Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Blaž Jeršan

Primerjava pogonov za igre

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matija Marolt

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00057/2012
Datum: 20.09.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **BLAŽ JERŠAN**

Naslov: **PRIMERJAVA POGONOV ZA IGRE**
A COMPARISON OF GAME ENGINES

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

V diplomski nalogi raziščite orodja, ki so na voljo razvijalcem iger. Osredotočite se na pogone za igre in naredite primerjavo brezplačnih različic dveh pogonov: Unity in CryEngine. Opišite prednosti in slabosti obeh pogonov na tipičnih nalogah s katerimi se srečujejo razvijalci iger: izdelava okolja, dodajanje modelov, animacija in umetna inteligenca. Razvijte tudi primera iger na obeh platformah, ki naj bosta podlagi za primerjavo.

Mentor:

doc. dr. Matija Marolt

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Blaž Jeršan, z vpisno številko **63090028**, sem avtor diplomskega dela z naslovom:

Primerjava pogonov za igre

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matije Marolta,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 12. septembra 2012 Podpis avtorja:

Kazalo

Povzetek

Živimo v svetu, kjer vedno več ljudi igra igrice. Ljudje si z njimi preganjajo dolgčas, se zabavajo, nekateri z igranjem iger celo služijo denar. Obstaja veliko različnih vrst iger. Nekatere so namenjene konzololam, druge računalnikom, nekatere pametnim telefonom in ostalim prenosnim napravam. Vse igre pa imajo nekaj skupnega. Nekdo jih je moral narediti in pri tem je uporabljal določena orodja.

V moji diplomski nalogi sem raziskal, kakšno izbiro orodij imajo na voljo razvijalci iger. Ker obstaja danes tudi veliko orodij, ki so na voljo zastonj, sem se odločil, da v moji diplomski nalogi poudarim to skupino.

Zato, da sem bolje spoznal nekatera od orodij, samo delo s pogoni za igre, in da sem lahko podal nekaj informacij tudi iz prve roke, sem tekom diplomskega dela sam naredil dve preprosti igri, ki sem ju v diplomi tudi opisal.

Ključne besede:

pogon za igre, računalniška grafika, računalniške igre, razvoj računalniških iger, Unity 3d, Cry Engine 3

Abstract

We are living in a world where an increasing number of people play video games. People use them to pass time, to have fun, some even play games to make a living. There are many types of games out there. Some are made for consoles, some for computers and some for smart phones and other portable devices. However, all the games have something in common. Behind every video game there is a guy who made them with the use of a certain set of tools.

In my bachelor's degree I decided to explore the tools that are used by the game designers. Today a lot of these tools are available free of charge, so I decided to focus on this group of tools.

While working on my bachelor's degree I also worked with some of the tools and made two simple games. I think that was necessary in order to give at least some first hand feedback on working with game engines and other similar tools.

Key words:

game engine, computer graphics, computer games, computer game design, Unity 3d, Cry Engine 3

Poglavje 1

Uvod

Živimo v svetu, kjer se vedno več ljudi odloča za odpiranje svojega podjetja, saj je dobre službe, oziroma kakršnekoli službe, vedno težje najti. Če imamo svoje podjetje, ki je dovolj uspešno, da se z njim preživljamo, si s tem, poleg službe, zagotovimo še veliko drugih prednosti, kot na primer fleksibilen urnik, odgovornost samemu sebi, itd.

Podobna stvar se dogaja v industriji izdelovanja iger. Vedno več ljudi se odloča za odpiranje svojega podjetja za razvoj iger. Za premik k samostojnim razvijalcem iger, je poleg ekonomskih razmer, še nekaj drugih razlogov.

Igračarska industrija leto za letom beleži relativno visoko rast, še posebej je rast velika na področju mobilnih iger. V zadnjih letih se tudi vedno bolj uveljavljajo tekmovanja v računalniških igrah (E-sports). Tekmovanja imajo vedno bolj bogate nagrade in pridobivajo na medijski prepoznavnosti. Nekaj let nazaj so nagradni skladi na takih tekmovanjih znašali do 100 000 dolarjev, letos pa je organizirano tekmovanje v računalniški igri "League of Legends", ki ima rekordni nagradni sklad 5 milijonov dolarjev, kar pa ni več malo, tudi če primerjamo s klasičnimi športi.

S pojavom pametnih telefonov in tablic se je močno povečala popularnost iger tudi med ljudmi, ki so na igre včasih gledali kot na popolno izgubo časa. Večina mobilnih iger je precej preprostih, hkrati pa imajo ogromen potencial za zaslužek. Zaradi tega se veliko posameznikov ali pa majhnih skupin odloči

za izdelavo igre za operacijski sistem Android ali pa iOS.

Vsi ti ljudje se soočajo z vprašanjem, kako se lotiti razvoja lastne igre. Ali naj igre razvijajo direktno v programskih jezikih (open gl), ali naj uporabljajo programska ogrodja (XNA), ali naj uporabijo že narejene pogone za igre (Unity 3d, Cry Engine, ...). Zato je osnovno raziskovalno vprašanje moje diplomske naloge, katero orodje uporabiti za razvoj računalniške igre. V diplomski nalogi sem poudaril predvsem zadnjo skupino orodji, torej že narejene pogone za igre.

Poglavje 2

Teoretične osnove

2.1 Kaj je pogon za igre?

Začnimo čisto na začetku z vprašanjem, kaj sploh je pogon za igre? Vsi, ki spremljamo igračarsko industrijo, smo že velikokrat slišali izraz "game engine" (sl. pogon za igre), vendar malo ljudi ve, kaj tak pogon pravzaprav obsega. Podobno kot motor pri avtomobilih, je pogon za igre nekakšno srce igre. Namen pogona za igre je abstrakcija različnih nalog, kot so nalaganje, prikazovanje in animiranje modelov, fizika, krmiljenje kontrol, upravljanje grafičnega vmesnika, detekcija trkov med objekti, itd. To omogoča, da se lahko razvijalci iger osredotočijo na samo vsebino igre in porabijo manj časa za tehnično plat izdelave. Na primer, če pogon za nas naredi zaznavanje trkov, je vsebina igre logika, oziroma pomen teh trkov. Tukaj lahko spet potegnemo analogijo z avtomobili. Če naš pogon za igre predstavlja motor v avtomobilu, potem predstavljata oblika avtomobila in komponente v notranjosti avtomobila vsebino igre.

Kadar govorimo o pogonih za igre, velikokrat slišimo omenjeni dve kratici, API ("application programming interface") in SDK ("software development kit"). API-ji so programski vmesniki, katere nam zagotovljajo operacijski sistemi, knjižnjice in različne storitve zato, da lahko izkoristimo njihove funkcije (so vmesnik med storitvami in uporabniki). SDK-ji so kolekcije knjižnjic,

API-jev in orodij, ki so nam na voljo za programiranje.

2.2 Rojstvo pogonov za igre

Na začetku so razvijalci iger izdelovali lastne pogone. Takrat to ni predstavljalo velikega problema, saj so bili računalniki še precej preprosti in nezmogljivi, zato so bile tudi same igre, in posledično pogoni za igre, ki so jih poganjali, precej preprosti. V zadnjih nekaj letih je s konstantnim naraščanjem moči računalnikov in posledično vedno bolj kompleksnimi igrami, cena razvoja takega pogona skokovito narasla. Zaradi tega so se pojavila podjetja, ki so se specializirala za izdelovanje celotnih pogonov za igre ali komponent za le-te. Pogone nato prodajo podjetjem, ki jih uporabijo za izdelovanje svojih iger. Ta podjetja se imenujejo "middleware provider". Ravno zaradi takih podjetij se je med razvijalci iger pojavilo vprašanje, ali naj pogon za svojo igro razvijejo sami, ali naj ga kupijo od enega izmed teh podjetij. V veliko primerih je odgovor preprost. Zakaj bi skupino programerjev plačevali celo leto za razvoj ustreznega pogona, ki bo na voljo šele, ko delo končajo, ko ga lahko za veliko nižjo ceno kupijo in ga imajo na voljo takoj.

