu kehämalliin, jossa toistettava animaatio siirtyi aina yhdestä kulmasta toiseen.

Asia korjattiin poistamalla viiveet animaatioiden vaihdosta, mutta tästä jäi mahdollinen ongelma animaattorin kehärakenteeseen, joten keskelle kehää asetettiin tyhjä animaatio, jonka kautta animaatio kulki. Näin ei päässyt enää syntymään tilannetta, jossa jokin animaatio jäi päälle, koska koodista syötettävä kokonaisluku ei vastannut seuraavien kulmien arvoja.

Liikkuvat tasot aiheuttivat paljon enemmän vaivaa, kuin oli ensin ajateltu. Ellei ongelmana ollut se, että tasot liukuivat pois pelaajan alta, joko pelaaja jäi aina hieman jälkeen

tasosta tai oli tasossa kiinni kuin liimattu. Loppujen lopuksi ongelma selvisi laittamalla tason päälle triggeri, joka asetti pelaajan rigidbodyn kinemaattiseksi ja pelaajahahmon periytymään liikkuvasta tasosta. Tasolla liikkuminen korjaantui, kun pelaajan kontrollerin skriptin kautta poistettiin nämä muutokset aina, kun liikkumisnappia painettiin. Täysin toimiva tämä ratkaisu ei ollut, sillä huomasin pientä liikkumista tasolla seistessä, vaikka en itse liikuttanut pelaajaa, mutta aikarajan puitteissa pidin tuloksia hyväksyttävinä.

5.3 Prototyypin vertaus edeltäjiinsä

Prototyyppi vertautuu hyvin esikuviinsa. Peli näyttää ja tuntuu aivan vanhan FPS-pelin uudemmalta versiolta oikeiden 3D-mallien ja korkeamman resoluution omaavien sprite-jen sekä tekstuurien takia. Kentät noudattavat melko uskollisesti vanhojen FPS-pelien luomaa mallia, jossa yleensä alusta alkaen nähdään määränpää, jonne pelissä mennä, mutta sinne päästäkseen joutuu kiertämään ensin muualle. Kaikilla pelin kenttien huo-neilla on myös jokin tarkoitus, eikä yhtään tyhjänpäiväistä huonetta ole, vaan jokainen niistä auttaa pelaajaa jollakin tavalla. Kuvissa 18–21 on esitettynä pelikuvaa valmiista prototyypistä.

Kuva 18. Pelin ensimmäisen kentän alku.

Kuva 20. Pelin aloitusruutu.

Kuva 19. Pelin toinen kenttä ja varjo-olento.

Kuva 21. Pelin toisen kentän viimeiseen huoneeseen johtava ovi.

5.4 Projektin jatkosuunnitelmat ja pohdintaa

Jos projekti toteutettaisiin nyt uudelleen, sitä lähestyttäisiin vähän eri tavalla. Projektin alussa ei ollut samanlaista käsitystä pelinkehittämisen prosessista kuin nyt sen jälkeen. Ei osattu täysin hahmottaa koko projektin laajuutta, ja se johti koodipohjassa muutamiin rakenteellisiin asioihin, jotka toteutettaisiin toisin kuin nyt. Esimerkiksi nyt ovat Player-controllerissa varastoituna sekä pelaajan tiedot että pelaajan liikkumiseen tarvittavat koodit. Nämä olisi mielestäni voinut toteuttaa erikseen. Myös skriptien määrä voisi olla paljon pienempi, mikäli ne olisi toteutettu viisaammin.

Prototyypin pelinkehitystä jatketaan lähinnä harrasteprojektina, sillä prototyypin tekemiseen käytetty aika osoitti, kuinka työläs, ja aikaa vievä peliprojekti voi olla, kun kaikki pelin komponentit tehdään itse.

6 Yhteenveto

Insinöörityössä aikaansaatuihin tuloksiin oltiin tyytyväisiä varsinkin, kun se oli opiskelijan ensimmäinen peliprojekti. Odotukset ylitettiin, ja saavutettiin arvokasta kokemusta pelinkehittämisestä Unity3D:llä käyttäen C#-ohjelmointikieltä toiminnallisuuksien toteuttamiseen. Lisäksi opittiin hieman 3D-mallinnuksesta ja 2D-grafiikan luomisesta. Kantapään kautta myös opittiin, kuinka iso ja työläs tämmöinen peliprojekti voi olla.

Projektissa opittiin käyttämään sekä Unity3D:tä että ohjelmoimaan C#-ohjelmointikielellä hyvin. Unity3D oli helposti lähestyttävä kehitysalusta, joka kiitos suuren määrän saatavilla olevia opetusvideoita ja tutoriaaleja oli mieluisa myös opetella. Blender- ja GIMP-ohjelmistoihin tutustuttiin myös ja opittiin käsittelemään pelinkehitystä monesta eri näkökulmasta, sillä projektissa toteutettiin itse peliin vaadittavat graafiset ja ohjelmoitavat komponentit.

