Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Ivan Antešić

Razvoj računalniške igre z vidika neodvisnega razvijalca

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Narvika Bovcon

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge: Razvoj računalniške igre z vidika neodvisnega razvijalca

V diplomskem delu opišite proces razvoja računalniške igre na lastnem primeru. Naštejte in kratko opišite potrebna orodja za razvoj ter zakaj ste izbrali ravno ta orodja. Podajte osnovne principe oblikovanja glavnih vidikov računalniških iger (igralnosti, izgleda, zvoka, tehnološke izvedbe) in pojasnite odločitve, ki ste jih sprejeli pri izdelavi svoje igre.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Pregled področja 2

1.1.1 Industrija 2

1.1.2 Gradivo 4

1.2 Metode razvoja programske opreme 4

1.2.1 Agilne metode 4

1.2.2 Izbrana metoda 5

2 Orodja **7**

2.1 Igralni pogon - Unity 7

2.2 Obdelava grafike - Pyxel Edit 9

2.3 Obdelava zvoka - Sfxr 11

3 Predprodukcija **15**

3.1 Dokument z idejo igre 16

3.2 Dokument s konceptom igre 17

3.2.1 Cilji 17

3.2.2 Motivacija igralca 18

3.2.3 Ciljna skupina 20

3.2.4 Žanr 21

3.2.5 Starostna ocena 23
3.2.6 Analiza konkurence 23
Shovel Knight 24
Risk of Rain 24
Množica iger 24
3.2.7 Ostale točke 25
3.3 Dokument z načrtom igre 25
3.3.1 Igralnost 25
Temeljne igralne mehanike 26
Mehanike igralca 27
Oblikovanje nivoja 28
Mehanike sovražnikov 29
3.3.2 Zgodba 30
Struktura zgodbe 30
Zgodba v moji igri 31
Krivulja zanosa 32
3.3.3 Vizualni slog 34
Določanje sloga moje igre 34
Pixel art slog 37

4 Produkcija **41**
4.1 Tehnološki vidik 41
4.1.1 2D igra v razvojnem okolju Unity 43
4.1.2 Simulacija prostorskih planov 48
4.1.3 Igralec 48
4.1.4 Nadgradnje 51
4.1.5 Sovražniki 52
4.1.6 Gospodar igre 55
4.1.7 Uporabniški vmesnik 56
4.2 Izgled 57
4.2.1 Omejitve 57
4.2.2 Risanje 60

4.2.3 Animacija 65
4.3 Zvok 65
4.3.1 Zvočni efekti 66
4.3.2 Glasba 66

5 Zaključne faze **69**
5.1 Alfa faza 69
5.2 Beta faza 69
5.3 Zlata faza in poprodukcija 70

6 Zaključek **71**

Povzetek

Naslov: Razvoj računalniške igre z vidika neodvisnega razvijalca

Avtor: Ivan Antešić

Razvoj igre obsega veliko različnih področij - programiranje, principe računalniške grafike in pripadajoče matematike. Poleg tega je igro potrebno ustrezno oblikovati, tako igralno kot vizualno, ustvariti glasbeno spremljavo ter zvoke. Pomembna sta tudi originalna ideja in kreativno razmišljanje, da naša igra izstopa na trgu. Naloga neodvisnega razvijalca je, da poskrbi za vse vidike igre in jih združi v zanimivo celoto, ki pritegne ter obdrži igralca. Diplomsko delo demonstrira celoten proces razvoja računalniške igre. Opisana so potrebna orodja in ustaljene razvojne faze ter zakaj in kako so bile uporabljene pri kreiranju igre.

Ključne besede: računalniška igra, 2D računalniška grafika, Unity, oblikovanje.

Abstract

Title: Game development from the perspective of an independent developer

Author: Ivan Antešić

Game development covers a spectrum of different areas - if we want to create a game we have to be a skillful programmer and be familiar with the principles of computer graphics and its mathematical fundations. We also need to design a game, both in terms of gameplay and visual style, create sound effects and the soundtrack for the game. Having an original idea and thinking creatively is also a key part of game development, so that our game stands out from all others that are available on the market. An independent game developer has to take care of all these aspects and join them together in an interesting game that will attract and motivate our players. This work demonstrates the process of creating a video game. It describes required tools, standard game development steps and explains why and how they were used during this process.

Keywords: computer game, 2D computer graphics, Unity, design.

Poglavje 1

Uvod

Računalniške igre so me zanimale že od otroštva. So tudi eden izmed glavnih razlogov, da sem se odločil za študij računalništva in informatike. V okviru študija sem se seznanil z razvojem preproste igre, vendar ne z vsemi pripadajočimi vidiki. Predvsem smo se usmerili na principe računalniške 3D grafike, v okviru diplomske naloge pa sem želel spoznati še ostale vidike, ki vplivajo na igro kot celovit izdelek.

V okviru dimplomske naloge bom razvil lastno igro ter opisal proces razvoja z vidika neodvisnega razvijalca. Menim, da je tema aktualna, saj danes bolj kot kadarkoli prej narašča število iger, ki jih razvijajo neodvisni razvijalci: zaradi dostopnosti tehnologije ima namreč vsakdo s potrebnim znanjem možnost, da izda svojo igro.

Odločil sem se za izdelavo ploščadne 2D igre - žanr, v katerem je igralec omejen na premikanje v dveh dimenzijah, objekti so namesto iz 3D modelov narejeni iz ploskih sličic, kamera pa nam omogoča stranski pogled na sceno. Tipičen primer takšne igre je Super Mario Bros (Nintendo, 1985). Razlog za takšno odločitev sta preprostejša izdelava in osebno zanimanje za starejše 2D igre. Končna igra je objavljena na mojem profilu, na spletni strani itch.io [1].

Diplomska naloga bo tako pilotna študija oz. demonstracija z vsemi fazami in vidiki razvoja ter predstavitvijo potrebnih orodij. Ugotovitve iz

diplomske naloge bodo koristile vsem, ki se lotevajo razvoja igre, saj poleg predstavitve celotnega procesa opozorim tudi na pomembne podrobnosti.

1.1 Pregled področja

1.1.1 Industrija

Področje računalniških iger je zelo široko. Obsega vse od preprostih mobilnih iger, ki so na voljo na spletnih trgovinah Google Play ali Apple App Store, pa do več miljard vredne industrije, ki se lahko meri s filmsko industrijo. Celotna filmska industrija (vključno z DVD, TV serijami, itd.) je bila leta 2016 vredna 286.17 milijard dolarjev, če se omejimo na kinematografe pa 38.3 milijard dolarjev [2]. V primerjavi je bila industrija iger leta 2017 v celoti vredna 108.9 milijard dolarjev do leta 2020 pa je predvideno, da bo narasla na 128.5 milijard dolarjev [3].

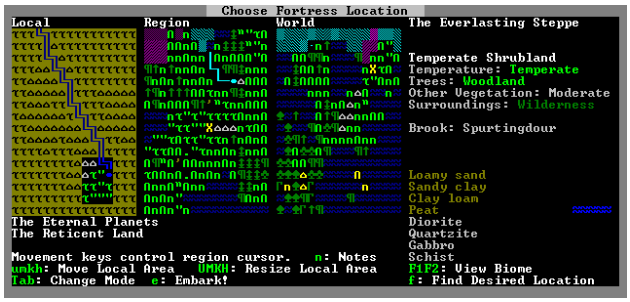
Ker je industrija iger tako narasla, se je povečala tudi velikost podjetij in založnikov. Danes so igre, ki jih izdajajo nekatera podjetja, kot na primer Activision ali Ubisoft, napredni tehnični izdelki, ki jih razvijajo ekipe več sto ljudi z različnimi znanji iz različnih področij. Te igre zahtevajo tudi veliko časa in denarja za razvoj.

Za lažjo predstavo vzemimo primer velikega podjetja, ki razvija veliko igro (tako imenovano AAA igro - igro z napredno grafiko in ogromnim proračunom, na primer igra GTA V, vidna na sliki 1.1). Za razvoj takšne igre potrebujemo vsaj 400 zaposlenih in tri leta časa. Založniki računajo, da bo razvoj igre stal povprečno 10000 dolarjev mesečno na zaposlenega. To vključuje plačo, najem prostorov, opremo in druge nepričakovane stroške. Tako za razvoj porabimo 144 milijonov dolarjev in to brez upoštevanja cene oglaševanja, ki je lahko enaka ali večja od cene razvoja [4].

Seveda so to produkti, ki jih manjša skupina neodvisnih razvijalcev z manjšimi sredstvi ne more narediti. Kljub temu je v zadnjem času zaradi dostopnosti tehnologije narastlo število neodvisnih razvijalcev - INDIE (skrajšano angl. independent) razvijalcev, povečalo se je tudi povpraševanje

Slika 1.1: Primer velike igre - GTA V (Rockstar North, 2013). Cena razvoja igre je bila 137 miljonov, cena oglaševanja pa 128 miljonov dolarjev [5].

po igrah, ki jih razvijajo. Čeprav takšni razvijalci nimajo finančne podpore založnika, imajo ravno zaradi tega več kreativne svobode - prosti so, da naredijo takšno igro, kot si želijo. Namesto visoko razvite tehnične plati iger se osredotočijo na originalne in zanimive ideje ter tako tekmujejo z večjimi produkcijami. Takšna igra je tudi Dwarf Fortress, (slika 1.2), napredna simulacija fantazijskega sveta, ki sploh nima grafike - vse je predstavljeno z ASCII znaki.



Slika 1.2: Primer INDIE igre - Dwarf Fortress. Igro razvijata brata Tarn in Zach Adams le s pomočjo denarnih prispevkov igralcev, sicer pa je igra zastonj [6].

1.1.2 Gradivo

Zaradi velikega razmaha na tem področju je izšlo veliko knjig in člankov, ki opisujejo postopek razvoja igre. Ta dela se lahko osredotočijo na oblikovanje ideje igre [7], pripadajočo matematiko [8], programiranje [9], pogone [10] ali pa na celoten pregled razvoja [11] ter splošne nasvete glede izvedbe takšnega obsežnega projekta [12]. Nekaj sorodnih diplomskih nalog so naredili že moji kolegi prejšnih let, nobeden od njih pa ni poskusil razviti računalniške igre v celoti in raziskati vseh pripadajočih vidikov in faz razvoja.

1.2 Metode razvoja programske opreme

Računalniške igre se, kot vso ostalo programsko opremo, razvija po vnaprej definiranih metodah razvoja [13]. Še posebej so primerne agilne metode razvoja programske opreme, namesto bolj linearnih, na primer slapovnega modela.

1.2.1 Agilne metode

Agilne metode želijo omogočiti prilagodljivost in sprotno reševanje problemov. Osnovane so na vrednotah, določenih v Manifestu za agilni razvoj programske opreme [14] :

1. Komunikacija in sodelovanje med razvijalci sta bolj pomembna kot individualno znanje posameznika, ki dela v samoti.

2. Delujoča programska oprema je bolj pomembna kot obsežna dokumentacija, ki vzame veliko časa in z nadaljevanjem projekta kmalu zapade.

3. Pomembno je vljkučiti uporabnika v razvojni cikel tako, da se produkt lahko prilagaja med samim razvojem.

4. Agilne metode so bolj naravnane na to, da se hitro odzivajo na spremembe, kot pa da sledijo začetnemu načrtu.

Dokumentacija in planiranje je še vedno pomemben del agilnih metod, vendar le toliko, da pomaga razvijalcem, ne da jih ovira. Bolj znane agilne metode so na primer SCRUM, Kanban in AUP.

1.2.2 Izbrana metoda

Pri svoji igri sem se zgledoval po vrednotah agilnega razvoja programske opreme. Nisem pisal obsežne dokumentacije, saj je kot edini razvijalec igre nisem potreboval za sporočanje napredka in sprememb ostalim članom. Vseeno pa menim, da je dokumentacija pomembna, saj sem si z njo lažje zastavil cilje in lastnosti igre, pomagala mi je predvsem pri oblikovanju igre. Torej dokumentacijo sem pisal le takrat, ko mi je to koristilo, in ne samo zato, da sem dokumentiral razvoj brez razloga.

Prav tako se nisem omejil na točno določen model razvoja programske opreme in natanko sledil pripadajočemu načrtu. Ker sem se z razvojem iger srečal prvič, sem predvideval, da bom zagotovo naletel na ovire in zahteve, ki jih na začetku ne morem predvideti. Zato sem se odločil za kombinacijo iterativnega in inkrementalnega modela, kjer sem iterativno razvijal določen vidik igre (npr. programiranje kontrol), ga postopoma nadgrajeval, vključil v prototip in kasneje po potrebi spet iterativno nadgradil ta del. Ko sem opazil problem, sem ga razrešil in nadaljeval z novo iteracijo, morda tokrat z drugimi cilji in zahtevami. Tako sem na naraven in fleksibilen način nadgrajeval prototip igre z različnimi funkcionalnostmi, dokler nisem bil zadovoljen s končnim izdelkom.

Poglavje 2

Orodja

Za razvoj igre je potrebne veliko različne programske opreme. V tem poglavju bom opisal glavne lastnosti in namen orodij za kreiranje igralnosti, grafike in zvoka ter zakaj sem se odločil za izbrano orodje.

2.1 Igralni pogon - Unity

Najbolj pomembna programska oprema za razvoj igre je igralni pogon. Igralni pogon je programsko ogrodje, ki razvijalcu omogoča vizualno uporabo več orodij v integriranem razvijalskem okolju (angl. Integrated development environment - IDE). Ta orodja vsebujejo komponente, ki omogočajo funkcionalnosti, ki jih potrebujejo razvijalci pri razvoju svoje igre [10]:

1. Grafični pogon za upodabljanje 3D ali 2D objektov v igri.
2. Fizikalni pogon za detekcijo trkov in simulacijo sil med objekti.
3. Podpora animiranja objektov.
4. Zvočni (avdio) pogon za predvajanje zvoka.
5. Skriptiranje za programiranje lastnosti objektov v igri.
6. Podpora za različna ciljna računalniška okolja (Xbox, PlayStation, iOS, Android, PC, Mac).

Vse komponente in sam igralni pogon lahko razvijalci sprogramirajo sami, kar v večjih podjetjih tudi naredijo. Takšni pogoni vzamejo veliko časa in denarja za razvoj, tako da za manjšo skupino neodvisnih razvijalcev ni primerno, da naredijo svoj pogon. Ponavadi takšna skupina uporabi že vnaprej narejen igralni pogon in v njem razvije želeni produkt.

V času pisanja te diplome je na voljo kar nekaj takšnih splošnih igralnih pogonov. Nekateri niso na voljo splošni javnosti ampak samo podjetjem, ki so ga razvili. Tak je na primer igralni pogon imenovan id Tech, ki ga je razvilo podjetje id Software, na voljo je le razvijalcem, ki spadajo v skupino ZeniMax Media. Med tistimi, ki so prosto dostopni (po različnih licencah), so najbolj znani Unity[15], Unreal Engine [16], Game Maker [17] in Cry Engine [18]. Za razvoj svoje igre sem se odločil, da bom uporabil pogon Unity.