Slika 2.1: Prvi pogon za igre.

Prvi pogon za igre, v pravem pomenu besede, je nastal ob izdelavi igre "Doom". Ker je bila igra zelo uspešna, so tudi ostali začeli posnemati ta koncept izdelave iger, kar je takrat še pospešilo in populariziralo razvoj pogonov za igre. Veliko popularnih iger je bilo narejenih z uporabo različnih pogonov za igre, ki so bili prvotno ustvarjeni za neko drugo igro. Tako je na primer nastala ena najbolj popularnih streljank, "Counter Strike", ki je uporabljala pogon ustvarjen ob igri "Half-Life". "Half-Life" je sicer uporabljal modificiran Quake-ov pogon, poimenovan GoldSrc.

2.3 Različni nivoji pogonov za igre in zakaj jih uporabljamo

2.3.1 Programska ogrodja

Programska ogrodja predstavljajo najnižji nivo. V to skupino spada XNA. Programska ogrodja so v bistvu orodja, ki nam pomagajo pri kreaciji lastnega pogona za igre. To pomeni, da moramo, preden se začnemo ukvarjati s samo vsebino igre, poskrbeti za celotno jedro igre. Sami moramo implementirati fiziko, detekcijo trkov, itd. V praksi je stvar vseeno malce bolj preprosta in nam ni potrebno prav vse sprogramirati iz nič, saj obstaja veliko knjižnic, ki nam nalogo malce olajšajo. Na primer za fiziko lahko uporabimo že obstoječo knjižnico "Havok". Za programska ogrodja je bolj primerno, da jih ne pojmujemo kot pogone za igre, ampak kot iztočnice za izdelavo svojega pogona.

Slika 2.2: XNA-jev logotip.

Če se odločimo za uporabo programskega ogrodja, kot je na primer XNA, si s tem zagotovimo popolno kontrolo in fleksibilnost pri izdelavi igre. Programerji se lahko sami odločijo, katere komponente bi radi uporabili, oziroma jih naredijo točno take, kot si želijo. Seveda to pomeni, da potrebujemo zelo veliko znanja programiranja in tudi veliko časa. Ker je čas danes zelo pomemben, se za ta pristop odloča vedno manj ljudi.

2.3.2 Pogoni za izris

Pogoni za izris predstavljajo naslednji nivo. Ti pogoni so pripravljeni za izrisovanje, upravljanje s kontrolami, upravljanje z grafičnim vmesnikom. Navadno imajo že vgrajen fizikalni pogon. V to kategorijo spada tudi Ogre engine. Pogoni za izris so kot nekakšen vmesni korak med programskim ogrodjem in pravim pogonom za igre. Še vedno moramo del funkcionalnosti sprogramirati sami, vendar je zahtevnejši del že narejen.

Slika 2.3: Ogre-jev logotip.

2.3.3 Pravi pogon za igre

V to skupino spada Unity, v katerem sem tudi sam delal, in še nekaj drugih, kot na primer Cry Engine. Za to skupino se uporablja tudi izraz "Point and click engine". Če uporabljamo programe iz te skupine, imamo že v naprej pripravljena vsa potrebna orodja. Tukaj lahko celotno igro naredimo le s klikanjem in naj ne bi potrebovali dobrega poznavanja programiranja.

Slika 2.4: Unity-jev logotip.

Če uporabljamo prave pogone za igre, je razvoj iger zelo hitro in precej preprosto opravilo, vendar smo kot razvijalci zelo omejeni, saj moramo uporabljati le orodja, ki so nam na voljo. Velikokrat so ti pogoni prirejeni le za določen tip iger, na primer streljanke, in je zato v njih težko narediti kaj drugačnega in inovativnega.

2.4 Prednosti uporabe pogona za igre

2.4.1 Hitrost razvoja igre

Prvi kriterij, po katerem lahko ocenjujemo prednosti razvoja igre z uporabo pogona za igre, je gotovo hitrost razvoja. Zanima nas, koliko časa lahko prihranimo, če našo igro razvijamo s pogonom za igre, namesto s preprostim programskim ogrodjem, kot je na primer OpenGL. Kot sem omenil že v drugem poglavju, je glavna prednost uporabe naprednega orodja za razvoj iger (pogon za izris in pogon za igre) prav v varčevanju s časom in poenostavljanju dela. Torej lahko pričakujemo, da bo hitrost razvoja igre z uporabo pogona za igre mnogo krajši.

2.4.2 Učna krivulja

Učna krivulja predstavlja, kako enostavno je neko stvar začeti uporabljati in kako težko je določeno stvar obvladovati. Učna krivulja je lahko strma ali položna. Strma učna krivulja pomeni, da je znanje, ki je potrebno, da stvar razumemo in jo lahko normalno uporabljamo, visoko. V primeru strme učne krivulje se nam navadno izplača, da se stvari posvetimo in jo obvladamo. Nasprotno pa položna učna krivulja pomeni, da za razumevanje in normalno uporabo ni potrebno veliko truda. V primeru položne učne krivulje, navadno ne pridobimo veliko, če se stvari posvetimo in jo obvladamo. Neodvisno od strmine krivulje je pomembna še začetna vrednost, ki določa, kako težko je stvar začeti uporabljati.

Tudi tukaj so v prednosti pogoni za igre, saj so narejeni prav z namenom, da delo olajšajoa. Učna krivulja pogonov za igre je sicer celo bolj strma kot za delo brez pogona, saj imajo pogoni ogromno naprednih funkcionalnosti, ki jih na začetku ne potrebujemo. Sam začetek je mnogo lažji in zato večini ljudi ostane več veselja za nadaljnje učenje.

Čeprav je lažje prvo igro izdelati v enem izmed pogonov za igre, sem zasledil, da večina profesionalnih razvijalcev iger predlaga, da vseeno začnemo

pri najnižjem nivoju, saj je za dobrega razvijalca nujno potrebno poznavanje osnov delovanja iger.

2.4.3 Ponovna uporabnost

Še ena privlačna točka pogonov za igre je možnost preproste ponovne uporabe kode, modelov, svetov, itd. Tukaj bi lahko argumentirali, da lahko tudi v primeru, ko uporabljamo OpenGL, napisano kodo skopiramo in uporabimo še enkrat, kar je seveda res, vendar pa je v pogonih za igre to narejeno veliko bolj elegantno in ne zahteva iskanja ter prilagajanja kode novi uporabi.

2.4.4 Zanesljivost

Pogone za igre izdelujejo velika podjedja z ogromnim številom programerjev, kar posledično v izdelavo pogonov prinese tudi preverjanje kakovosti. Navadno smo lahko za pogone prepričani, da so zanesljivi in da se nam ne bodo pojavljali naključni hrošči. Če se celotne izdelave lotimo sami, se nam hitro zgodi, da v igro zakodiramo kakšnega hrošča, ki ga je nato zelo težko odstraniti.