Lähteet

1 Wolfenstein 3D IPhone Code Review. Verkkoaineisto. Fabien Sanglard. < http://fabiensanglard.net/wolf3d/index.php>. Luettu 14.2.2018.

2 Lode's Computer Graphics Tutorial, Raycasting. Verkkoaineisto. Lode's. < http://lodev.org/cgtutor/raycasting.html>. Luettu 14.2.2018.

3 Is the Doom Engine a Raycaster?. Verkkoaineisto. Youtube. < https://www.youtube.com/watch?v=96nFJIxW-34>. Katsottu 10.1.2018.

4 Z-Clipping. Verkkoaineisto. Doom Wiki.< https://doomwiki.org/wiki/Z-clipping>. Luettu 13.4.2018

5 Doom Engine Code Review. Verkkoaineisto. Fabien Sanglard. < http://fabiensanglard.net/doomIphone/doomClassicRenderer.php>. Luettu 10.2.2018.

6 Duke Nukem 3D Code Review. Verkkoaineisto. Fabien Sanglard. < http://fabiensanglard.net/duke3d/index.php>. Luettu 11.2.2018.

7 Doom Reviews Metacritic. Verkkoaineisto. Metacritic. < http://www.metacritic.com/game/pc/doom>. Luettu 1.4.2018.

8 Problem With Modern First Person Shooter Games. Verkkoaineisto. New Statesman< https://www.newstatesman.com/culture/2014/07/problem-modern-first-person-shooter-video-games>. Luettu 1.4.2018.

9 Doom. A Classic Game Post-Mortem. Verkkoaineisto. Youtube. < https://www.youtube.com/watch?v=NnkCujnYNSo&t=977s>. Katsottu 10.12.2017.

10 The Doom Bible. Verkkoaineisto. Doomworld. < https://5years.doomworld.com/doombible/>. Luettu 15.2.2018.

11 Development of Doom. Verkkoaineisto. Doom Wiki. < https://doomwiki.org/wiki/Development_of_Doom>. Luettu 15.2.2018.

12 A Visual History of First Person Shooters. Verkkoaineisto. Ars Techica. < https://arstechnica.com/gaming/2016/02/headshot-a-visual-history-of-first-person-shooters/>. Luettu 13.3.2018.

13 2.5D Engine: Build. Verkkoaineisto. Moby Games < http://www.mobygames.com/game-group/25d-engine-build>. Luettu 1.4.2018.

14 Have Single-player FPS gone backwards?. Verkkoaineisto. Youtube. < https://www.youtube.com/watch?v=0Q6UQ2QlRH0>. Katsottu 7.10.2017.

15 Build(game engine), Wikipedia. Verkkoaineisto. Wikipedia < https://en.wikipedia.org/wiki/Build_(game_engine)>. Luettu 1.4.2018.

16 Strife: Veteran Edition Steam Spy. Verkkoaineisto. Steamspy < http://steamspy.com/app/317040>. Luettu 15.3.2018.

17 Devs Play: Doom Map 1 Hangar. Verkkoaineisto. Youtube. < https://www.youtube.com/watch?v=rV6HlBa88js>. Katsottu 17.2.2018.

18 Sprite. Verkkoaineisto. Doom Wiki. < http://doom.wikia.com/wiki/Sprite>. Luettu 1.10.2017.

19 Rigidbody. Verkkoaineisto. Unity3D. < https://docs.unity3d.com/ScriptReference/Rigidbody.html>. Luettu 5.4.2018.

20 Retro. Verkkoaineisto. Cambridge Dictionary < https://dictionary.cambridge.org/dictionary/english/retro>. Luettu 12.4.2018.

21 Kushner, David. 2003. Masters of Doom. E-kirja. Random House

22 Doom Models. Verkkoaineisto. Doom Wiki < https://doomwiki.org/wiki/Models >. Luettu 13.4.2018.

23 Binary Space Partitioning. Verkkoaineisto. Wikipedia. < https://en.wikipedia.org/wiki/Binary_space_partitioning >. Luettu 13.4.2018.

24 Front-to-Back display of BSP Trees. Verkkoaineisto. Researchgate. < https://www.researchgate.net/publication/3208236_Front-to-back_display_of_BSP_trees>. Luettu 13.4.2018.

25 Robert Prince. Verkkoaineisto. Doom Wiki. < http://doom.wikia.com/wiki/Robert_Prince >. Luettu 13.4.2018.

26 Call of Duty Induction Walkthrough. Verkkoaineisto. Prima Games. < https://www.primagames.com/games/call-duty-advanced-warfare/guides/call-of-duty-advanced-warfare-signature-series-strategy-guide/single-player-walkthrough/induction>. Luettu 13.4.2018.

27 E1M6 Doom 1993. Verkkoaineisto. Doom Wiki. < http://doom.wikia.com/wiki/E1M6:_Central_Processing_(Doom)>. Luettu 13.4.2018.

28 2.5D. Verkkoaineisto. Wikipedia. < https://en.wikipedia.org/wiki/2.5D>. Luettu 13.4.2018