Pogon Unity je od svojega izida leta 2005 pa do danes postal najbolj razširjen pogon v industriji iger. V lasti ima 45% trga računalniških pogonov, uporablja pa ga več razvijalcev kot katerikoli drug pogon [19]. Čeprav je verjetno velik delež iger, razvitih z Unity, preprostih mobilnih iger, obstajajo tudi igre, ki so jih izdelala velika podjetja, kot na primer Activision Bllizard (HearthStone) in Obsidian Entertainment (Pillars of Eternity).

Pogon je v industriji zelo razširjen. Velik vpliv na to ima dejstvo, da je bil pogon Unity prvi izmed naštetih pogonov, ki je imel na voljo brezplačno licenco ter si je tako zagotovil velik delež neodvisnih uporabnikov. Velika skupnost omogoča lažjo uporabo, še posebej začetnikom, saj lahko morebitno težavo razrešiš z vprašanjem na uradnem forumu Unity Answers. Unity ima tudi največjo spletno tržnico sredstev (angl. Asset Store), kjer lahko kupujemo in prodajamo sredstva (angl. assets) - različne vsebine, ki so jih naredili drugi uporabniki, ki lahko novemu razvijalcu prihranijo čas in olajšajo delo. Uporabniku prijazna je tudi obsežna dokumentacija orodij, jezika in učnih videov. Uporabniški vmesnik, prikazan na sliki 2.2, omogoča preprosto kreiranje in spreminjanje lastnosti objektov.

Čeprav s pogonom ni možno narediti tako napredne grafike kot s Cry

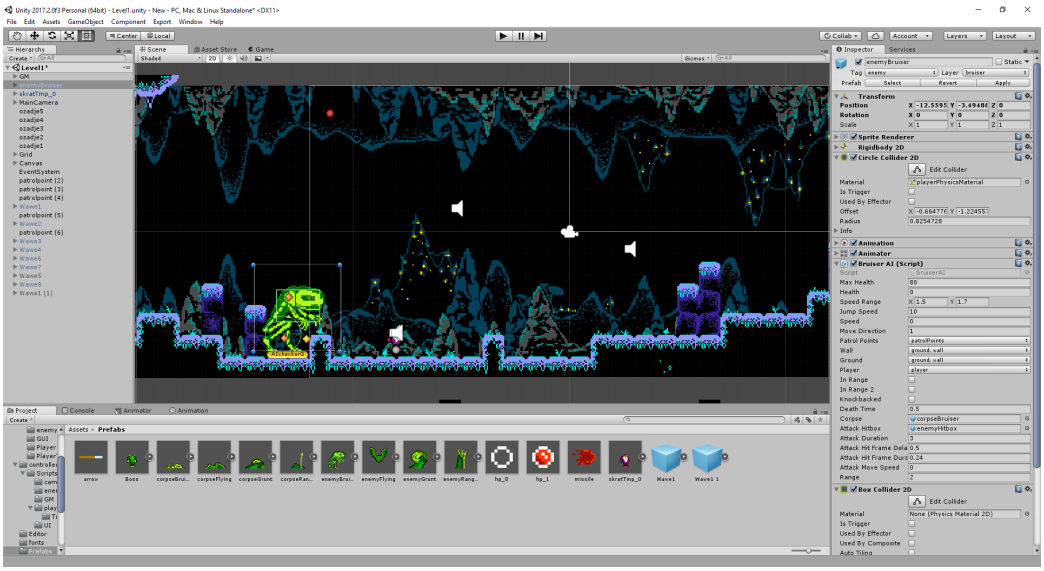
Enginom in nima vseh specialnih orodij za kreiranje 2D igre, ki jih ima GameMaker Studio, ter nima toliko funkcionalnosti kot Unreal Engine, je glavna prednost Unity pogona njegova splošnost. Je primeren za vse vrste iger, saj tudi če nima določene funkcionalnosti, ki bi pomagala razvijalcu, jo morda lahko najdemo na tržnici sredstev ali pa jo zaradi intuitivnega urejevalnika in dobro zasnovane arhitekture dodamo v pogon sami. Razvito igro lahko brez spreminjanja izdamo na več kot 25 računalniških okoljih vključno s PC, Mac, Android, iOS, PS4 in Xbox One, kar je zelo koristno pri prodaji izdelka. Skriptiranje je možno v jeziku C# ali pa v JavaScriptu, čeprav se podpora za ta jezik ukinja.

Za pridobitev pogona je na voljo več licenc: Unity Personal je zastonjska različica, ki jo podjetje lahko uporablja le, če zasluži manj kot 100.000 dolarjev letno. Naslednja različica se imenuje Unity Plus, ki stane 35 dolarjev mesečno in ima dostop do funkcionalnosti, s katerimi sledimo zmogljivosti aplikacije, ki jo razvijamo. To različico lahko uporabljamo, dokler ne presežemo 200.000 dolarjev dobička letno. Zadnja različica je Unity Pro, ki za ceno 125 dolarjev mesečno omogoča dodatne funkcionalnosti, kot so delo v oblaku, podpora večjega števila spletnih igralcev in nakupovanja v igri, itd. Ta različica nima omejitve na zaslužek podjetja.

Zaradi naštetih lastnosti se mi je pogon Unity zdel najboljša odločitev za razvoj moje igre. Osnovna različica je zastonj, dodatne funkcionalnosti ostalih različic pa niso tako pomembne za razvoj načrtovane igre. Igre ne bom prodajal, zato me ne skrbi, da bi presegel pogoje licence. Pritegnilo me je tudi dejstvo, da podpira skriptiranje v uveljavljenem „pravem“ programskem jeziku, kot je C#. Imel sem znanje programiranja v C in C++, s tem projektom pa bi pridobil še izkušnje s C#.

2.2 Obdelava grafike - Pyxel Edit

Pri razvoju igre so pomembna tudi orodja za obdelavo grafike. Potrebujemo jih za različne naloge, na primer za risanje ozadij in tekstur za 3D modele,



Slika 2.1: Uporabniški vmesnik Unity 2017.1. Zasnovan je tako, da se v njem lahko premikamo kot v običajni prvoosebni igri, s klikom na objekt pa dobimo na razpolago vse njegove lastnosti in parametre.

z njimi kreiramo vizualni slog in izgled končnega produkta. Sam bom potreboval orodje, ki mi bo predvsem omogočalo hitro in učinkovito risanje 2D rasterskih slik z majhno ločljivostjo, ki bodo predstavljale objekte na zaslonu. V računalniški grafiki je takšen tip slike imenovan „sprite“. Za obdelavo grafike sem se odločil za uporabo orodja Pyxel Edit [11].

Pyxel Edit [20] je ena od aplikacij, posebej namenjenih kreiranju *spritov* nizke ločljivosti v slogu Pixel Art (opisan v poglavju 3.3.3 Vizualni slog). Aplikacijo je razvil Daniel Kvarfordt, licenca pa je na voljo za plačilo 9 dolarjev. Vsebuje orodja, s katerimi je to delo olajšano in učinkovito:

1. Orodje za kreiranje ploščic (angl. tiles), s katerimi nato v igralnem pogonu gradimo okolje.

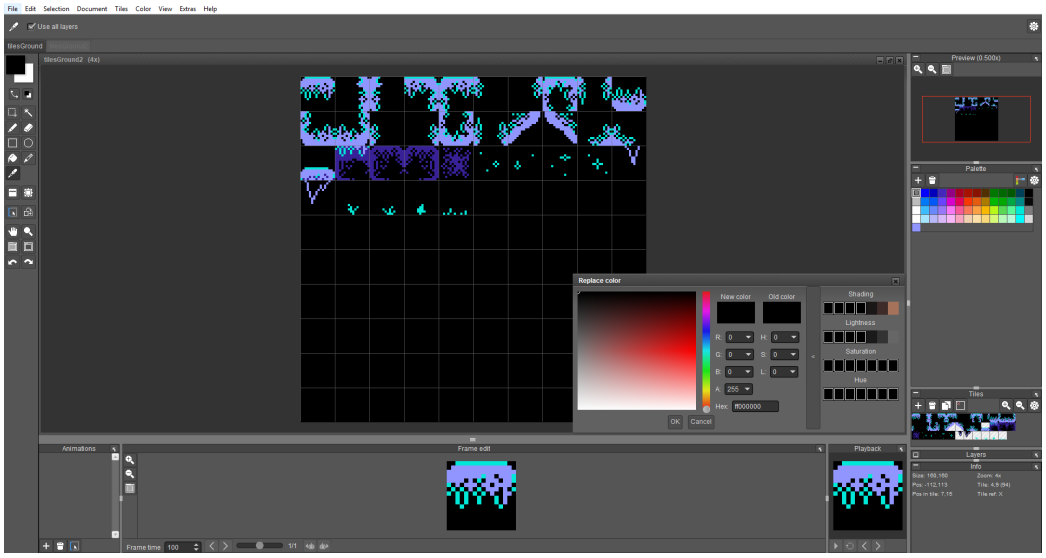
2. Orodje za kreiranje animacij. V urejevalniku medtem, ko rišemo posamezne sličice, že vidimo predogled animacije.

3. Podpira več načinov izvoza končnih sličic - v posameznih datotekah ali pa vse na eni sliki z določenim razmakom. To uporabniku prihrani ogromno časa pri uvažanju teh istih slik v igralni pogon.

4. Orodje za barvne palete omogoča, da naredimo svojo barvno paleto ali pa izberemo barve, ki nam jih priporoča program glede na izbrani slog.

5. Vsa ostala orodja so prilagojena kreiranju grafike v Pixel art slogu.

Poleg tega je uporabniški vmesnik podoben tistemu v Photoshopu, tako da sem lahko prenesel svoje znanje in izkušnje s Photoshopom v Pyxel Edit ter nemudoma postal domač z urejevalnikom. Te lastnosti in cena licence so me prepričale, da sem izbral Pyxel Edit kot aplikacijo za obdelavo grafike v moji igri.



Slika 2.2: Uporabniški vmesnik orodja Pyxel Edit.

2.3 Obdelava zvoka - Sfxr

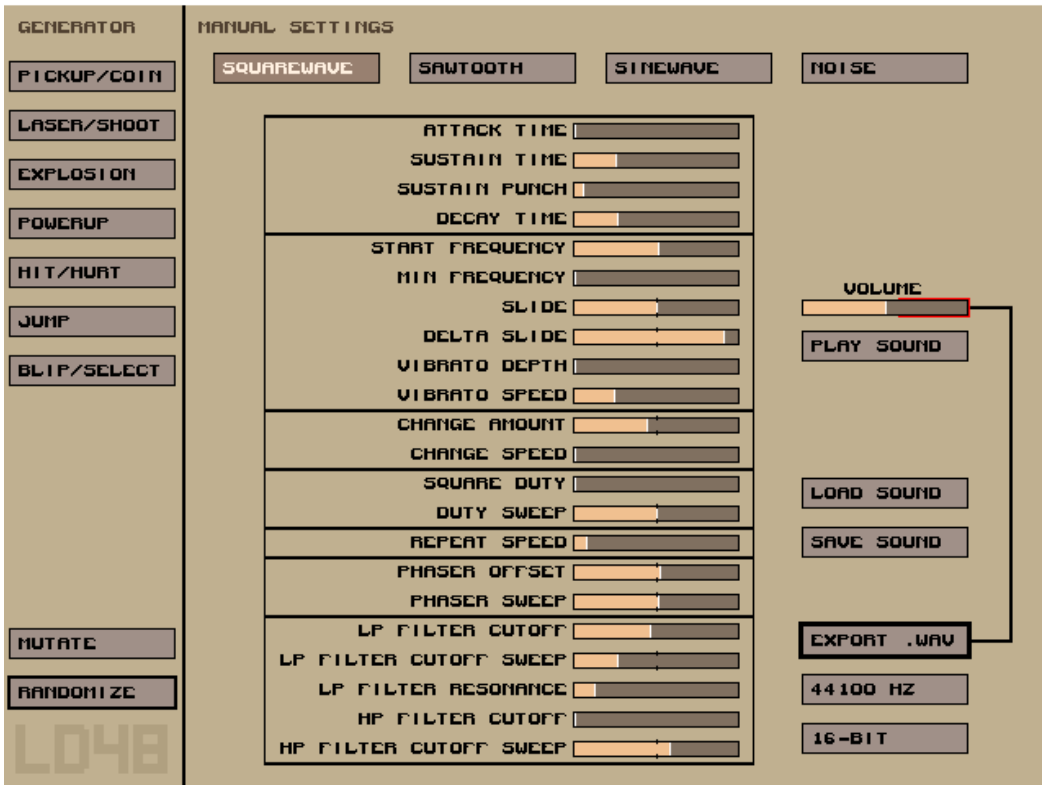
Zvok je eden od pomembnih delov igre. Obstaja več orodij, namenjenih izdelavi zvočnih efektov ali glasbe. Med njimi sta razširjena FL Studio [21],

orodje namenjeno izdelavi skladb, in Audacity [22], s katerim lahko urejamo posnete zvoke. Ker ne bom ustvaril lastne glasbene spremljave in sam posnel zvočnih efektov, sem se odločil za uporabo preprostejšega orodja Sfxr.

Sfxr je leta 2007 razvil Tomas Pettersson, da bi omogočil udeležencem izziva Ludum Dare (tekmovanje, kjer morajo razvijalci narediti igro v 48 urah) hiter način generiranja preprostih zvočnih efektov. Program je kmalu postal priljubljen med neodvisnimi razvijalci zaradi brezplačne licence in učinkovite ter preproste uporabe. Uporabljajo ga marsikatere uspešne igre, na primer. Spelunky in Tiny Barbarian.

Sfxr je virtualni modularni sintetizator - orodje, kjer s spreminjanjem parametrov zvočnega signala generiramo različne zvoke. Najprej izberemo eno izmed treh valovnih oblik signala, npr. sinusoido (možna je tudi izbira šuma), ter nato spreminjamo amplitudo, frekvenco, hitrost in uporabimo različne filtre, s katerimi spreminjamo obliko signala in poslušamo nastale zvoke. Če ne želimo nastavljati parametrov vsakega posebej, lahko izberemo tudi generatorje, ki imajo vnaprej določene omejitve pri spreminjanju parametrov, in tako naključno produciramo zvoke, podobne efektom za eksplozijo, strel, skok in podobno. Ko smo zadovoljni z zvokom, ga lahko shranimo ali izvozimo v WAV format in ga kasneje dodamo v igro. Orodje je na voljo kot spletna aplikacija ali namizna aplikacija za macOS in Windows operacijske sisteme.