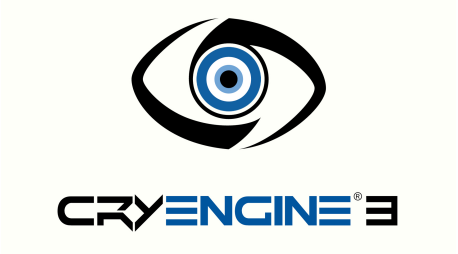
2.4.5 Prilagodljivost

Na področju prilagodljivosti pa je v veliki prednosti izdelovanje iger samostojno, saj nimamo popolnoma nobenega pravila, po katerem se moramo ravnati. To je, po mojem mnenju, tudi glavna slabost pogonov za igre. Težko je narediti univerzalno orodje, ki bo vse opravljalo perfektno. To dokazuje Cry Engine, ki na področju prvoosebnih streljank prednjači, vendar pa česa drugega v njem sploh ne moremo narediti.

Poglavje 3

Primerjava pogonov za igre

Med mnogimi brezplačnimi pogoni za igre, sem se odločil, da bom naredil bolj obsežno primerjavo dveh. To sta Cry Engine 3 in Unity 3d. Za ta dva sem se odločil zato, ker ima vsak neko svojo posebnost. Unity je trenutno najpopularnejši pogon za igre, popularnost katerega še vedno hitro raste. Cry Engine pa je trenutno najnaprednejši pogon za igre. Poleg tega, sem v Unity-ju že izdelal preprosto igro. Zanimalo me je, če je konkurenčen Cry Engine-u.



Slika 3.1: Cry Engine-ov logotip.

Primerjave sem se lotil v istem vrstnem redu, kot se navadno lotimo izdelovanja igre, saj se mi je zdelo to najbolj pametno in uporabno za bralca.

3.1 Podpora različnim platformam

Prva stvar, ki nas zanima pri pogonu za igre je, ali lahko z njim realiziramo našo idejo za igro. Ideje so lahko zelo različne. Nekateri bi radi izdelali sodobno streljaško igro, drugi pa zabavno arkadno igro, ki bi jo lahko prodajali na iStore-u. Ob izboru pogona se moramo prepričati, ali le-ta podpira platformo, na kateri bi radi izdelali našo igro. Na tem področju se primerjana pogona zelo razlikujeta.

Unity je tukaj zelo močan, saj podpira razvoj iger za vse mogoče platforme. Z njim lahko naredimo igro za računalnike, ki uporabljajo operacijski sistem Windows ali Apple-ov OS X (v novi verziji Unity-ja, ki izide kmalu, bodo podprti tudi Linux-i), konzole (Xbox 360, Play Station 3 in Nintendo Wii) in tudi pametne telefone (Android in iOS). Unity podpira celo igre v spletnih brskalnikih.

Cry Engine podpira veliko manj platform. Z njim je možno razvijati igre le za Windows-e, Xbox 360 in Play Station 3.

Omenil bi še, da je Cry Engine specializiran za izdelavo prvoosebnih iger, kar nas še dodatno omejuje pri njegovi uporabi.

Na tem področju je torej v veliki prednosti Unity, s katerim je možno realizirati praktično vse ideje za katerokoli platformo.

3.2 Cena

Ko se prepričamo, da je z določenim pogonom našo igro možno narediti, je naslednje vprašanje njegova cena. Če igro izdelujemo le za lastno veselje, je to vprašanje odveč, saj si lahko oba pogona pridobimo brezplačno. Takoj ko želimo našo igro tržiti in s tem nekaj zaslužiti, pa želijo nekaj od tega imeti tudi razvijalci pogona. Tako kot pri prejšnjem poglavju, se tudi tukaj pristop obeh pogonov popolnoma razlikuje.

Razvijalcem Unity-ja je dovolj, da kupimo licenco za Unity Pro, ki stane 1500 dolarjev. Ob nakupu licence ne pridobimo le dovoljenja za prodajo iger, ampak tudi nekaj funkcij, ki so pri brezplačni verziji zaklenjene. Poleg

Unity Pro-ja lahko kupimo posebne verzije, ki so prirejene za posamezne platforme (na primer Unity iOS Pro in Unity Android Pro), tudi te stanejo 1500 dolarjev.

Cry Engine razvijalcem, ki želijo svoje igre prodajati, ponuja dve možnosti. Prva je nakup licence, ki stane kar 1.2 milijona dolarjev, kar je za posamezne razvijalce, oziroma manjše skupine, odločno preveč. Druga možnost pa je, da jim razvijalci odstopijo 20 odstotkov od vrednosti prodanih izvodov igre.

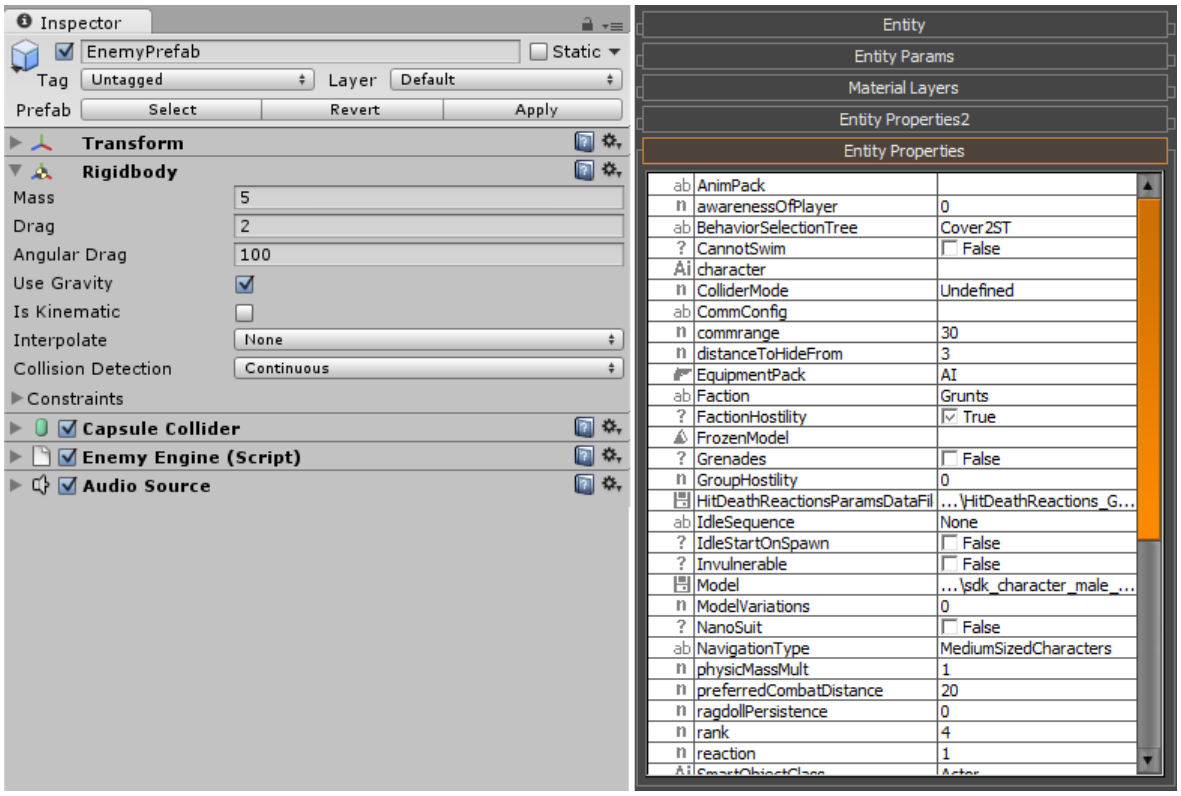
Oba pristopa imata svoje prednosti. Ko pri Unity-ju opravimo nakup, nam na to ni potrebno več misliti in imamo ves zaslužek zase. Pri Cry Engine-u ne potrebujemo nikakeršnega začetnega vložka, vendar nam neprestano jemljejo del dobička. Če na problem gledamo matematično, se Unity izplača v primeru, da prodamo za več kot 7500 dolarjev izvodov naše igre, kar se v primeru, če želimo od razvoja iger živeti, zgodi zelo hitro.

3.3 Težavnost uporabe

Delo s pogoni za igre naj bi bilo preprosto in intuitivno, vendar so, kot pri vsaki stvari, tudi tukaj potrebna določena začetna znanja.

Unity velja za enega najpreprostejših pogonov za začetek dela. Na začetku je veliko preprostejši za razumevanje. Zaradi modularne zasnove "inspector-ja" (sl. raziskovalec), kjer se ob izboru določenega objekta prikažejo le komponente, ki jih objekt trenutno obsega, lažje sledimo našemu delu. Na primer, ko prvič naredimo kocko, skozi katero želimo, da ne bo možno prosto prehajati, v raziskovalcu vidimo komponento "box collider" (sl. škatljasti trkalnik), za katero si bomo tako hitro zapomnili, da je njen namen določanje fizičnih mej objekta. Za programiranje je pri Unity-ju na voljo več programskih jezikov, kar poveča možnost, da smo s kakšnim v preteklosti že delali. Prav tako se vso logiko, ne glede na namen uporabe, kodira v enotnem skriptnem sistemu.

Cry Engine je na začetku nekoliko bolj okoren. Na zaslonu urejevalnika nas pričaka ogromno različnih gumbov, za katere moramo najprej ugotoviti,



Slika 3.2: Raziskovalec objektov, Levo - Unity, Desno - Cry Engine.

kaj sploh naredijo. Raziskovalec objektov, ki se v Cry Engine-u imenuje "Rollup Bar", ima precej podobno zasnovo, kot pri Unity-ju. Prikaže nam vse komponente, ki jih trenutni objekt obsega, vendar imamo ob samem pregledu komponent na voljo veliko večje število parametrov, za katere prav tako potrebujemo nekoliko več časa, da jih spoznamo. Programiranje v Cry Engine-u se lahko odvija na več različnih načinov. Lahko, kot pri Unity-ju, pišemo kodo, lahko pa uporabljamo Cry Engine-ovo vizualno programiranje, poimenovano "Flow Graph". Sam sistem vizualnega programiranja je sicer zelo zanimiv, vendar je tudi tukaj potrebno nekoliko več časa, preden se navadimo na nov pristop.

Mislim, da je lažje začeti delati z Unity-jem, saj nas na začetku ne preplavi s prevelikim številom opcij in informacij. Prav tako menim, da ima povprečni človek, ki želi izdelati svojo igro, vsaj malo predznanja o programiranju, in se bo zato na začetku bolj domače počutil v Unity-jevih skriptah kot pa Cry Engine-ovih "Flow Graph-ih".

3.4 Izdelava okolja

Oba pogona imata sistem za izdelavo okolja. Na prvi pogled izgledata zelo podobno. V obeh lahko nastavimo velikost osnovne ploskve, ki ji nato lahko določamo višino in teksturo. Cry Engine ima z orodjem "Layer painter" možnost barvanja teksture na podlago, kar omogoči preprosto prehajanje med območji (na primer, peščena plaža preide v travnato površino). Oba imata že nekaj predpripravljenih objektov, ki jih lahko poljubno postavljamo v sceno. Število predpripravljenih objektov je veliko večje v Cry Engine-u, vendar to ni ključnega pomena, saj ob izdelavi lastne igre navadno izdelamo lastne modele.

Cry Engine za sence na sliki 3.2 poskrbi avtomatsko. V Cry Engine-u so vse sence dinamične, kar pomeni, da se ob premikanju objektov sence premikajo z njim. V Unity-ju je potrebno sence nastaviti naknadno, z uporabo "lightmapping-a" (sl. mapiranje svetlobe), sence narejene z uporabo te

Slika 3.3: Preprosta scena, Levo - Unity, Desno - Cry Engine.

tehnike so zapečene na teksturo in so statične. Če želimo v Unity-ju imeti dinamične sence, je potrebno imeti kupljeno Pro verzijo. Na sliki vidimo še eno zelo uporabno funkcijo Cry Engine-a, ki je uporabljena na levem drevesu. Če želimo, da se objekt poravna glede na površje, lahko v Cry Engine-u samo odkljukamo opcijo "Align object to the surface normal" (sl. poravnaj objekt z normalo površja), ki nam avtomatsko poravna objekt s površjem. To je potrebno v Unity-ju narediti ročno ali pa s pomočjo skript.

Če želimo, da okolje v naši igri dosega visok nivo realizma, je boljša izbira Cry Engine.

3.5 Uvoz modelov

Modele, ki jih želimo uporabiti v naših igrah, navadno izdelamo v programih, ki so namenjeni 3d oblikovanju, zato je pomemben faktor pri izboru pogona, katere programe pogon podpira in kako težko je modele iz programa uvoziti v pogon.

V Unity lahko uvažamo modele iz večine 3d programov (Maya, 3ds Max, Cinema 4D, Cheetah 3D, Modo, Lightwave, Blender, itd.). V večini primerov je za uvoz potrebno samo prestaviti datoteke modelov v mapo s sredstvi("assets"), ki se nahaja v našem projektu.

Cry Engine podpira le tri programe in sicer: 3ds Max, Maya in Softimage XSI. Za pravilen uvoz modelov, moramo uporabiti ustrezne vtičnike ("plugin"), ki naredijo ustrezno hierarhijo datotek (v Cry Engine-u so animacije, geometrija in teksture shranjene v ločenih datotekah).

3.6 Animacija likov

V Cry Engine-u poznamo tri vrste animacij. Animacija trdnih geometrijskih objektov ("animated hard body geometry data"), animacija likov ("animated character data") in animacija obraza ("animated facial sequence"). V Unityju se vse animacije obravnavajo enako.

Preproste animacije premikov trdnih objektov so v obeh orodjih praktično enake. V orodju za animiranje določimo začetno in končno pozicijo, orientacijo objektov in trajanje animacije, vmesne korake pa naredi orodje avtomatsko.

Pri animaciji človeških likov postane animiranje bolj zanimivo. V Cry Engine-u potrebujemo pravilno sestavljen model, ki vsebuje okostje. To je najlažje narediti že v 3d programu ob izdelavi samega modela. Če imamo pravilno nastavljena imena kosti v skeletu, nam Cry Engine v "Character Editor-ju" (sl. urednik likov), omogoči uporabo vseh predpripravljenih animacij, ki obsegajo mirovanje, tek, streljanje z orožjem, itd. Če izdelujemo prvoosebno streljanko, nam v Cry Engine-u praktično ni potrebno izdelovati svojih animacij za like. Premikanje okončin je realizirano z inverzno kinematiko (določimo pozicijo dlani, glede na katero sistem avtomatsko preračuna kote med ostalimi kostmi v roki in jih pravilno postavi). Za pomoč pri animaciji obrazov ima Cry Engine še eno zanimivo funkcijo. Kadar morajo naši liki povedati kakšno vrstico besedila, lahko uporabimo avtomatsko animira-

nje govora. Z uporabo te opcije Cry Engine naredi valovno analizo zvoka, določi foneme in glede na njih izvede animacijo govora.