Uporabniški vmesnik je prikazan na sliki 2.3.



Slika 2.3: Uporabniški vmesnik sfxr orodja.

Poglavje 3

Predprodukcija

Razvoj igre poteka po različnih metodologijah razvoja in vsebuje različne korake, odvisno od skupine razvijalcev ter njihovega načina dela. Kljub temu razvoj iger v industriji večinoma poteka po ustaljenih fazah razvoja. Vsaka faza se osredotoča na določene cilje, ki jih moramo izpolniti, preden nadaljujemo z naslednjo fazo. Takšen strukturiran pristop pripomore k obvladljivosti projekta.

Verjetno najpomembnejša in hkrati tudi najtežja faza razvoja igre je predprodukcija. Čeprav v tej fazi ne razvijamo igre neposredno (naše delo v tej fazi ne bo vidno v končnem produktu), je predprodukcija bistvenega pomena. Z razmišljanjem, odgovarjanjem na ključna vprašanja ter testiranjem določimo osnovno idejo igre, ki jo postopoma nadgrajujemo. Po končani predprodukciji imamo v dokumentaciji definirane zahteve za fazo produkcije, načrt, kako jih uresničiti ter prototip, s katerim dokažemo, da je naša ideja izvedljiva in privlačna.

Razvijalci večjih iger tekom predprodukcije napišejo vsaj tri ključne dokumente: dokument z idejo igre (angl. High concept document), dokument s konceptom igre (angl. Concept document) in tako imenovani dokument z načrtom igre (angl. Game Design Document - GDD). Vsak dokument zahteva odgovore na določena ključna vprašanja in opiše igro na postopoma

bolj podroben način. Uporabljajo se predvsem za poročanje zahtev različnim članom razvijalske ekipe. Poleg tega služijo kot predloga založnikom, ki se morajo odločiti, ali bodo idejo finančno podprli. Treba je pripomniti, da se lahko vsebina in naslov dokumentov rahlo razlikujeta pri različnih podjetjih. Tako lahko v različnih literaturah zasledimo, da dokument ideje obsega do 4 strani, medtem ko je drugje ista vsebina ločena na dva dokumenta. Kljub temu smisel in vsebina dokumentov ostajata ista.

Pri razvoju igre mi ne bo treba prepričati založnika ali poročati zahtev drugim razvijalcem. Vseeno pa sem v fazi predprodukcije sledil ključnim točkam teh treh dokumentov, saj sem tako lažje razvil svojo začetno idejo. Tako bom na konkretnem primeru predstavil, kako poteka proces oblikovanja igre, ki je eden od glavnih vidikov razvoja igre.

3.1 Dokument z idejo igre

Dokument ideje na kratko opiše bistvo igre. Namen dokumenta je, pritegniti pozornost in zanimanje za idejo, ki bi lahko postala dobra igra. Ker imamo na začetku veliko idej, je dokument kratek - ponavadi dva stavka, kjer odgovorimo na ključne točke dokumenta:

1. Cilj igre.
2. Žanr igre.
3. Glavne zanimivosti, ki bodo pritegnile igralca.

Dokument ideje si lahko predstavljamo kot nekaj, kar bi napisali za oglas igre. Dokument ideje moje igre, bi lahko bil:

„2D Pixel art igra, kjer se kot škrat boriš proti pošasti iz globin. Bodi pazlijv, da v zameno za moč ne žrtvuješ nečesa bolj pomembnega ...“

V tem kratkem opisu je lahko razbrati, da gre za žanr akcijske 2D igre z vizualnim slogom Pixel arta. Cilj igre se je v vlogi škrata boriti proti sovražnikom. Na koncu je tudi namig na glavno zanimivost igre - sistem odločitev in posledic.

3.2 Dokument s konceptom igre

Ko določena ideja pritegne zanimanje, je potrebno napisati dokument koncepta. Namen tega dokumenta je, podrobneje predstaviti igro in razviti osnovno idejo iz prejšnjega dokumenta. Potrebno je opisati tudi, komu je igra namenjena in zakaj bi bila igra konkurenčna ostalim, podobnim igram na trgu. Ponavadi se dokument koncepta preda založniku (ki mu je že všeč ideja igre), ki se nato odloči, ali bo podprl razvoj igre. Uporaben je tudi za predstavitev igre ostalim članom razvojne ekipe, saj se včasih razvoj začne pred odobritvijo založnika. Dokument je dolg okoli 10 strani in vsebuje sledeče točke.

3.2.1 Cilji

Med razvojem igre ne smemo preveč zaiti od vizije, ki jo želimo uresničti. Sprememba fokusa igre lahko podaljša čas razvoja in uniči načrt, ki smo ga naredili v predprodukciji. Na koncu takšna igra postane nezanimiva zmes različnih igralnih mehanik, ki niso povezane v celoto. Zato je pomembno, da se v fazi predprodukcije določijo cilji igre - kakšen ton bo imela igra in kakšne občutke ima namen vzbuditi v igralcu. To ne pomeni, da med razvojem ni dovoljeno spremeniti nekaterih oblikovnih odločitev, če se izkažejo za neuspešne. Seznam ciljev nam pomaga, da kljub spremembam ohranimo prvotno vizijo in občutek igre.

Cilji moje igre so:

1. Vzdušje temačne pravljice. Bolj podobno Grimmovim in slovanskim pravljicam kot Disneyevim. Postopno odkrivanje informacij o svetu, ki se razlikuje od klasične fantazije v stilu Tolkiena. Svet je preprostejši, vendar bolj divji in barbarski.

2. Nenavadno okolje in izgled sveta. Kontrast živih barv s popolnoma črnim ozadjem. Barvit, retro vizualni stil, ki je kvalitetnejši kot večina konzolnih iger devedesetih let, po katerih se zgleduje.

3. Arkadno nasilje brez krvi, vendar z zanimivimi animacijami, ki dajo igralcu občutek zadovoljstva med bojevanjem. Ne sme biti realistično ali depresivno.

4. Napeti trenutki bojevanja v kontrastu z mirnim oddihom po težkem izzivu.

5. Občutek moči in napredovanja, ki sproži željo po nadaljnem igranju in radovednost po odkrivanju nadgradenj.

6. Preprosta, kratka igra.

3.2.2 Motivacija igralca

V tej točki je potrebno opisati, kaj bo motiviralo igralca, da ne bo izgubil zanimanja med igranjem. Kakšne lastnosti in mehanike ima igra, da v igralcu vzbudi željo po igranju naše igre?

Različni ljudje igrajo igre zaradi različnih razlogov. Z napredovanjem tehnologije in popularnostjo so igre postale kompleksnejše in privlačijo igralce, ki jih zanimajo različni vidiki igranja. Raziskovalci so s preučevanjem odziva igralcev med igranjem odkrili ponavljajoče se vzorce čustev, obnašanja in zadovoljstva. Na osnovi teh spoznanj so bile narejene tipologije, s katerimi klasificiramo različne igralce.

Tipologije slonijo na psiholoških modelih, kot je Mayers-Briggs Type Inditicator (MBTI) in uporabljajo nova odkritja iz nevrobioloških raziskav. Ena izmed novejših tipologij je BrainHex model [23], ki definira 7 različnih tipov igralcev:

1. Iskalec (angl. Seeker) je tip igralca, ki ga motivira raziskovanje sveta igre. Procesiranje zaznavnih informacij in uporaba dela možganov, namenjenega asociacijam ter spominu (hipokampus), sprošča endomorfine. Ti nato sprožijo center za užitek, ki vzbudi zadovoljstvo v igralcu.

2. Preživeli (angl. Survivor) je igralec, ki uživa v občutku straha. Strah in napete situacije povzročijo sproščanje adrenalina, ki okrepi učinke

dopamina - hormona, ki se sprosti, ko se napeta situacija konča in povzroči občutek zadovoljstva ter oddiha ob končani nevarnosti.

3. Drznež (angl. Daredevil) je podoben tipu preživeli. Razlikuje se v količini strahu, ki ga igralec občuti - namesto v strahu in terorju, igralec uživa v napetih situacijah ter nadzorovanih tveganjih.

4. Organizator (angl. Mastermind) rad rešuje uganke in probleme. Ob tem občuti zadovoljstvo zaradi bližine centra za odločanje in centra za užitek.

5. Zavojevalec (angl. Conqueror) v igri išče zahtevne izzive. Soočanje z izzivi sprosti adrenalin in noradrenalin v sklopu instinktivnega odziva borba-beg. Po uspešnem končanju izziva možgani občutijo zadovoljstvo.

6. Družabnež (angl. Socializer) je tip igralca, ki išče pozitivno interakcijo z drugimi igralci. Med interakcijo se v centru za socializacijo, sproži nevrotransmiter, povezan z zaupanjem. Če oseba ustvari družabno vez z drugim igralcem, ob tem čuti zadoščenje ali jezo, če je njegovo zaupanje izdano.

7. Dosežkar (angl. Achiever) občuti zadovoljstvo ob dosegu cilja v igri. Ni pomembno, ali je cilj zahteven ali lahek, vsakič, ko je določen cilj dosežen, se sprosti hormon dopamin. To lahko vodi v obsesivno iskanje in končanje vseh nalog in ciljev, ki jih ima igra na voljo.

Če smo seznanjeni s tipologijo igralcev, lahko svojo igro lažje oblikujemo, saj vemo, katere lastnosti igre motivirajo določen tip igralcev.

Moja igra bo imela tri glavne lastnosti, ki bodo motivirale igralca na različne načine. S svojo atmosfero bo igra omogočala igralcem, da se vživijo in raziskujejo ta svet. To bi motiviralo predvsem igralce tipa iskalec.

Igralce bo motiviral bojevalni sistem. Igra bo postopoma zviševala zahtevnost, kar bo motiviralo igralce, da premagajo nove izzive in se domislijo

novih strategij, ki jim bodo pri tem pomagale. Te motivacije ustrezajo predvsem igralcem tipa zavojevalec in organizator, v nekaterih situacijah tudi drznežu.

Igra bo po uspešnem izzivu igralcu za nagrado darovala različne nadgradnje. To bo gnalo igralčevo željo po napredovanju in sprožilo občutek zadovoljstva, ko bo z novo pridobljenimi močmi z lahkoto premagal prejšnje sovražnike. To ugaja tipu dosežkar in okrepi motivacije tipov iskalec (odkritje nove informacije o fantazijskem svetu), zavojevalec (občutek večje moči) in organizator (nova orodja za premagovanje problemov).

3.2.3 Ciljna skupina

Danes igre niso več omejene na mladostnike in otroke, ampak so postale razširjen način zabave za ljudi različnih demografskih skupin. Kar 71% prebivalcev ZDA, starih od 6 do 49 let, redno igra igre. Leta 2016 je bila v ZDA povprečna starost igralcev 35 let. Igre niso omejene glede na spol, 41% vseh igralcev je žensk. Z leti bodo te številke še narasle, saj igralci odraščajo, tehnologija in posledično industrija iger pa napredujeta [24].

Vendar vsi igralci ne igrajo istih iger. Ena od preprostejših ločitev igralcev je na tako imenovane občasne (casual) in resne (hardcore) igralce. Posledično so različne tudi igre, ki ciljajo na različno skupino. Vendar je pri oblikovanju iger potrebno upoštevati več dejavnikov, kot jih ponuja ta preprosta delitev.

Eden od demografskih dejavnikov je spol. Kljub izjemam, večina žensk igra preproste mobilne igre, s katerimi se lahko, v prostem času, zamotijo za nekaj minut. Prav tako so pri ženskah priljubljene konzole, kot je Nintendo Wii, ki z globinsko kamero omogočajo igranje različnih fizičnih aktivnosti, na primer tenis ali jogo.

Še bolj kot spol na izbiro iger vpliva starost igralca. Tako so igre, namenjene majhnim otrokom, preprostejše in morajo ustrezati določenim omejitvam. Poleg tega je možno posplošiti lastnosti in vrednote različnih generacij. Igralcem, ki so odraščali v času interneta, je tako pomebno, da imajo igre spletno povezavo, kjer lahko delijo svojo izkušnjo z vrstniki. Starejše igralce,

ki so odraščali v devetdesetih letih, privlači retro izgled igre, ki jih nostalgično spominja na otroštvo. Še starejšim igralcem verjetno nista tako pomembna spletna povezava in izgled igre, ampak miselni izziv, ki ga predstavlja igra.

Poleg demografskih dejavnikov so pomembni tudi psihografski - kakšno osebnost in motivacije ima igralec (opisano v poglavju 3.2.2 Motivacija igralca) ter geografski dejavniki. Geografska lokacija igralcev vpliva na lastnosti trga v določeni regiji. Kitajski trg iger je osredotočen na brezplačne MMO (massively multiplayer online games) spletne igre. Igralci v teh igrah običajno ponavljajo iste naloge, kar bi se igralcem evropskega trga zdelo dolgočasno. V Južni Koreji prevladujejo spletne igre na osebnih računalnikih, medtem ko je na japonskem največji del trga iger namenjen konzolam. Regija vpliva tudi na lokalizacijo iger. Da lahko igro prodajamo na različnih trgih, jo je potrebno prevesti, včasih pa tudi spremeniti vsebino. V Nemčiji je, na primer, prepovedana prodaja zelo nasilnih iger ali iger, ki vsebujejo nacistične simbole.

Jasno je, da posamezna igra ne bo ustrezala zahtevam in željam vseh igralcev. Zato je potrebno, na podlagi opisanih dejavnikov, določiti ciljno skupino igralcev, ki jim bo igra namenjena.

Ciljna skupina igre, ki sem jo razvil v diplomski nalogi, so igralci stari od 20 do 35 let, ki so v mladosti igrali klasične 2D Super Nintendo igre, kot so Castlevania ali Ninja Gaiden. Sedaj, ko so odrasli, želijo igrati igro podobnega izgleda z novimi, zanimivimi igralnimi mehanikami. Igra bo v angleškem jeziku, imela bo elemente klasične, zahodne fantazijske zgodbe, zato bo namenjena predvsem zahodnemu trgu (Evropa, Severna Amerika).

3.2.4 Žanr

Izbira žanra za igro je pomemben del oblikovanja. Žanri v igrah so definirani glede na način igranja in ne po zgodbi ter vzdušju, kot pri filmih ali knjigah. Žanr vpliva na ciljno računalniško okolje, za katero razvijamo igro, na ciljno skupino, ki jo bo igrala in celo na prodajo igre. Nekateri žanri so v določenem času bolj popularni kot drugi. Tako so v času pisanja diplome popularne

preživetvene igre, medtem ko so bile konec devetdesetih na vrhuncu strateške igre. Založniki so skeptični podpreti igro v žanru, ki se v kratkem ni izkazal za zelo priljubljenega, kar se odraža v nasičenosti podobnih iger na trgu.