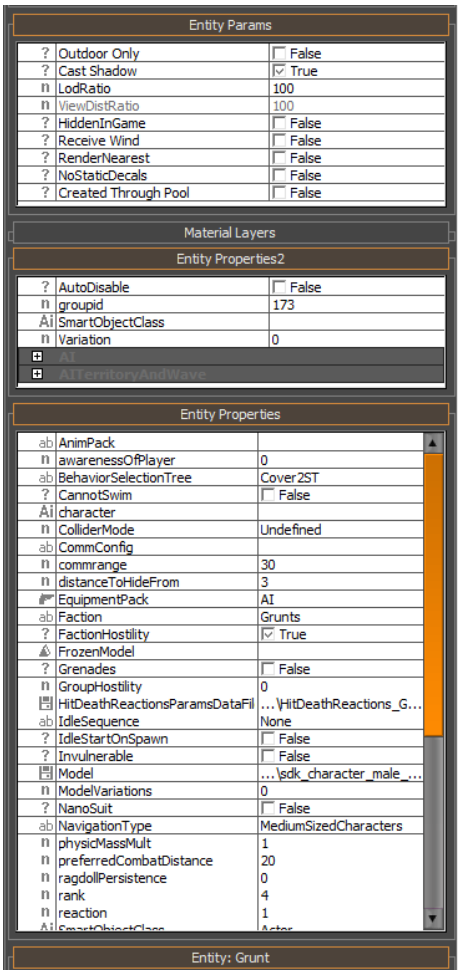
V Unity-ju predpripravljenih animacij za premikanje ljudi nimamo, ampak jih moramo izdelati sami. Podobno kot pri animaciji preprostih trdnih objektov, tudi tukaj nastavimo pozicijo okončin v nekaj "key frame-ih" (sl. ključnih slikah), med katerimi sistem naredi prehod. Če želimo v Unity-ju uporabljati inverzno kinematiko, jo moramo načeloma realizirati sami, v praksi pa si lahko pomagamo z Unity-jevim "asset store-om" (sl. trgovina s sredstvi), kjer je preprosta skripta za inverzno kinematiko na voljo celo brezplačno.

Za igre, v katerih nastopa veliko človeških likov in je realizem animacij pomemben, je Cry Engine veliko bolj pametna izbira. Če v naši igri nastopajo ne-humanoidni liki, kot na primer v moji igri, sta pogona za animiranje zelo enakovredna.

3.7 Umetna inteligenca in skriptiranje

Ko imamo izdelane like, ki nastopajo v naši igri, jim je potrebno dodati še logiko obnašanja oziroma umetno inteligenco.

Cry Engine vsebuje veliko orodij za delo z umetno inteligenco. Možnost imamo tudi popolnoma avtomatske generacije celotne umetne inteligence. Če uporabimo to opcijo, je celotno obnašanje računalniško vodenih likov odvisno od parametrov, ki jim jih nastavimo v urejevalniku. V tem primeru so liki sposobni iskanja poti, iskanja zaklona in samoobrambe. Če so označeni kot naši sovražniki, nas bodo ob detekciji napadli.



Slika 3.4: Parametri lika.

Če želimo temu sistemu dodati malce več kontrole, si lahko pomagamo z nekaj dodatnimi pomagali. Ker je osnovno iskanje poti precej slabo in se velikokrat zgodi, da se liki ob kakšne objekte zatikajo, lahko okoli njih definiramo "forbidden area" (sl. prepovedano območje). Temu območju se bo lik sedaj izognil.

Če nam tudi to ni dovolj in želimo natančno določiti obnašanje likov, moramo uporabiti Cry Engine-ove "Flow Graph-e" in LUA skriptni jezik, s pomočjo katerih imamo popoln nadzor nad obnašanjem likov.

V Unity-ju moramo celotno umetno inteligenco narediti sami s pomočjo skript. Unity podpira tri skriptne jezike in sicer: C#, Javascript in Boo script. Največ težav povzroča iskanje poti, saj Unity ne vsebuje nobenega

algoritma, ki bi nam pri tem pomagal.

V obeh pogonih je možno narediti enako kompleksno umetno inteligenco, vendar moramo za enak rezultat v Unity-ju bolje obvladati programiranje. To sem dobro spoznal tudi sam pri izdelovanju preproste igre v obeh pogonih. V Cry Engine-u sem z uporabo preprostega "Flow Grapha", določil točko proti kateri se nasprotniki premikajo, za vse ostalo je poskrbel pogon sam. V Unity-ju sem za stražarje porabil veliko več časa, vendar na koncu vseeno niso znali zaobiti ovir, ampak so se le premikali po vnaprej določeni poti in ob izpolnjenih pogojih napadli igralca.

3.8 Zunanja pomoč

Kadar se nam pri izdelovanju igre zatakne, je vedno dobro na razpolago imeti nekoga, ki ga lahko vprašamo za nasvet. Oba pogona imata na spletu obsežno dokumentacijo, s katero si navadno lahko pomagamo sami.

Unity je trenutno zelo popularen pogon za igre, kar pomeni, da je na voljo veliko učnega gradiva ter ljudi, ki so nam v primeru težav pripravljeni pomagati. Če odgovora na kakšno vprašanje ne najdemo na spletu, ima Unity zelo aktivno skupnost, kateri lahko zastavimo svoja vprašanja kar preko Unity-jeve domače spletne strani (podstran answers.unity3d.com).

Cry Engine trenutno uporablja nekoliko manj ljudi, kar pomeni, da je pomoč nekoliko težje najti. Prav tako nisem našel nobene strani, kjer bi bili ljudje tako aktivni kot na Unity-jevi strani.

3.9 Povzetek

Vsak izmed pogonov ima svoje prednosti in slabosti, ki ju naredi različno primerne v različnih situacijah. Za konec primerjave, bi rad še enkrat poudaril v katerih situacijah je določen pogon primernejši.

Unity je za začetnike primernejši, saj je cenejši, enostavnejši za uporabo in omogoča izdelovanje kakršnekoli igre za katerokoli platformo. Glavna slabost

Unity-ja je nekoliko slabši grafični pogon, ki pa na samo igralnost igre ne vpliva. Cry Engine je bistveno dražji, bolj kompleksen in optimiziran za prvoosebne streljaške igre.

Poglavje 4

Lastno delo

4.1 Ideja

Izdelave lastne igre sem se lotil zaradi predmeta Računalniška grafika in tehnologija iger, ki smo ga imeli v tretjem letniku. Pri predmetu smo morali za seminarsko nalogo izdelati preprosto igro. Ob seminarski nalogi smo dobili groba navodila, v katerih je bilo tudi zapisano, kaj mora igra vsebovati.

Idejo za igro sem dobil tako, da sem združil elemente iz različnih iger, ki so mi všeč. Hotel sem narediti igro, ki je podobna igri "Splinter Cell", to pomeni, da vsebuje elemente skrivanja in streljanja. Želel sem uporabiti tretjeosebno kamero, ker menim, da ponuja dobro kombinacijo preglednosti in natančnosti nadzora nad našim karakterjem ter je dokaj preprosta za implementacijo. Prednost tretjeosebne kamere je, da ne potrebujemo zelo natančnih modelov in tekstur, saj vse stvari gledamo iz neke srednje razdalje. Tako kamero uporablja veliko iger, kot na primer "Diablo", kar priča, da je tak način nadzora ljudem všeč.

Ime igre je nastalo po precej zanimivi poti. V času nastajanja igre sem gledal serijo "Breaking Bad", v kateri nastopa policaj Hank, ki je bil prvotno mišljen kot glavna oseba igre. Ko sem se odločil, da bo glavna oseba igre robot in ne človek, je takoj nastala ideja za naslov "Hank the tank".

4.2 Priprava

Preden sem se lotil izdelovanja igre, sem naredil precej natančen načrt igre. Najprej sem se lotil glavnega lika. V prvotni ideji sem za glavno osebo hotel imeti človeški lik. To zamisel sem hitro opustil, saj sem ugotovil, da je z modeliranjem in animiranjem človeškega lika ogromno dela. Glavni lik je zato postal robot.

V načrt sem zapisal, česa vsega bo glavni lik sposoben. Želel sem, da ima poleg uporabe nekega strelnega orožja, na voljo tudi orožje za tiho onesposabljanje nasprotnikov. Ker sem v tej točki vedel, da bodo tako glavni lik kot nasprotniki nekakšni roboti, sem si za sekundarno orožje zamislil nekakšen šoker, ki je namenjen onesposabljanju robotov. Poleg oborožitve sem si moral še zamisliti, kako se bo robota kontroliralo. Tukaj se je spet vmešala želja, da bi bilo v igri omogočeno skrivanje nasprotnikom. Potreboval sem dva načina premikanja. V prvem, navadnem načinu, je robot sposoben vsega, vendar ga stražarji hitro opazijo oz. slišijo. V drugem, prikritem načinu, pa se lahko tihotapi mimo stražarjev, saj ga ne slišijo. Cena neslišnosti je onemogočena uporaba glavnega orožja. Naredil sem tudi seznam potrebnih kontrol in kako jih bom razporedil po tipkovnici.

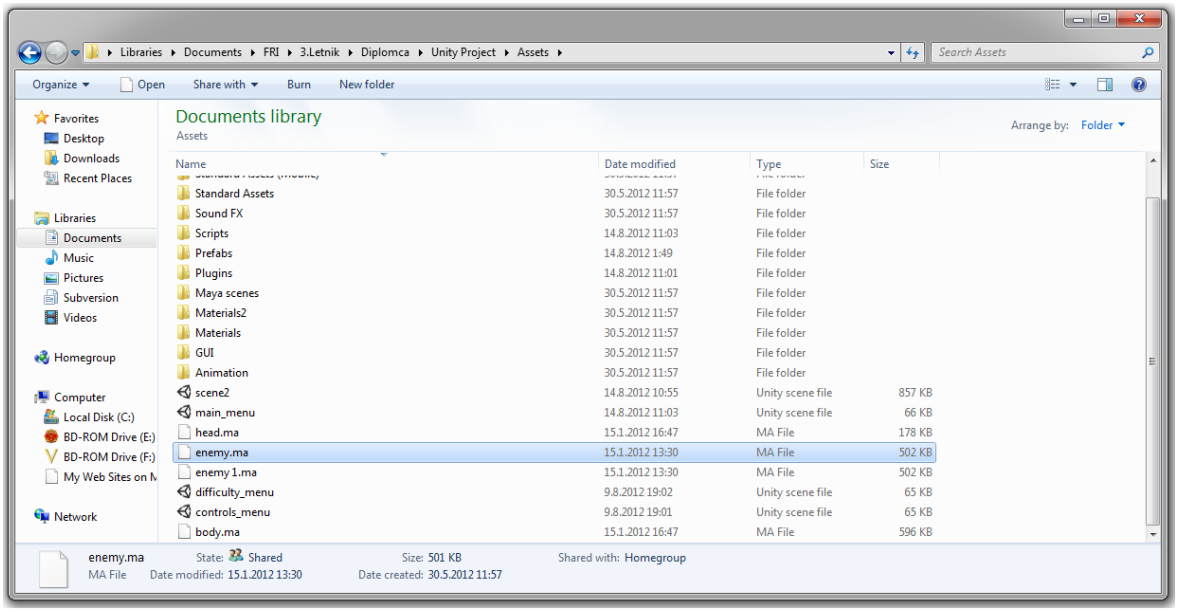
O nasprotnikih v samem načrtu nisem veliko napisal, vedel sem le, da potrebujem neke generične stražarje.

Na koncu načrta sem grobo skiciral razporeditev mape in označil, kje na mapi bodo postavljene kakšne pomembne točke.

4.3 Implementacija

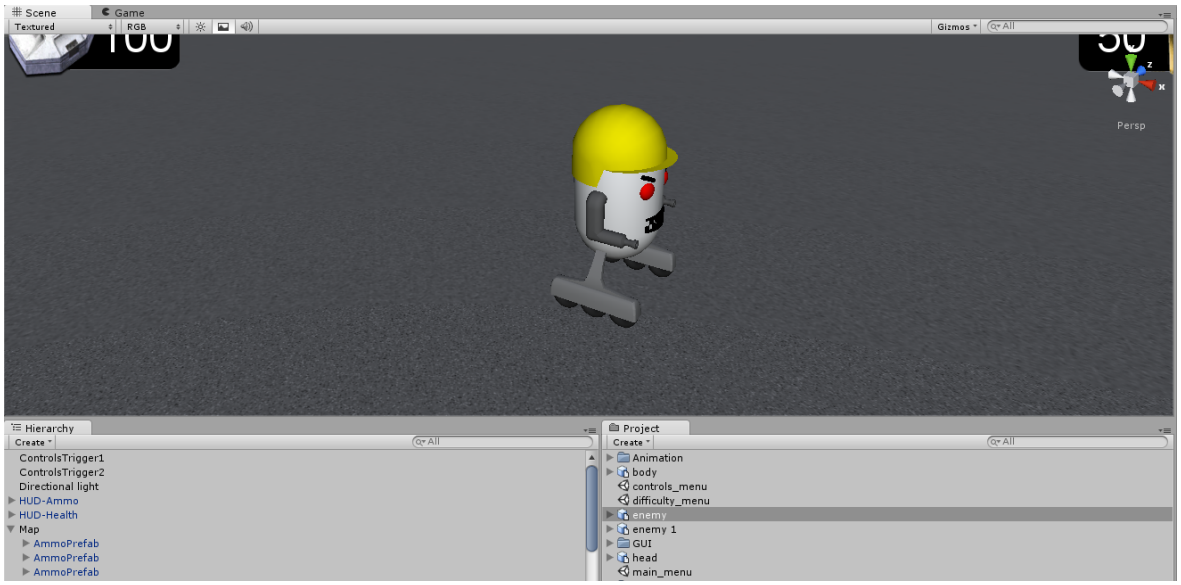
Najzanimivejši del implementacije je bila gotovo izdelava robotov stražarjev, zato bi rad ta del postopka opisal nekoliko natančneje.

Postopek se začne z izdelavo modela. Model je izdelan v programu Maya. Uvoz modela v unity je zelo preprost. Datoteko z modelom premaknemo v mapo s sredstvi našega projekta (naš_projekt/Assets). (slika 5.1)



Slika 4.1: Uvoz modela v Unity.