Vendar žanr ne predstavlja omejitev, ampak osnovno ogrodje in priročno orodje, s katerim lahko opišemo stil igranja naše igre. Nenazadnje je vsak žanr posledica originalne igre, ki je postala tako uspešna, da so njene igralne mehanike postale ustaljen koncept pri igranju in oblikovanju iger. Primer je igra Doom, ki je izšla leta 1993 in bila ena prvih prvoosebnih iger, kjer ima igralec možnost streljanja. Podobne igre, ki so posnemale Doom, so bile znane kot „Doom-clones“, dokler se ni uveljavilo ime za nov žanr - prvoosebna strelska igra (angl. first person shooter - FPS).

Glavni žanri iger so:

1. Akcijske igre. Različne arkadne, strelske, ploščadne, dirkaške in pretepaške igre. Akcijske igre so napete, ponavadi realnočasovne igre, kjer se igralec zanaša na svoje reflekse za doseg zmage.

2. Pustolovske igre so bile prve računalniške igre. Igralec potezno sprejema odločitve. Opišemo jih lahko kot interaktivne zgodbe.

3. Igranje vlog. Igralec prevzame vlogo osebe v navideznem svetu. Glavne lastnosti tega žanra so napredovanje igralca, bolj zapletena in daljša zgodba ter izrazito vzdušje v igri.

4. Miselne igre, na primer Tetris, so namenjene reševanju abstraktnih miselnih izzivov.

5. Strateške igre se delijo na potezne in realnočasovne. Ponavadi igralec upravlja z vojsko in preprosto ekonomijo določene države v igri.

6. Simulacije poskušajo poustvariti izkušnjo resničnega sveta, na primer upravljanje letala ali nadzorovanje železniške infrastrukture.

7. Kakršnakoli kombinacija dveh ali več žanrov. Na primer akcijsko-pustolovske igre, ki združujejo hitro akcijsko bojevanje z reševanjem izzivov in napredovanjem zgodbe.

Igra iz diplomske naloge spada v akcijski žanr. Bolj natančno podžanr ploščadnih iger, kjer se igralec premika v 2D svetu, sestavljenem iz ploščadi. Igra vsebuje še okrnjene elemente žanra igranja vlog - preprosto napredovanje igralca ter zgodbo.

3.2.5 Starostna ocena

Igra pridobi starostno oceno, ko jo razvijalci prijavijo na ocenitev pri eni od ocenitvenih agencij. Te agencije, podobno kot za filme, ocenijo igro kot primerno za določeno starostno skupino na podlagi različnih kriterijev. Dva najbolj razširjena sistema ocenitve sta ESRB [25] in PEGI [26]. ESRB se uporablja predvsem v ZDA, medtem ko je PEGI razširjen v Evropi. Starostna ocena v veliki večini držav ni določena z zakonom. Kljub temu igra potrebuje oceno, če jo želimo prodajati na večjih računalniških okoljih kot Playstation ali Xbox. Prav tako večina fizičnih trgovin igre ne bo prodajala, če ni ocenjena z določenim sistemom.

Manjše igre neodvisnih proizvajalcev ponavadi niso starostno ocenjene. Eden od razlogov za to je cena ocenitve, ki znaša od 300 do 2500 evrov [27], odvisno od velikosti igre. Spletne trgovine, kjer objavijo igre, nimajo tako strogih pogojev prodaje in ne zahtevajo ocene za vsako igro. Zaradi istih razlogov ne bom zaprosil za starostno oceno moje igre. Če igro ocenim sam, po kriterijih sistema PEGI, bi dobil oznako PEGI 7 - igra vsebuje prizore, ki bi lahko prestrašili mlajše otroke, vendar nasilje ni prikazano realistično, osebe v igri niso podrobno prikazane.

3.2.6 Analiza konkurence

Pri oblikovanju igre je potrebno pomisliti na sorodne produkte in ugotoviti, kako bomo tekmovali z njimi. Tudi če igre ne bomo prodajali, lahko z analizo konkurence izboljšamo vidike naše igre.

Shovel Knight

Verjetno najbolj uspešna 2D ploščadna igra v zadnjih letih. Izšla je leta 2014 in v enem letu prodala miljon kopij [28]. Je obsežna akcijsko-pustolovska igra, z retro izgledom in igralnostjo v stilu klasičnih, konzolnih iger devedesetih let. Moji igri je podobna predvsem zaradi oblikovalnega cilja - narediti igro v retro stilu z modernimi izboljšavami.

Kot neodvisni razvijalec brez proračuna težko konkuriram takšni večji igri. Zato sem igro oblikoval osredotočeno samo na nekaj vidikov, ki sem jih naredil kvalitetno in zanimivo (bojevanje). Omejitev vsebine in kvalitetna izvedba tiste vsebine, ki jo nameravamo implementirati, je dober način konkuriranja večjim produkcijam.

Risk of Rain

Risk of Rain je primer konkurenčne, 2D akcijske ploščadne igre. Podobno kot moja igra se osredotoča na bojevanje pošasti v areni, z različnimi nadgradnjami igralca in naraščujočo težavnostjo. Moja igra je lahko konkurenčna tej večji produkciji, če se od nje razlikuje v izvedbi: večji poudarek da zgodbi, uporablja drugačen vizualni slog in druge mehanike bojevanja s sovražniki.

Množica iger

Zaradi dostopnosti tehnologije in lahkote izdaje iger preko spleta (brez preverjanja kvalitete), je trg iger nasičen. Samo na spletni trgovini Steam [29], izide povprečno 21 iger dnevno. V celotnem letu 2017 je bilo izdanih kar 7672 iger [30, 31]. To so v večini majhne neodvisne igre, ki so si zelo podobne in niso kvalitetne. Da je naša igra uspešna, mora konkurirati tako večjim uspešnim igram kot tudi tej množici iger, ki jo lahko zasenčijo, ne s kvaliteto, ampak s samim številom. Sama kvaliteta naše igre za to ne zadošča - potrebno je oglaševanje glavnih zanimivosti igre, ki bodo pritegnile pozornost.

Sam igre ne bom oglaševal, ker služi samo kot pilotna študija. Igro

lahko oglašujemo brezplačno prek družabnih omrežij, kontaktiranja novinarjev, spletnih strani in youtuberjev, ki se ukvarjajo z igrami.

3.2.7 Ostale točke

V dokumentu koncepta ponavadi obstajajo še druge točke, ki jih nisem opisal. Razlogi za to so, da jih za razvoj igre nisem potreboval (spletne značilnosti, proračun, produkcija, licenca) ali pa sem jih opisal v naslednjem dokumentu (igralnost, zgodba, izgled in konceptne slike).

3.3 Dokument z načrtom igre

Dokumnet načrta igre (GDD) je daljši kot ostala dokumentacija, ponavadi obsega od 50 do 200 strani. Namen dokumenta je, podrobno opisati igralne mehanike, zgodbo, izgled in ostale točke, ki so potrebne v fazi produkcije. GDD mora opisati igro tako podrobno, da dobimo predstavo o izgledu in delovanju končne igre. S tem dokument služi kot referenca vsem članom razvojne ekipe. Tekom produkcije se pogosto dopolnjuje in spreminja, nova verzija pa je vedno dostopna vsem članom.

3.3.1 Igralnost

V splošnem je igralnost pojem, s katerim označujemo, kaj igralec dela v igri in na kakšen način to počne. Je ločen vidik igre, poleg izgleda in zvoka. Kljub temu dopolnjuje ostale vidike in je z njimi povezan v zaključeno celoto.

Za oblikovanje igralnosti je potrebno dobro razumevanje iger, ki nam omogoča prepoznati posamezne komponente in njihov vpliv na igro ter igralca. Za ta namen je na voljo različna literatura, ki opisuje teorijo iger (kakšni izzivi obstajajo v igri in kakšni so lahko odzivi igralca) [11] ali pa vsebuje vrsto vprašanj, katera nas opomnijo na pomembne mehanike (pogoji zmage in poraza, uravnoteženost težavnosti) [7]. Za oblikovanje iger kljub temu ne obstaja en pravi ali ustaljeni postopek. S poznavanjem različnih igral-

nih mehanik, izkušnjami igranja in oblikovanja iger je potrebno združiti ter prilagoditi izbrane mehanike v nov, željen način igranja.

Temeljne igralne mehanike

V dosedanji dokumentaciji smo opisali osnovno idejo naše igre in njene cilje. Okoli teh ciljev je potrebno oblikovati temeljne igralne mehanike. To so pravila, akcije in igralni elementi, ki sestavljajo ogrodje igre in omogočijo željeno igralnost.

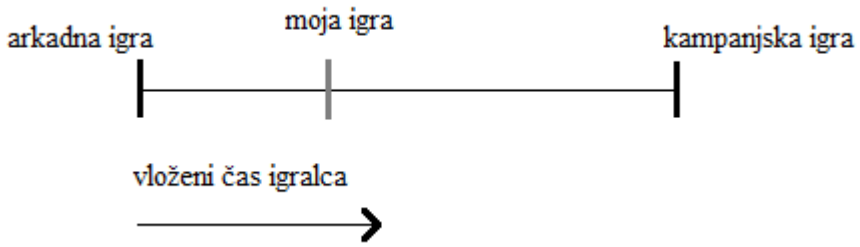
Cilji naše gre (preprosta arkadna igra z zanimivo zgodbo, vzdušjem in nadgradnjami) si deloma nasprotujejo, kar predstavlja izziv za oblikovanje igralnih mehanik.

Pri preprostih, arkadnih igrah velja, da jih igramo nekaj minut ter nato umremo ali odnehamo in pri naslednjem igranju začnemo znova od začetka. Zaradi takšnega načina igranja v igri ne moremo imeti kompleksnega sistema napredovanja - igralec bi izgubil motivacijo, če bi moral vsakič znova pridobiti isto opremo in igrati isto zaporedje nivojev. Njegov vloženi čas bi se zdel izgubljen.

Nasprotno obstajajo kampanjske igre, ki so oblikovane tako, da igralec napreduje skozi zaporedne linearne nivoje. Ko igralec umre ali prekine z igranjem, lahko naslednjič nadaljuje od tam, kjer je končal. To omogoča implementacijo kompleksnejših sistemov, na primer pridobivanje denarja, s katerim kupimo opremo. Igralcu lahko lažje podamo kompleksnejšo zgodbo, na podoben način kot v filmu ali knjigi. Z ekspozicijo mu povemo ali prikažemo ključne dogodke. Lastnost takšne igre je, da je zanimivo le prvo igranje. Ob naslednjih igranjih poznamo potek zgodbe in nivojev. Poleg tega je razvoj takšne igre daljši, saj se ne osredotoča na ponavljajoča, kratka igranja.

Pri oblikovanju igre sem torej moral narediti kompromis (na sliki 3.1 je prikazana uvrstitev moje igre). Oklestiti sem moral različne lastnosti obeh tipov iger in jih združiti. Igra je sestavljena iz dveh faz: najprej se igralec bori proti navalu pošasti, kar predstavlja glavni del igre. Bojevanje je preprosto ter ostaja dokaj nespremenjeno čez celotno igro. Po končani bitki nastopi

faza miru. Igralec si lahko vzame čas in si v miru ogleda videz okolja v nivoju in se odloči, kako bo nadgradil svoj lik. V tej fazi podamo igralcu zgodbo in vzdušje igre. Vendar igralec ne bo prisiljen v odkrivanje zgodbe in se bo lahko temu v celoti izognil ter se osredotočil samo na izziv bojevanja.



Slika 3.1: Shema vloženega časa, ki ga igralec preživi v neprekinjenem igranju

Mehanike igralca

Nadgradnje morajo dopolnjevati temeljne igralne mehanike in narediti igranje bolj zanimivo. Igralec bo pridobival nadgradnje s trupli padlih pošasti. Imel bo izbiro preučiti ali pojesti truplo. Odvisno od odločitve bo pridobil „pojej“ ali „razišči“ točke. Ko bo dosegel določeno število točk, bo pridobil nadgradnjo pripadajoče stopnje, opisane v tabeli 3.1.

Nadgradnje morajo povečati moč igralca, brez dodajanja kompleksnosti bojevalnemu sistemu. Namesto da igralec pritisne pravo zaporedje gumbov, da izvede močnejši napad, lahko z njim preprosto nadomestimo njegov prejšnji napad. Občutek napredka in moči, ki ga igralec dobi ob nadgradnjah dosežemo tako, da z njimi opazno spremenimo dosedanji način igranja. Novo pridobljeni napad ima morda mnogo daljši doseg, kar omogoči nove pristope pri bojevanju. Toda če je nadgradnja premajhna in nezanimiva (na primer 5% večja odpornost igralca), ga ne bo motivirala in popestrila igranja.

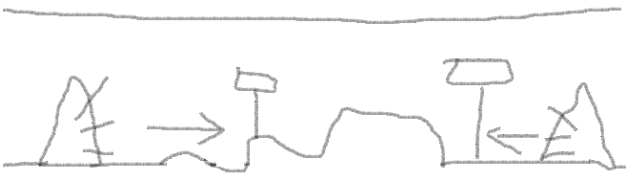
Poleg povačanja moči bodo nadgradnje služile kot orodje, s katerim bomo podali igralcu delčke zgodbe in s tem poglobili vzdušje igre. Ob pridobitvi nove veščine se bo prikazal kratek opis občutja, ki preveva njegov lik.

Tip nadgradnje	Pojej	Razišči
Napad z bližine	Igralcu zraste dodatna roka, ki omogoča hitrejše napade.	Igralec zažge svoje orožje, kar ojača napad.
Napad z daljave (porablja omejeno zalogo energije)	Igralec se spremeni v neranljivo demonsko obliko ter se požene čez zaslon. Rani vse pošasti, ki se ga dotaknejo.	Igralec naredi lok, s katerim lahko močno rani enega sovražnika.
Izboljšava lika	Igralec ima 50% možnosti, da ostane nepoškodovan, ko ga udari sovražnik.	Igralcu se podvoji količina zdravja.

Tabela 3.1: Možne nadgradnje igralca

Oblikovanje nivoja

Oblikovanje nivoja močno vpliva na igro. Z različnimi elementi igre, ki jih imamo na razpolago, je potrebno sestaviti nivo in s tem celotno igro. Z nivojem oblikovalec usmerja potek igre, kreira scenarije in prilagaja težavnost. Dobro oblikovan nivo naredi igranje tekoče in zanimivo, medtem ko lahko slab nivo popolnoma uniči predstavo igre, tudi če so ostale mehanike popolne. Zato imajo večje produkcije zaposlene profesionalne oblikovalce nivojev, ki so vešči različnih disciplin, da kreirajo tehnično funkcionalen in vizualno privlačen nivo.



Slika 3.2: Začetna skica glavnega nivoja igre.