Model v samem Unity-ju je potrebno samo še prestaviti iz našega projektnega okna v svet igre. To je vse, kar je v Unity-ju potrebno narediti za uvoz modela. (slika 5.2)

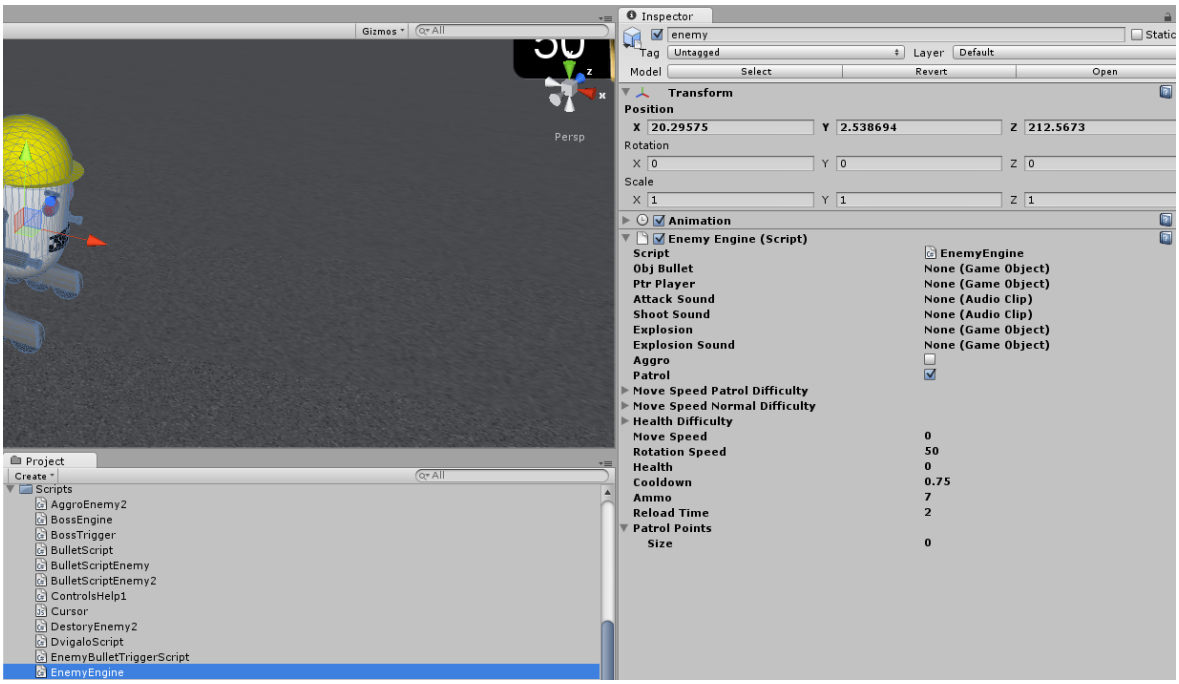


Slika 4.2: Naš model v svetu igre.

Končali smo z uvozom modela, vendar je naš robot zaenkrat prav to, samo model. Sedaj je potrebno robotu dodati še "možgane" in druge dodatke, za katere želimo, da jih robot ima (npr. luči).

Da lahko našemu robotu spišemo kodo, po kateri se bo ravnal, je modelu najprej potrebno dodati skripto, v katero bomo zapisali to programsko kodo. V Unity-ju je dodajanje skripte prav tako preprosto opravilo. V projektnem oknu ustvarimo novo skripto in jo poimenujemo. Ob kreaciji skripte je že potrebno določiti, v katerem programskem jeziku želimo skripto napisati. Unity podpira tri programske jezike, in sicer Javascript, C# in Boo script. V svojem projektu sem uporabljal le Javascript in C#, saj sem v teh jezikih že prej programiral.

Za tem je potrebno le klikniti naš uvožen model in skripto povleči v okno z lastnosmi trenutno izbranega objekta (inspector). (slika 5.3)



Slika 4.3: Dodajanje skripte.

Skripto nato odpremo v poljubnem urejevalniku kode, tudi Unity ima že vgrajen urejevalnik kode (MonoDevelop), ki pa je precej povprečen, vendar začetnikom ponuja povsem dovolj.

Koda, ki kontrolira robota, obsega vse skupaj 220 vrstic. Za predstavo, kako poteka skriptiranje v Unity-ju, bom opisal kodo, ki določa obnašanje robota v mirnem stanju (robot patruljira med vnaprej določenimi točkami na mapi). Skriptiranje v Unity-ju se največ vrti okoli funkcije Update(). Ta funkcija se izvede enkrat za vsako novo sliko (ang. frame).

```
45      //spremenljivke patrol
46      public GameObject[] patrolPoints;
47      private GameObject patrolPoint2;
48      private int currentPath = 0;

76      void Update () {
77          if (health <= 0 || transform.position.y < -7.5f) {
78              destroyEnemy();
79          }
116         if (aggro == true) {
117             NapadiPlayerja();
118         }
119         if (patrol == true) {
120             Patruliraj();
121         }
122         //}
123         }
124     }

126     void Patruliraj() {
127         Vector3 vecToPP = patrolPoint2.transform.position - transform.position;
128         float distanceToPP = Vector3.Dot(vecToPP, vecToPP);
129         if(distanceToPP > 1f) {
130
131             Vector3 smerToPP = transform.InverseTransformPoint(patrolPoint2.transform.position);
132             if (smerToPP.x > 0.6) {
133                 Levo();
134             } else if (smerToPP.x < -0.6) {
135                 Desno();
136             } else {
137                 MoveEnemy();
138             }
139         } else {
140             currentPath += 1;
141             currentPath = currentPath % patrolPoints.Length;
142             patrolPoint2 = patrolPoints[currentPath];
143             Desno();
144         }
145     }
```

Slika 4.4: Koda v skripti.

Z rdečo so označeni deli kode, ki se nanašajo na upravljanje robota v mirnem stanju. Mirno stanje določa spremenljivka “boolean patrol“, ki je na začetku avtomatsko postavljena na “true“. Vidimo, da sta dve spremenljivki na začetku kode tipa “GameObject“. “GameObject“ je vsak objekt, ki ga postavimo v naš svet igre v Unity-ju. Prva spremenljivka “patrolPoints“ je javno dostopna (“public“), kar pomeni, da jo lahko urejamo v našem svetu v Unity-ju. Na tem seznamu ima vsak dokončan robot točke med katerimi mora opravljati svojo patruljno pot. Točke so pravzaprav le prazni “GameObject-i“, ki jih lahko v svetu prosto postavljamo in premikamo. V funkciji Update(), ki se, kot sem že prej povedal, izvede enkrat ob vsaki

novi sliki, robot pogleda ali mora trenutno patruljirati ("patrol == true") in začne izvajati kodo v funkciji Patruliraj(). V tej funkciji si najprej naredi vektor med trenutno točko, do katere je namenjen in svojo pozicijo. Iz tega vektorja si izračuna oddaljenost. Če je od točke oddaljen več kot eno enoto, nadaljuje s potjo proti točki, drugače pa pogleda, kje je naslednja točka in si jo nastavi za nov cilji.

4.4 Testiranje

Po mojem mnenju je testiranje igre najpomembnejši del pri izdelavi kvalitetne igre. Zato sem sam opravil ogromno testiranja. Že med samo izdelavo igre je potrebno konstantno preverjati objekte, skripte, itd., če delujejo tako, kot smo si želeli. Ta del testiranja je najbolj preprost, saj točno vemo, kateri del kode oziroma kateri objekt je pod drobnogledom.

Ko igro dokončamo, je potrebno še ogromno testiranja, da odkrijemo vse hrošče in jih odpravimo. Tukaj je stvar malo težja, saj pri hroščih navadno ne vemo, kaj natančno jih povzroča. Sam sem največ časa porabil za odpravljanje hrošča, ki je v samo enem zelo majhnem delu igre pokvaril delovanje merilnika.

Igro je bilo prav tako potrebno preizkusiti na različnih računalnikih. Tukaj sem doživel presenečenje. Na mojem računalniku je igra delovala brezhibno, ko pa sem jo preizkusil na malce starejšem računalniku, jo je le-ta komaj poganjal. Ker sem želel, da lahko igro poganjajo tudi starejši računalniki, sem moral to popraviti. Na koncu se je izkazalo, da sem močno pretiraval z osvetlitvijo. V igri so bile sobe, ki so imele tudi več kot deset luči. To pa je predstavljalo problem, saj sem luči nastavil tako, da je vsaka luč za vsak osvetljen objekt računala in izrisovala tudi senco. Toliko računanja pa malce starejši računalnik ni bil sposoben opraviti dovolj hitro. Težavo sem rešil tako, da sem nekaj luči odstranil, ostalim pa nastavil preprostejšo tehniko senčenja. Po tem popravku je igra delovala brezhibno tudi na štiri leta starem notesniku.