Večina moje igre se bo dogajala na enem nivoju. Na sliki 3.2 je prika-

zana skica nivoja. Ta bo deloval kot arena, kjer se bo igralec boril proti nasprotnikom, ki se bodo premikali proti njemu. Nasprotniki bodo prihajali iz obeh strani zaslona. S tem prisilimo igralca, da se premika po celotnem nivoju in preprečimo, da stoji na mestu ter čaka, da nasprotniki pridejo v njegov doseg napada. Nivo ima razgibano površino z občasnimi ploščadmi. To omogoča vertikalno premikanje, ki popestri igro. S skokom na ploščad se bo lahko igralec izognil tudi nekaterim pošastim, kar bo nujno, ko bo obkoljen. Hkrati je nivo dovolj preprost, da bo umetna inteligenca nasprotnikov brez problemov sledila igralcu. Nivo bo obsegal približno 2 zaslona, dovolj za manevriranje igralca in ne preveč, da bi se nasprotniki razpršili ter s tem uničili občutek bitke. Poleg tega mora biti nivo smiseln v kontekstu zgodbe in se ujemati z zmožnostmi nasprotnikov in igralca.

Mehanike sovražnikov

V igri bo več tipov sovražnikov:

1. Pešak - običajen vojak, ki z dotikom rani igralca. Nima nobenih posebnih napadov in je povprečno trpežen.

2. Strelec - periodično vrže izstrelek v igralca, če je ta v dosegu. Je zelo ranljiv.

3. Velikan - ogromen vojak, ki lahko porine igralca s ploščadi. Je zelo trpežen in ima napad, ki močno odrine igralca.

4. Letalec - leteči vojak. S svojo potjo letenja ovira igralca pri skakanju. Je zelo ranljiv.

5. Šef - končni sovražnik. Je kopija igralca z vsemi njegovimi sposobnostmi. Je trpežen in nevaren, predstavlja najtežji izziv v igri.

Ti bodo predstavljali izziv, ki ga bo moral rešiti igralec. Vsak tip sovražnika bo imel določene lastnosti, ki bodo prisilile igralca, da spremeni dosedanjo taktiko bojevanja. Poleg tega se bodo sovražniki dopolnjevali med

seboj. Tako bo običajen vojak deloval kot blokada in ščitil ranljivega strelca pred napadom igralca.

Igralec se bo srečal z različnimi tipi postopoma. Nov tip sovražnika se bo pojavil v igri, ko bo igralec navajen bojevanja z dosedanjimi sovražniki. Novi sovražnik ga bo presenetil in mu predstavil težek izziv, saj ne bo imel znanja ter nadgradnje, ki bi mu lahko olajšala bojevanje z njim. Z zmago nad sovražnikom se bo spoznal z njegovimi lastnosti in pridobil točke iz njegovega trupla, s katerimi bo dobil novo nadgradnjo. To mu bo omogočalo, da bo naslednjič lahko premagal ta tip sovražnika lažje in ob tem izkusil občutek zadovoljstva in napredovanja.

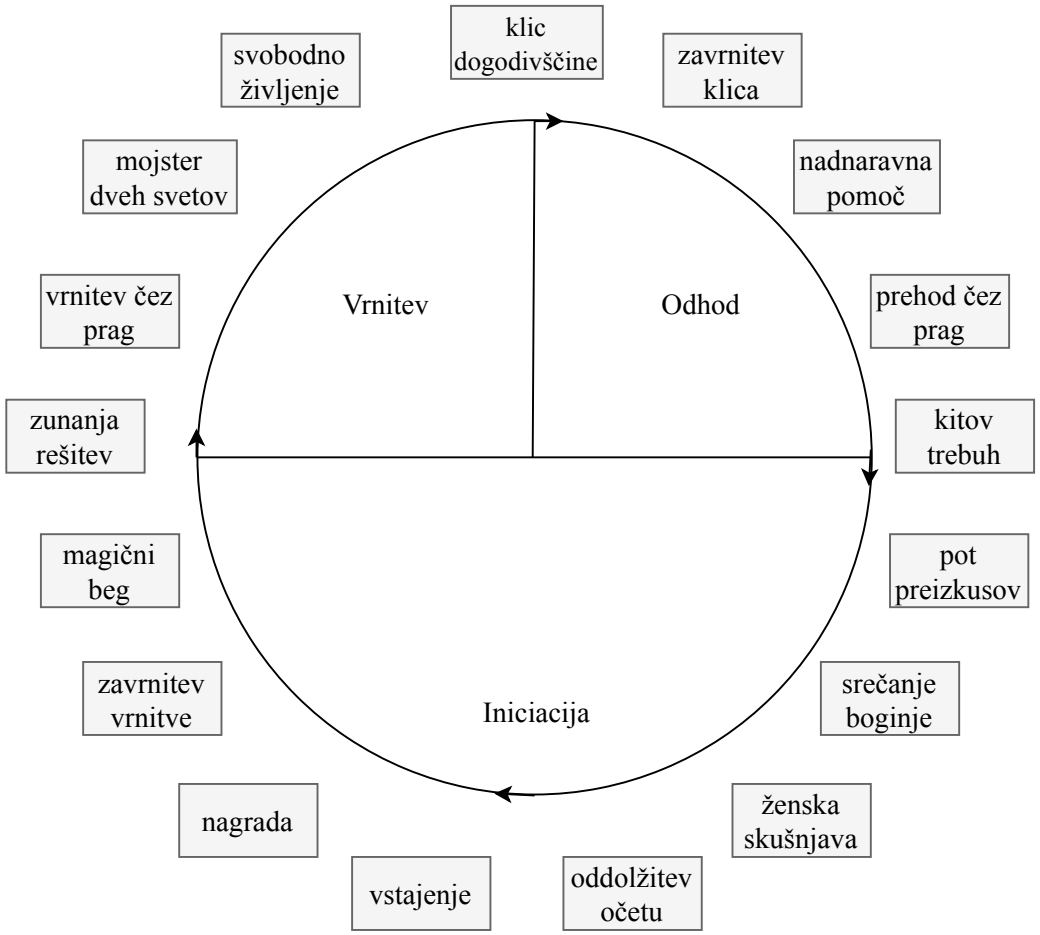
3.3.2 Zgodba

Struktura zgodbe

Skozi igro lahko igralcu podamo zgodbo. Dobra zgodba obogati igralnost, vzpostavi ton in vzdušje ter doda smisel ostalim vidikom. Igra ima lastnosti, s katerimi se razlikuje od ostalih medijev. V primerjavi s filmom ni časovno omejena, kar nam omogoča, da ustvarimo obsežnejše in podrobnejše ozadje zgodbe. Pomembna lastnost igre je, da lahko igralec vpliva na potek zgodbe in s svojimi odločitvami spreminja konec.

Kljub temu zgodba večinoma sledi ustaljenim strukturam, ki se niso spremenile od antičnih časov. Joseph Campbell je v knjigi The Hero with a Thousand Faces [32] opisal osnovno pripovedno strukturo, ki jo je poimenoval monomit (prikazan na diagramu 3.3). Ta opisuje pot heroja skozi različne faze: od začetka v normalnem okolju, do zmage nad preprekami v nadnaravnem okolju in vrnitve spremenjenega heroja v začetno okolje.

Faze monomita lahko organiziramo tudi v klasično strukturo dramskega trikotnika. Ta vsebuje zasnovo, zaplet, vrh, razplet in razsnovo. Takšne strukture je mogoče zaslediti v različnih filmih (tipičen primer monomita in potovanja heroja je Luke Skywalker v filmu Star Wars), knjigah (še posebno v antičnih dramah) in tudi v igrah.



Slika 3.3: Diagram strukture Campbellovega monomita.

Zgodba v moji igri

Zgodba se lahko razlikuje od klasične strukture in s tem določi ton pripovedi. V moji igri se bo začela pred (ponavadi) prvo fazo - predstavitvijo heroja in problema. Zgodba se začne z igranjem škratovskega rudarja, ki pomotoma prebije zid in odkrije nenavadno jamo. V jami ga napade pošast. Ko jo premaga, se ga polasti nenadzorovana lakota, ki ga prisili, da poje truplo. To sproži večji naval sovražnikov. Igralec se bo poskušal boriti proti njim, vendar na koncu ne bo uspešen. Ko rudarjevo truplo pade na tla, se konča predhodna faza. S tem smo presenetili igralca, saj smo ga takoj na začetku postavili v

nevarno, napeto situacijo. Kljub igralčevemu uspehu v tej sceni, njegov lik izgubi. Podobno kot v grozljivkah prikažejo prvi umor pred začetkom zgodbe, ta scena vzpostavi večjo napetost. Poleg tega scena služi kot uvod v igralne mehanike, ki igralca nauči premikanja in bojevanja.

Sledi uvodna animacija, ki opiše ozadje zgodbe in glavni problem. V animaciji igralec izve, da je minilo nekaj mesecev od smrti rudarja. Podzemno civilizacijo škratov so napadle neznane pošasti, ki so uničile vsak odpor. Zadnji branik je njihovo glavno mesto, ki ga mora igralec kot poveljnik straže braniti s svojim življenjem. Opisana je tudi surova narava škratov, ki imajo sposobnost prevzeti lastnosti drugih vrst s tem, ko pojejo njihova trupla. Igra opozori igralca, da se to lahko izkaže kot nevarno, saj poleg prenosa moči, truplo pošasti tudi okuži škrata z zlom.

Igralec nato prične z glavnim delom igre, v katerem se bojuje proti navalom pošasti.

Ko uspe premagati vse navale, se sooči z glavnim sovražnikom. Izkaže se, da je to isti rudar, ki ga je igralec nadzoroval v uvodu. Postal je nasprotna, zlobna kopija heroja, ki ga skuša premamiti na svojo stran. Takšen lik je v pripovedništvu znan kot senca (na primer temačna osebnost heroja v romanu Dr. Jekyll & Mr. Hyde [33]). V zgodbah se pogosto pojavljajo različni arhetipi, tipični primeri osebnosti, ki služijo kot osnova za kreiranje zanimivih oseb in situacij.

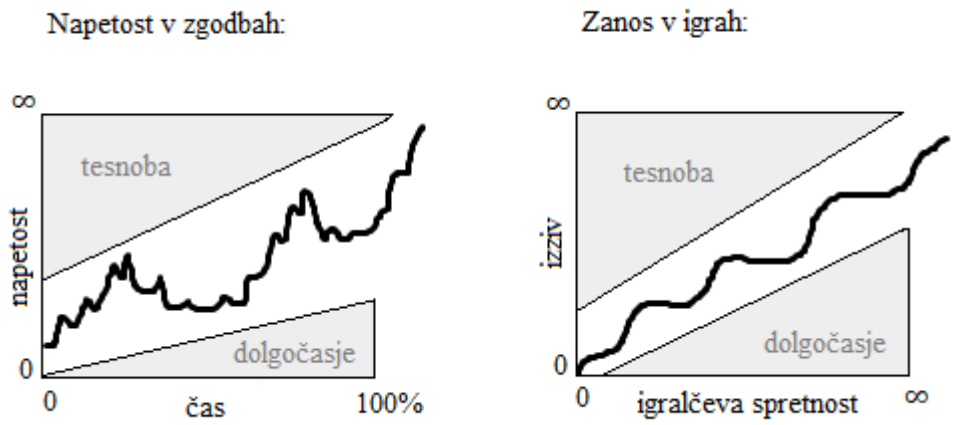
Po težavni bitki igralec premaga nasprotnika in zgodba se zaključi z enim od dveh koncev. Če je igralec nadgradnje pridobil s preučevanjem trupel pošasti, se upre skušnjavi in uspešno obrani škratje mesto pred pošastmi, vendar pri tem izgubi življenje. Če pa je v prejšnji fazi igralec jedel trupla pošasti, se njegov lik vda svojemu nagonu, poje truplo glavnega sovražnika in prevzame njegovo mesto kot vodja pošasti.

Krivulja zanosa

Zgodba je najprivlačnejša, ko sledi krivulji zanosa. Madžarski psiholog Mihaly Csikszentmihalyi je vpeljal pojem zanosa - stanja, ko smo osredotočeni

na neko aktivnost, ki nas popolnoma prevzame [34]. Da dosežemo takšno stanje, mora biti aktivnost v ravnovesju. To teorijo lahko prenesemo tudi na pripovedovanje zgodb. Napetost zgodbe mora postopoma naraščati, ne prehitro, da ne povzroči prevelikega stresa, in ne prepočasi, da ne postane dolgočasna.

Ponavadi ta krivulja ni gladka, ampak nazobčana. To predstavlja dvige in padce napetosti v posamezni sceni. Vsaka scena je lashko strukturirana v iste faze, kot jih ima celotna zgodba - torej je zaključena celota s konflikti in razpletom, ki nato vpliva na zgodbo v širšem smislu. Na koncu vsake scene se mora zgoditi sprememba. Podobno velja tudi za igralnost, kot lahko vidimo na sliki 3.4) [35].



Slika 3.4: Graf krivulje zanosa v zgodbah in v igrah.

Tako bom naredil tudi v moji igri. Kot sem opisal v igralnosti, sem oblikoval vsako srečanje z novim sovražnikom kot posamezno sceno. Ta služi kot uvod v nov problem, ki ga nato v napeti bitki igralec razreši in pridobi znanje in nadgradnjo. Ta predstavlja spremembo, ki vpliva na ostalo igralnost tako, da lažje premagamo iste nasprotnike. Težavnost in s tem napetost naraščata postopoma in se ujemata s krivuljo zanosa v pripovedništvu. S takšno strukturo bom dosegel zanimiv in uravnotežen potek igralnosti in zgodbe, ki se

prepletata v igri.

3.3.3 Vizualni slog

Vizualni slog je izgled igre z določenimi lastnostmi, ki si jih deli z ostalimi igrami, narejenimi v istem slogu. Vsako igro lahko uvrstimo v eno izmed širših katergorij, vendar ima lahko drugačen izgled v primerjavi s sorodnimi igrami - ima unikaten vizualni slog. Primer je igra Warcraft 3, ki tako kot igra Dungeon Siege spada v kategorijo 3D iger z „low-poly“ izgledom (modeli z majhnim številom podrobnosti). Na sliki 3.5 lahko vidimo, da Warcraft 3 izstopa s svojim barvitim, risankastim vizualnim slogom, čeprav sta obe igri izšli leta 2002. Tudi danes, ko je tehnologija močno napredovala, starejše igre z unikatnim slogom izgledajo dobro, medtem ko igre s takrat napredno grafiko in brez navdihnjenega sloga delujejo zastarelo.

To ponazori tudi razliko med vizualnim slogom in grafiko igre. Grafika z različnimi tehnikami izriše sliko igre na zaslonu. Te tehnike se razvijajo s tehnološkim napredkom strojne opreme, ki nam omogoča na primer bolj podroben izris večjega števila poligonov v sceni ali naprednejšo simulacijo senčenja in svetil. Vizualni slog ustvari izgled, ki igri doda željeni občutek in ton. Dober slog dopolnjuje ostale vidike igre in z izgledom posreduje informacije igralcu (igra Darkest Dungeon (prikazana na sliki 3.6) se s svojim temačnim, stripovskim slogom ujema z neizprosnimi igralnimi mehanikami ter zgodbo in vzbudi depresivno, napeto vzdušje).

Določanje sloga moje igre

Vizualni slog igre sem določil postopoma. Najprej sem naredil seznam ključnih besed, ki opisujejo vzdušje, ključne objekte ali lastnosti sveta, ki ga želim upodobiti. Nato sem poiskal pripadajoče slike, ki sem jih zbral v referenčno bazo slik. V večjih produkcijah izbrane slike pomagajo posredovati občutek in ton igre članom razvojne ekipe. V fazi produkcije referenčna baza vodi različne umetnike, ki kreirajo potrebne slike, animacije in modele, da ohranjajo konsistenten slog.

Slika 3.5: Primerjava izgleda igre Warcraft 3 (zgoraj) in Dungeon Siege (spodaj).

Kot sem opisal v prejšnjih poglavjih, želim, da je svet igre bolj podoben starim, temačnim pravljicam, kot običajnemu fantazijskem svetu. Zbrane slike prikazujejo staro slovansko arhitekturo, škrate, upodobljene v ruski in

Slika 3.6: Prizor iz igre Darkest Dungeon (Red Hook Studios, 2016)

germanski folklori. Ker se igra dogaja pod zemljo, sem pozornost namenil tudi slikam, ki prikazujejo stare rudnike, naravne jame in bioluminscenco v organizmih. Sovražnike v igri sem si zamislil kot sprevržene različice živali, ki jih je pokvarila zla energija, zato sem v bazo dodal fotografije polžev, netopirjev, žab in insektov.

Poleg slik, pridobljenih s spleta, sem narisal tudi nekaj preprostih skic, s katerimi sem lažje določil izgled sveta, kjer se bo dogajala igra. Ena izmed skic je prikazana na sliki 3.7.

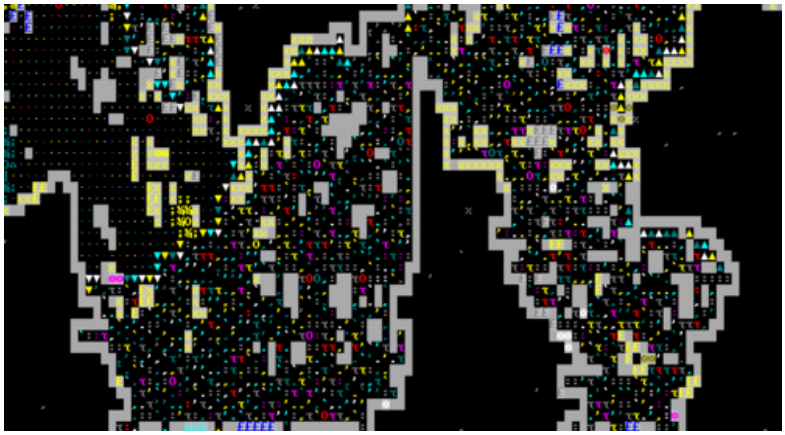
Ko sem določil, kaj želim prikazati v igri, sem se moral odločiti tudi, kako bom to prikazal - v kakšnem slogu. Glavni navdih je bil izgled igre Dwarf Fortress (prikazan na sliki 3.8). Z nasičenimi barvami, na popolnoma črnem ozadju in z majhno količino informacij, ki jih posredujejo simboli, igra vzbudi domišljijo. Podobno kot med branjem knjige, igralcu ni podana podrobna slika objekta, ampak samo namig, ki si ga nato igralec delno interpretira sam. Ker sem želel doseči podoben učinek, sem se odločil za uporabo Pixel art sloga.

Slika 3.7: Skica škratjega mesta, kjer se bo odvijala igra.

Pixel art slog

Pixel art je pojem, s katerim označujemo slog digitalne umetnosti, pri katerem manipuliramo s sliko na nivoju pikslov. Ponavadi so te slike nizke resolucije, da so lahko vidni posamezni piksli. Cilj sloga je, z majhnim številom podrobnosti posredovati čim več informacij in s tem vzbuditi več občutkov, kot če bi direktno, podrobno narisali objekt. Zato je pomembno, da umetnik natančno izbere barve in ročno postavi vsak piksel slike ter s tem kotrolira končen izgled slike. Razvil se je zaradi tehničnih omejitev prvih konzol in računalnikov, ki so poganjali grafične igre. Primer Pixel art sloga sta sliki 3.10 in 3.11, po slednji sem se zgledoval pri izdelavi vizualne podobe svoje igre.

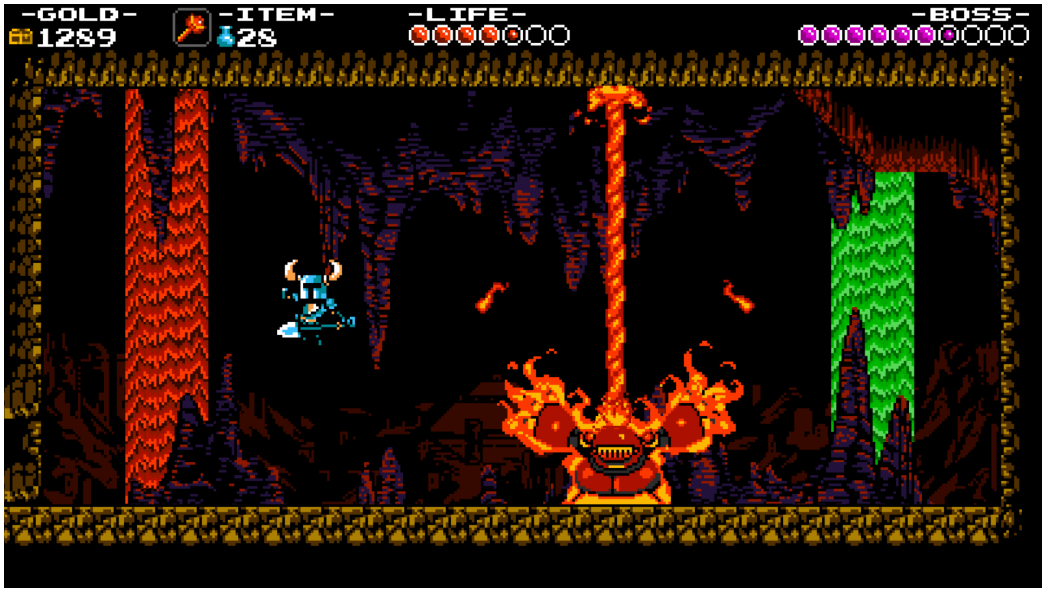
Kljub napredku tehnologije in razvoju 3D grafike je Pixel art še vedno popularen slog. Uporablja se na napravah z manjšo zmogljivostjo ali resolucijo, na primer na mobilnih telefonih in ikonah na namizju osebnih računalnikov. Pogost je v sodobnih neodvisnih igrah, saj ga odlikuje zanimiv retro videz, hkrati pa ne zahteva veliko časa, izkušenj, programske opreme in sredstev za



Slika 3.8: Prizor iz igre Dwarf Fortress (Bay 12 Games, 2006). Prikazan je tloris jame, kjer vsak simbol ponazarja določen objekt. Turkizna vejica tako pomeni, da se na tistem mestu v jami nahaja turkizno svetleča trava.

izdelavo. Kar pa ne pomeni, da je preprost - za obvladanje Pixel art sloga je potrebno dobro poznavanje teorije barv, veščin slikarstva, animiranja in principov računalniške grafike.

Slika 3.9: Prizor iz igre The Secret of Monkey Island (Lucasfilm Games, 1990).



Slika 3.10: Prizor iz igre Shovel Knight (Yacht Club Games, 2014).

Poglavje 4

Produkcija

Prudukcija je faza, v kateri celotna ekipa kreira vsebino igre. Programerji pišejo izvorno kodo in prototipirajo nove funkcionalnosti, ki so jih definirali oblikovalci iger. Umetniki ustvarjajo teksture in modele, glasbeniki ter skladatelji proizvajajo potrebne zvoke in skladbe. Tudi pisatelji pišejo dialoge za različne like v zgodbi.

Produkcija je najdaljša faza v razvoju igre, ponavadi traja okoli šest mesecev za manjše igre in več let pri večjih igrah (od dveh do štirih let).

Za mojo igro sem načrtoval, da bom fazo produkcije zaključil v enem mesecu. Izkazalo se je, da sem potreboval več kot 2 meseca. V zaključnih dneh sem igro razvijal večino dneva in bil primoran odstraniti nekatere načrtovane funcionalnosti. Ker sem igro razvijal prvič, nisem predvidel vseh funkcionalnosti, ki jih je bilo potrebno implementirati ali pa sem naletel na različne ovire, ki sem jih moral razrešiti (na primer pravilni prikaz pikslov opisan v nadaljevanju).

4.1 Tehnološki vidik

Igralni pogon, kot je Unity, omogoča uporabo že narejenih komponent, s katerimi lahko igralnim objektom dodajamo različne funkcionalnosti. Vendar so te v večini primerov osnovne, nezadostne za potrebe igre. Zato je potrebno

implementirati dodatne komponente, ki jih pogon ne vsebuje, ali pa dopolniti že vsebovane. To naredimo s pisanjem skript.

V Unityju je skripta datoteka, napisana v programskem jeziku C# ali JavaScript, ki se v urejevalniku, kot komponenta, doda določenemu igralnemu objektu. Nova skripta ima določeno strukturo:

1. Na začetku so definirani namenski prostori, ki jih bomo uporabljali v kodi.

2. Sledijo spremenljivke, ki so (če so javno definirane) vidne v uporabniškem vmesniku Unity za lažje spreminjanje vrednosti.

3. Nato je definiran razred, ki je izpeljan iz vgrajenega, osnovnega razreda MonoBehaviour.

Ko se med igranjem zgodi določen dogodek, Unity preko razreda MonoBehaviour požene funkcijo dogodka, ki je definirana v izpeljanem razredu naše skripte. Takšni funkciji sta na primer Update, ki se bo izvedla za vsako sliko (angl. frame) igre, in Start, ki se izvede pred začetkom igre in je namenjena inicializiranju spremenljivk. S pisanjem kode v teh funkcijah lahko kreiramo in kontroliramo funkcionalnosti igralnega objekta, ki vsebuje našo skripto.

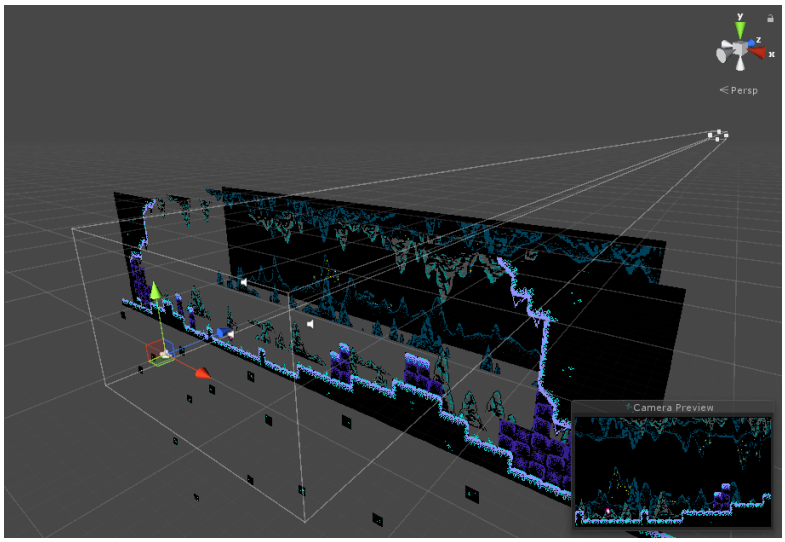
Za potrebe moje igre sem napisal 40 skript, s skupno 3000 vrsticami kode (brez upoštevanja vmesnih, praznih vrstic). Poleg tega sem uporabljal že obstoječe komponente, ki so na voljo v pogonu.

Med tem procesom sem uporabljal začasne slike in zvoke, saj sem se razvoja ostalih vidikov lotil šele, ko sem imel dokončan prototip igre, z vsemi igralnimi mehanikami. Vsako novo dodano vsebino sem testiral posebej ter nato še v igri, z ostalimi igralnimi objekti. Ko je nova vsebina delovala pravilno in je popestrila igranje, v skladu z oblikovalnimi cilji igre (določeni v predprodukciji, v dokumentu koncepta), sem se lotil razvoja naslednje funkcionalnosti.

4.1.1 2D igra v razvojnem okolju Unity

Pred začetkom razvoja igralnih mehanik je potrebno prilagoditi razvojno okolje. Unity ponuja možnost razvijanja v 2D načinu, ki nam določi nekatere privzete nastavitve urejevalnika in s tem olajša delo na 2D projektu. Ena bolj pomembnih je nastavitev kamere na ortografsko projekcijo.

Ortografska (pravokotna) projekcija je tip planarne projekcije (preslikave 3D predstavitve objekta na 2D ravnino), kjer so projekcijski žarki vzporedni in pravokotni na projekcijsko ravnino. V primerjavi s perspektivno projekcijo, kjer žarki konvergirajo v isto točko (na primer oko), se pri ortografski projekciji nadaljujejo v neskončnost, kot da bi zanemarili globino ali z os. V praksi to pomeni, da so predmeti blizu kamere enake velikosti, kot tisti ki so bolj odaljeni - izgubimo občutek perspektive. Kot v večini 2D iger bom tudi v moji igri uporabil ortografsko kamero (prikazana na sliki 4.1).



Slika 4.1: Ortografska kamera moje igre. Vidne so tudi posamezne plasti ozadja.