4.5 Cry Engine 3

Osnovni projekt diplomskega dela sem naredil v Unity-ju, vendar sem za boljše spoznavanje obeh pogonov za igre, kratko igro naredil tudi v Cry Engine-u. Igra je zelo preprosta prvoosebna streljanka, v kateri branimo zapuščeno tovarno pred napadalci, ki jih vodi računalnik. Cilj igre je ubiti čim več napadalcev, preden smo sami uničeni.

Glavni razlog za izdelavo te igre je bila želja po spoznavanju skriptiranja. V Cry Engine-u uporabljamo vizualno skriptiranje, poimenovano "Flow Graph".

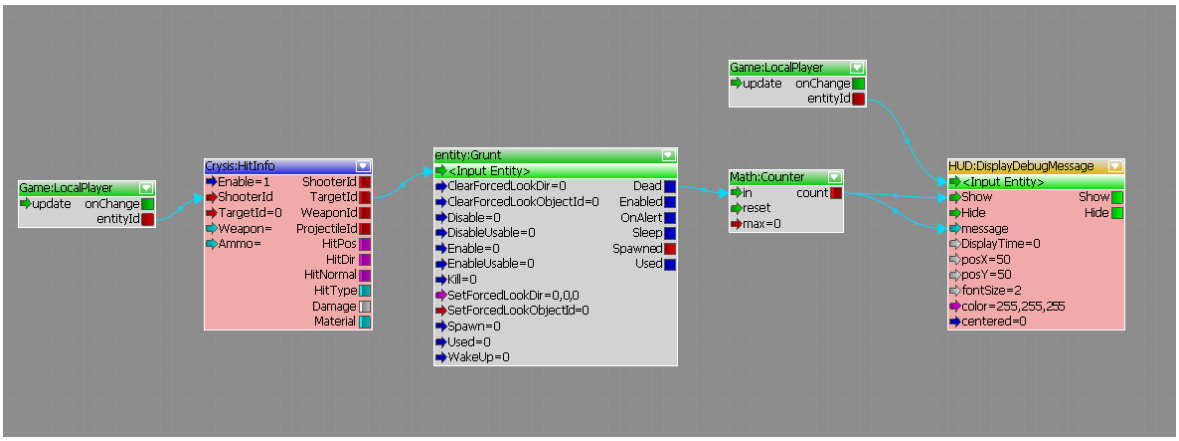
4.5.1 Flow Graph

V igri sem s pomočjo Flow Graph-a sprogramiral števec ubojev (ang. kill counter), katerega bom malo bolj podrobno opisal.

Slika 4.5: Števec ubojev.

Pri uporabi Flow Graph-a delamo s tabelami. Vsaka tabela predstavljaja svojo entiteto in ima vhode (na levi strani) ter izhode (na desni strani). Izhode in vhode tabel ustrezno povežemo med sabo, tako da dobimo delujočo skripto.

Za realizacijo števca, sem potreboval šest tabel (pet unikatnih). Prva tabela, "LocalPlayer", predstavlja nas, druga tabela je informacija o zadetku ("HitInfo"), ki je tudi glavni del logike našega grafa, tretja tabela predstavlja naše nasprotnike ("Grunt"), četrta tabela je števec ("Counter"), peta tabela je sporočilo, ki se nam izpiše na zaslonu ("DisplayDebugMessage"). Logika števca je precej preprosta. Vsakič, ko ubijemo nasprotnika, naj se števec poveča za ena in sporočilo izpiše na zaslonu.



Slika 4.6: Programiranje s Flow Graph-om.

V grafu je to predstavljeno s povezavami. Vhod “ShooterId“ (Id strelca) tabele “HitInfo“ povežemo z izhodom iz tabele, ki predstavlja nas. Tako dobi tabela “HitInfo“ referenco na nas. Izhod “TargetId“ (Id tarče) povežemo z vhodom “Input Entity“ tabele, ki predstavlja naše nasprotnike. Trenutni graf je ekvivalenten kodi, ki bi vsakič, ko zadanemo nasprotnika, prožila nek dogodek. Ker nas zanimajo le primeri, v katerih naša tarča tudi umre, uporabimo iz tabele “Grunt“ izhod “Dead“. Ta izhod lahko povežemo direktno na vhod našega števca, ki bo sedaj zabeležil vsako smrt sovražnika. Vse kar je še potrebno je, ob povečanju števca prikazati sporočilo z njegovo vsebino, za kar služita povezavi med števcem in tabelo s sporočilom.

Poglavje 5

Zaključek

Med delom diplomske naloge sem si ustvaril svoje mnenje o pogonih za igre in uporabi le-teh. Mislm, da sama uporaba pogona za igre sploh ni vprašljiva. Če želimo narediti sodobno igro, bo v skoraj vseh primerih bolje uporabiti nek pogon, kot pa igro razvijati čisto sam. Bolj pomembno se mi zdi vprašanje, kateri pogon je primeren za določeno igro.

V veliki večini primerov bo odgovor na to vprašanje Unity. Unity je bolj prijazen do začetnikov in bolj spodbuja ustvarjalnost. Vsakdo, ki se loti razvoja igre, bi rad realiziral neko svojo idejo in največkrat to ni prvoosebna igra, kot je to značilno za Cry Engine. Prav tako ima Unity veliko prednost na področju alternativnih platform. Danes so vedno bolj popularne igre za pametne telefone, katerih se v Unity-ju lahko lotimo enako kot katerekoli druge igre, Cry Engine pa teh platform sploh ne podpira.

Seveda pa to ne pomeni, da je Cry Engine neuporaben. Če je naša ideja prav prvoosebna igra, in če smo pripravljeni vanjo vložiti malo več truda, potem bo končni rezultat mnogo lepši z uporabo Cry Engine-a in vseh njegovih naprednih funkcij.

Če bi moral izbrati zmagovalca, bi se odločil za Unity, saj sem mnenja, da preprostost in prilagodljivost Unity-ja odtehtata tehnično dovršenost in naprednost Cry Engine-a.

Moje izkušne z Unity-jem so pozitivne in so v skladu z večino mnenj, ki

sem jih našel na internetu. Zelo preprosto je začeti delati z njim, sam sem namreč že po dveh dnevih dela razumel Unity, po dveh tednih pa se mi je zdelo, da že znam uporabiti večino njegovih funkcionalnosti. Menim, da je Unity pravi pogon za igre samostojnih razvijalcev.

Literatura

[1] Watt Policarpo, "3D Games"

[2] CryEngine 3, http://devmaster.net/devdb/engines/cryengine-3

[3] Game Engine Comparison, http://www.esenthel.com/?id=compare

[4] Jeff Ward, "What is a game Engine?", http://www.gamecareerguide.com/features/529/what_is_a_game.php?page=1

[5] Unity 3 Review, http://www.doolwind.com/blog/unity-3-review/

[6] CryEngine 3 Review, http://forum.unity3d.com/threads/106166-Yes-its-about-CryEngine-3-again...

[7] CryEngine 3 Features, http://www.pcgameshardware.com/aid,679921/Crytek-Features-of-the-Cryengine-3-explained/News/?page=2