Poleg privzetih 2D nastavitev je potrebno nastaviti še dodatne lastnosti, da lahko Unity pravilno prikaže retro izgled moje igre. V Pixel art slogu želimo, da so posamezni piksli razločni in ostri. V ta namen so teksture

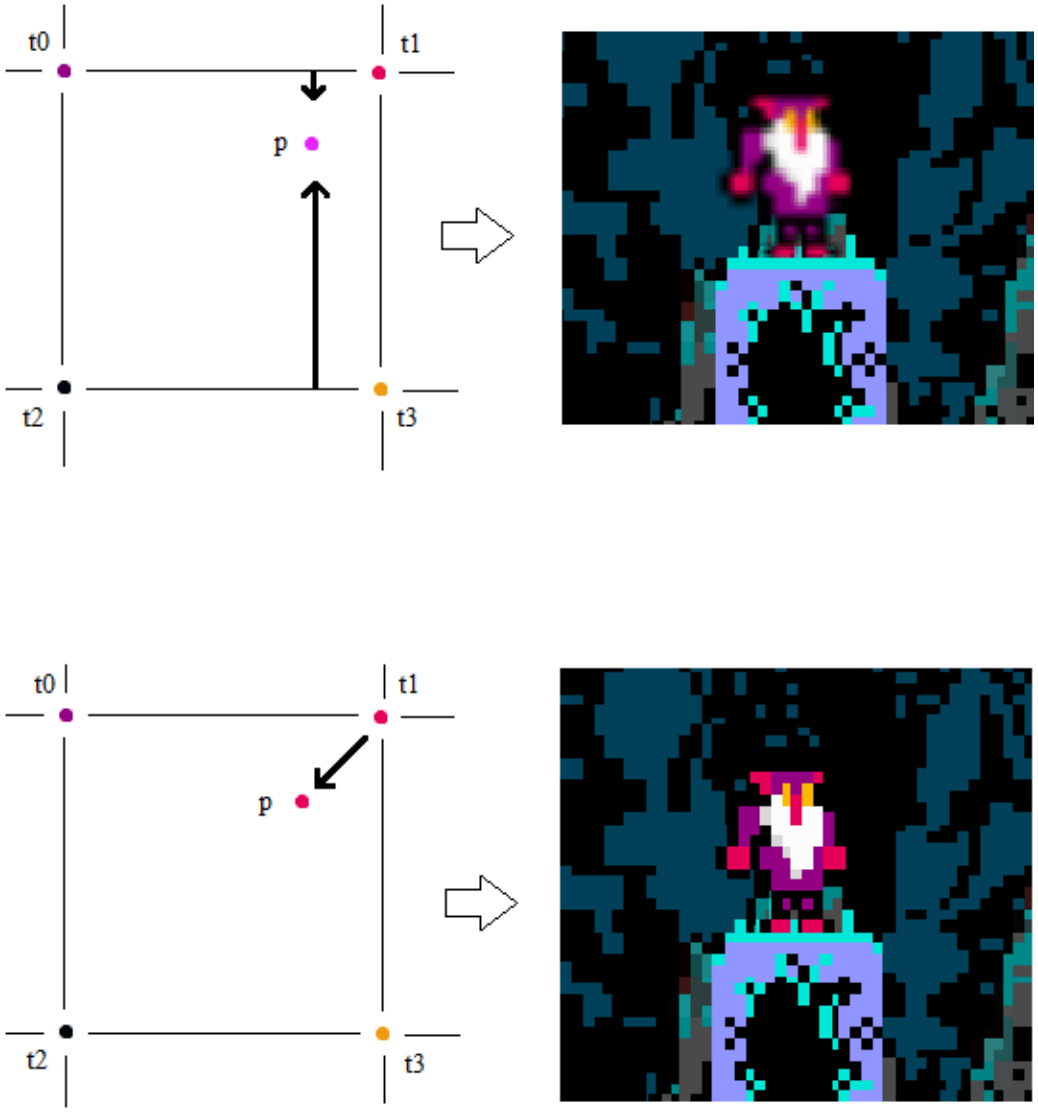
namensko narejene v nizki ločljivosti.

Tako kot je digitalna slika predstavljena s poljem vrednosti, ki jih imenujemo piksli (angl. pixel - okrajšava za picture element), je tudi tekstura predstavljena s teksli (angl. texel - okrajšava za texture element). To je osnovni element teksture. Med izrisom teksture na igralni objekt, ki zavzema določeno območje zaslona, se vsakemu tekslu določi, v katere piksle objekta se bo preslikal. Zgodi se, da zaradi različne velikosti teksture in objekta posamezni piksel ne pade vedno direktno na posamezni teksel. Da določimo barvo takšnega piksla, je potrebno filtriranje z različnimi metodami.

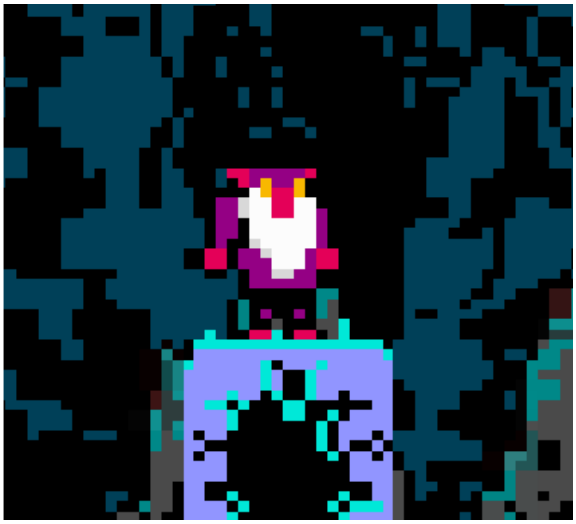
Ponavadi se uporablja bilinearna interpolacija, kjer določimo barvo piksla z uteženim povprečjem štirih najbližjih tekslov okoli njega. Ta metoda zgladi prehode med posameznimi teksli, kot lahko vidimo na sliki 4.2. Za igro v Pixel art slogu takšen rezultat ni primeren.

Zato je pomembno, da za uvoženo teksturo metodo filtriranja nastavimo na interpolacijo najbližjega soseda. Pri tej osnovni metodi se vsakemu pikslu pripiše barva najbližjega teksla. Vsak teksel v teksturi se tako poveča v več pikslov in ohrani kvadratast izgled.

Pri tem lahko nastane problem, ko število tekslov ne ustreza večkratniku števila pikslov. Če imamo na primer teksturo dimenzij 4x4 tekslov, ki se bo izrisala na območje zaslona dimenzije 38x38 pikslov, bo vsak teksel obsegal 9,5 pikslov zaslona ($38/4 = 9{,}5$). Ker ne izrisujemo polovice piksla, bo včasih določen teksel velik devet pikslov, včasih deset. Piksli bodo tako neenakih velikosti. To je še posebej opazno pri translaciji (premikanju) objekta, katerega piksli vsako sličico spreminjajo velikost in povzročijo efekt utripanja. Efekt je viden na sliki 4.3



Slika 4.2: Primer bilinearne interpolacije (zgoraj) in interpolacije najbližjega soseda (spodaj) piksla p, glede na bližnje teksle t0, t1, t2 in t3.



Slika 4.3: Popačene velikosti pikslov - posledica nepravilnega izrisa tekstur.

Problemu se izognemo, tako da v urejevalniku ustrezno nastavimo velikost ortografske kamere in PPU teksture. Gostota pikslov ali PPU (pixels per unit) nam pove, koliko pikslov teksture (tekslov) bo zasedalo eno enoto [1]. Z velikostjo kamere določimo, koliko enot obsega polovica zaslona.

Spodaj je prikazana tabela 4.1 potrebnih velikosti kamere, izračunanih po sledeči enačbi:

$$VelikostKamere = \frac{(vertikalnaResolucija)}{(PPUskala * PPU))} * 0,5 \tag{4.1}$$

V prvem primeru je velikost ortografske kamere enaka 33,75. To pomeni, da je vertikala zaslona razdeljena na 67,5 (33,75*2) enot in posledično je ena enota velika 16 pikslov (1080/67,5 = 16). Tako se bo 16 tekslov teksture pravilno preslikalo v 16 pikslov zaslona, vendar bodo posamezni piksli nerazločni zaradi velike resolucije.

[1] Unity uporablja abstraktno enoto imenovano preprosto „enota“, ki določa velikost sveta igre in se uporablja v fizikalnih izračunih. Velikokrat jo uporabniki enačijo z metrom - torej ena enota je enaka enemu metru v igri.

resolucija	PPU	PPU skala	velikost kamere
1080	16	1	33,75
1080	16	5	6,75
216	16	1	6,75

Tabela 4.1: Velikost ortografske kamere glede na resolucijo, PPU in PPU skalo.

Če želimo doseči izgled nizke resolucije, preprosto skaliramo posamezne piksle. Če, kot v drugem primeru, za zaslon resolucije 1080 pikslov nastavimo PPU na 16 in velikost kamere na 6,75, povzročimo petkratno skaliranje, kar pomeni, da se bo vsak teksel teksture preslikal v 5 pikslov zaslona (80/16 = 5). Tako dosežemo navidezno resolucijo 216 pikslov in enak izgled kot v tretjem primeru tabele.

Skaliranje bo delovalo le za resolucije enakega razmerja prikaza (aspect ratio), ki določa razmerje med dolžino in višino zaslona. Pri ostalih bodo teksture postale popačene, ker se teksli nepravilno preslikajo v necelo število pikslov. Če želimo pravilen izris na vseh zaslonih, imamo na voljo tri rešitve:

1. Ohranimo višino vidnega polja in povečamo širino, tako da dosežemo željeno razmerje prikaza. S tem pokažemo več sveta naše igre, kot če bi oddaljili pogled.

2. Lahko ohranimo širino in povečamo višino vidnega polja. Tako povečamo pogled, vendar pokažemo manj sveta igre.

3. Če nočemo spremeniti velikosti vidnega polja, ker ne želimo, da igralec vidi več ali manj igre, kot smo si zamislili, moramo ob straneh dodati črne robove. S tem zapolnimo dodatni prostor in ohranimo prvotno razmerje prikaza.

Sam sem igro izdelal v razmerju prikaza 16:9, ker je to najbolj razširjeno razmerje na osebnih računalnikih, ki so ciljna računalniška okolja moje igre.

Za ostale resolucije sem uporabil skripto, ki doda črne robove, da se primerno prilagodi zaslonu.

4.1.2 Simulacija prostorskih planov

Z uporabo ortografske kamere smo igralcu odvzeli občutek perspektive. Starejše 2D igre so to težavo rešile z uporabo simulacije prostorskih planov. S premikanjem plasti ozadja z različnimi hitrostmi (angl. Parallax scrolling) ustvarimo navidezno globino in naredimo sceno bolj dinamično.

Napisal sem skripto, ki omogoča kameri, da sledi igralcu. Ta pridobi vektor pozicije igralca in če se je ta premaknil za določeno razdaljo, s transformacijo premakne kamero na igralčevo novo pozicijo. Pri vseh transformacijah vektorjev pozicije je potrebno uporabiti funkcijo, ki postopoma z glajenjem spremeni vektor, tako da prehod ni preveč diskreten in skakajoč.

Nato sem v novi skripti opisal algoritem, ki premika objekt glede na premike kamere. Razdaljo, ki jo je prepotovala kamera v eni sličici (frame) uteži z negativnim skalarjem. Za to uteženo razdaljo nato premakne objekt v nasprotno smer. Skripto sem dodal vsem petim plastem ozadja moje igre in vsakemu določil drugačen skalar - bolj je plast oddaljena od ospredja, manjši skalar ima in počasneje se premika.

4.1.3 Igralec

Unity za 2D objekt ne ponuja že vnaprej narejenega krmilnika, zato sem poleg dodatnih funkcionalnosti igralca naredil tudi osnovne. Najprej sem omogočil premikanje. Preprosto simuliranje sil, ki delujejo na objekt in ga premikajo, lahko sprogramiramo sami, možno pa je za to uporabiti že vgrajen fizikalni pogon. Če objektu dodamo komponento Rigidbody2D, ga postavimo pod simulacijo fizikalnega pogona. Zato se, za razliko od ostalih objektov, simulirani objekti ne smejo premikati s spreminjanjem vektorja pozicije - fizikalno neutemeljen premik bi povzročil težave v ostalih izračunih. Od izbire tipa telesa je odvisno, katere sile bodo delovale na objekt in kako se

bo premikal:

1. Dinamično telo je glede na porabljene vire najdražji tip telesa, saj na njega delujejo vse sile fizikalnega pogona (gravitacija, trenje, ...). Določimo mu lahko različne parametre, kot sta masa in linearni upor. Dinamično telo se premika s spreminjanjem hitrosti, kar lahko neposredno določimo v skriptah z dodajanjem sile, ali pa posredno v interakciji z drugimi objekti.

2. Kinematično telo uporabimo, ko želimo sami natančno opisati vsa pravila premikanja in interakcije z drugimi objekti. Sile fizikalnega pogona nimajo vpliva na kinematično telo, zato ima manjšo porabo virov kot dinamično telo.

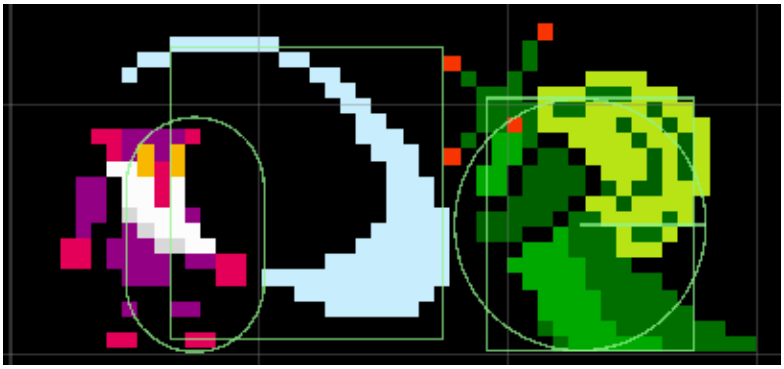
3. Statično telo je nepremičen objekt, ki ni namenjen premikanju in ne vsebuje nikakršne simulacije.

Igralec in sovražniki imajo dinamični tip telesa. Ker v igri želim preprosto, arkadno premikanje objektov, sem poenostavil delovanje dinamičnega telesa in uporabil le osnovne lastnosti fizikalnega pogona (predvsem gravitacijo). Skripta namesto s silo premika objekt z direktnim dodajanjem hitrosti. S tem sem odstranil pospeševanje in upočasnjevanje, kar naredi premikanje nerealistično, ampak bolj odzivno in gibčno. Odstranil sem tudi upor in trenje objektov - objekt se premika gladko in se ustavi tisti trenutek, ko zaznamo spremembo vnosa. Vnos se zaznava v funkciji Update, ki se izvede vsako sličico igre, medtem ko funkcionalnosti Rigidbody teles opišemo v funkciji FixedUpdate, ker je sinhronizirana s časovnim korakom fizikalnega pogona. S tem se izognemo morebitnim zamudam ali prezrtim vnosom.

Poleg premikanja objektov moramo tudi zagotoviti interakcijo med njimi. To lahko dosežemo z uporabo trkalnikov. Trkalnik je komponenta, ki določa obliko objekta v fizikalnem pogonu. Trkalnik prilagodimo objektu s kombiniranjem primitivnih oblik (kvadrat, krog, cilinder). Tako dosežemo zadovoljive rezultate z manjšo porabo virov, kot če bi uporabili po meri narejen trkalnik.

Ko se dva trkalnika dotakneta, pride to trka. Pogon izračuna fizikalni odziv in sproži klic določenih funkcij v vseh pripadajočih skriptah. Tako lahko v funkciji OnCollisionEnter opišemo akcijo, ki se bo zgodila v prvi sliki trka. Podobno velja za funkciji OnCollisionStay in OnCollisionExit.

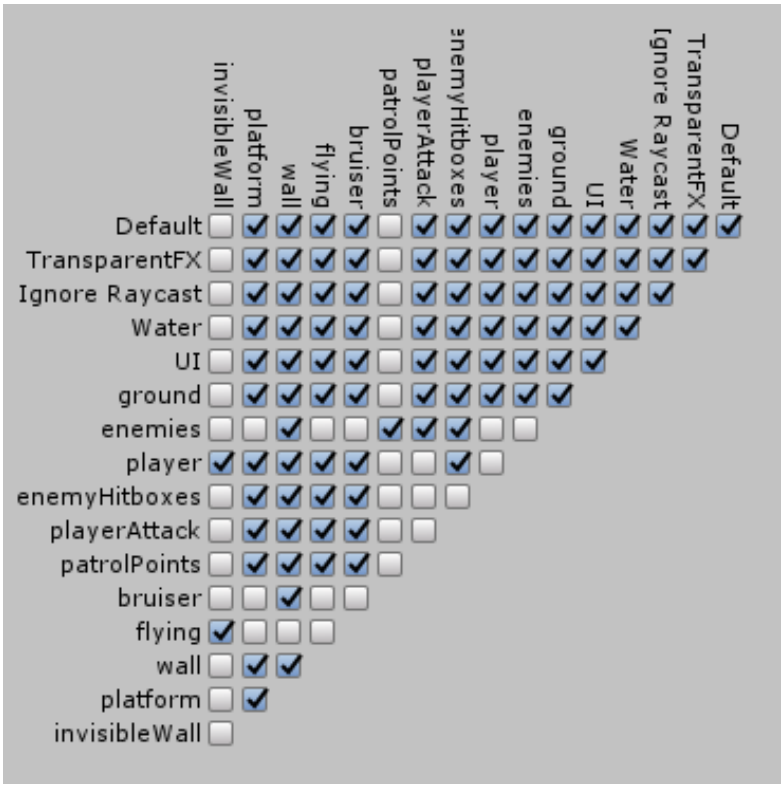
Trkalniku lahko onemogočimo fizikalni odziv in ga uporabljamo samo kot sprožilec. To sem uporabil pri kreiranju bojevanja. Igralcu sem za vsak napad dodal neaktiven trkalnik, ki je nastavljen kot sprožilec. Ko krmilnik zazna vnos tipke za napad, aktivira ta sprožilec za čas napada. Če pride do trka med sprožilcem napada in sovražnikovim trkalnikom, se skupaj z atributi napada (koliko škode naredi, kolikšna sila se bo generirala, ...) pošlje sporočilo sovražnikovi skripti, kjer je opisan odziv na igralčev napad. Na sliki 4.4 so vidni trkalniki igralca in sovražnika).



Slika 4.4: Trkalniki, vidni kot zelene črte okoli objekta, predstavljajo njegovo fizikalno obliko v pogonu.

Potrebno je tudi nastaviti matriko plasti trkov, ki določa, katere fizikalne plasti so lahko v interakciji med sabo. Objektu in posledično trkalniku nato določimo, na kateri fizikalni plasti obstaja. To funkcionalnost sem v razvoju krmilnikov uporabil velikokrat. Ker sem želel, da se sovražnik lahko premika čez druge sovražnike in igralca, vendar ga pri tem poškoduje, sem trkalniku sovražnika dodal drugo fizikalno plast kot sprožilcu njegovega napada. Prva plast je v interakciji s plastjo tal nivoja, da se lahko sovražnik premika po tleh, medtem ko je druga plast v interakciji le s plastjo igralčevega trkalnika,

tako da lahko povzroči napad. Na podoben način sem dosegel, da leteči sovražnik lahko leti čez trkalnike tal nivoja in da velikan odrine igralca. Na sliki 4.5 je prikazana celotna matrika plasti trkov.



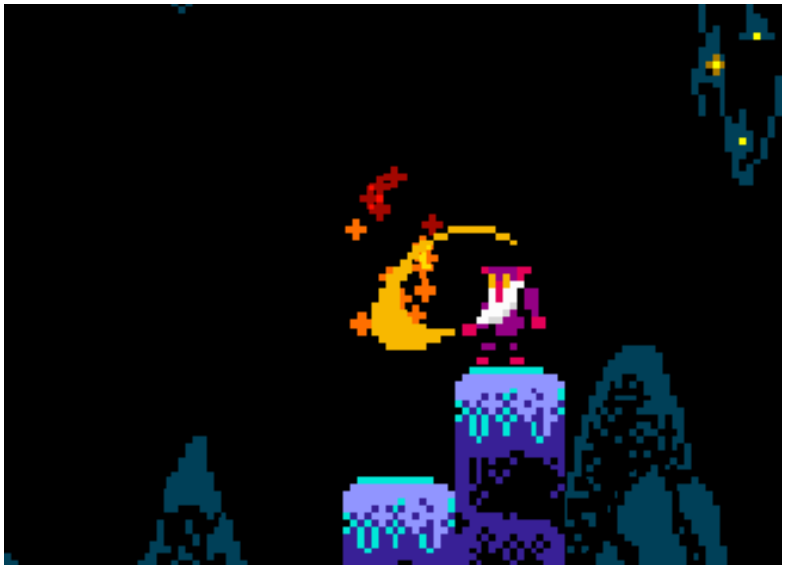
Slika 4.5: Matrika plasti trkov.

4.1.4 Nadgradnje

Igralec vsebuje tudi skripto, ki s trkalniki preverja, kdaj pride v stik s truplom sovražnika. Truplo odstranimo z izbrano akcijo (pojej ali razišči) in glede na izbiro pridobimo točke. Zgodi se, da pogon zazna stik s truplom in nato po kratkem času neaktivnosti, zaradi optimizacije, preneha s simuliranjem trka. Da se takšnim situacijam izognemo, mora biti eno od teles (ponavadi glavni igralec) označeno z opcijo never sleep, kar pomeni, da ga pogon nikoli ne

bo optimiziral. S prištevanjem točk se igralcu odklene dostop do naslednje stopnje moči in pripadajoče nadgradnje.

Nadgradnje spremenijo parametre igralca, na primer povečajo njegovo zdravje ali hitrost napada, onemogočijo trkalnik ter ga tako naredijo nesmrtnega za nekaj časa. Ena od nadgradenj vsebuje tudi efekt ognja, narejen s sistemom delcev. Sistem delcev (angl. particle system) nadzoruje življenje delcev - majhnih slik, ki v velikih količinah predstavljajo enitete, ki jih je težko upodobiti z običajnimi 3D ali 2D objekti, na primer dim, tekočine ali iskre. Delcem lahko spreminjamo hitrost ustvarjanja in premikanja, barvo, obliko, vplivamo nanje s silami, itd. Ker starejše igre niso vsebovale takšnih efektov, sem v svoji igri prilagodil sistem delcev, tako da efekt izgleda bolj enostaven in stiliziran (viden na sliki 4.6).

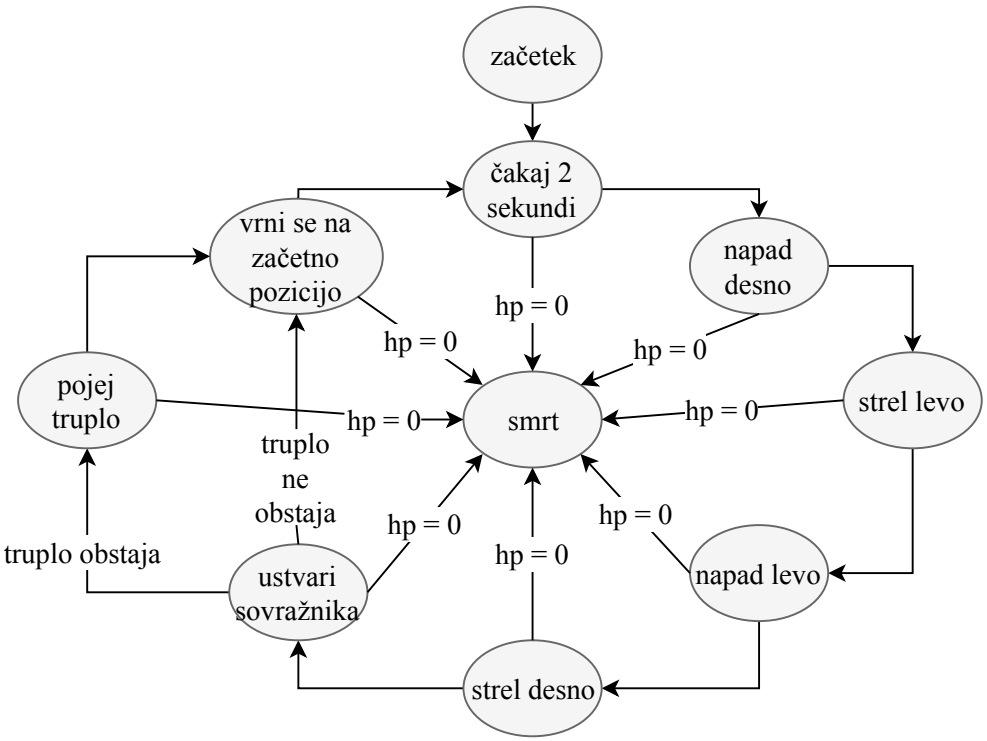


Slika 4.6: Stiliziran efekt ognja, narejen s sistemom delcev.

4.1.5 Sovražniki

Krmilnik sovražnikov ne upravlja z objektom s pomočjo uporabnikovega vnosa, ampak deluje kot končni avtomat. Glede na okolje sovražnika in tre-

nutno stanje preklaplja med ostalimi stanji avtomata. V stanjih so opisane različne akcije, ki jih lahko izvaja sovražnik. Od tipa sovražnika je odvisno, kako kompleksne so te akcije in koliko jih ima na voljo - osnovni vojak se samo premika levo in desno, medtem ko ima šef 7 stanj s pripadajočimi akcijami. Avtomat stanj šefa lahko vidimo na sliki 4.7. Kljub temu večina sovražnikov deluje podobno: sledijo igralcu in sprožijo napad, ko ugotovijo, da so v dosegu. Pri tem uporabljajo podobno logiko za napade in premikanje, ki je opisana zgoraj za igralca.



Slika 4.7: Avtomat stanj šefa. V naslednje stanje preide, če je izpolnjen določen pogoj ali če se konča akcija stanja. Hp je okrajšava za življenje.

Če se vsi sovražniki preprosto premikajo proti igralčevi poziciji, pride do težave. Ker lahko prehajo drug čez drugega, se čez čas vedno zlijejo skupaj v gmoto objektov. Da bi se temu izognil, sem naredil algoritem, ki poskrbi, da

se sovražniki premikajo različno, vendar vedno v bližini igralca. Po nivoju sem postavil sprožilce, ki skupaj s preprekami delujejo kot kontrolne točke, med katerimi se bo premikal sovražnik. Sovražnik vsako sliko predse izstreli žarek za določeno razdaljo. Ko se žarek dotakne kontrolne točke, se glede na trenutno pozicijo igralca določi novo smer premikanja. Če se igralec nahaja v nasprotni smeri kot sovražnik, se le ta obrne. Če pa se igralec nahaja v isti smeri, sovražnik ohrani svojo smer premikanja. Na koncu sovražnik izvede premik v določeni smeri in v primeru, da žarek zazna kontrolno točko, ki je prepreka, še skok. S tem pridobimo tudi boljšo igralnost in občutek bitke, saj so sovražniki bolj razpršeni po celotnem nivoju. Osredotočimo se lahko na bitko s posameznimi sovražniki in ne samo na bežanje pred njimi.

Poleg logike posameznih sovražnikov mora igra vsebovati tudi logiko, ki skrbi za njihovo ustvarjanje in valove napada. Vsak val je sestavljen iz objektov, imenovanih ustvarjalci. Postavljeni so na različnih pozicijah v nivoju in vsebujejo skripto, ki ustvarja določene sovražnike.

Pri tem sem si pomagal s t.i. montažnimi (angl. Prefab) objekti. Montažni objekt je predloga nekega objekta, ki se pogosto pojavlja v igri. Objekti, narejeni iz te predloge, so že vnaprej nastavljeni in vsaka sprememba predloge se odraža v vseh instanciranih objektih. Tako nam ni potrebno ročno dodajati in nastavljati vsakega sovražnika posebej. V igri sem vsak tip sovražnika shranil kot predlogo, iz katere nato ustvarjalec ustvari novega sovražnika.

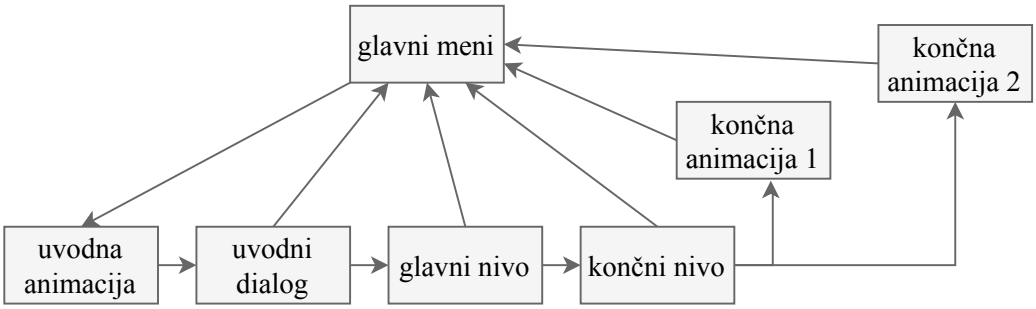
Da se sovražniki ustvarjajo postopoma in ne vsi naenkrat, je potreben časovni zamik. To je možno narediti v funkciji Update, z odštevanjem preteklih slik lahko kreiramo osnoven števec. Bolj primerna rešitev pa je uporaba korutin. Korutina je funkcija, ki lahko prekine izvajanje na določenem delu in nadaljuje z izvajanjem naslednjo sliko. S tem se razlikuje od običajne funkcije v Unity pogonu, ki se mora popolnoma končati do izteka slike.

Koliko sovražnikov, katerega tipa in koliko časovnega zamika bo med njimi, so javne spremenljivke skripte, ki jih nato lahko določimo za vsak ustvarjalec v Unity vmesniku. Ko se ustvarjanje sovražnikov v valu konča, se to sporoči objektu gospodar igre (angl. Game Master - GM), ki nato

pregleduje objekte v igri in ugotavlja, ali je igralec odstranil vsa trupla. Tudi tukaj se uporabi časovni zamik s korutino. Pregled celotne hierarhije objektov v igri je zamudno in procesorsko zahtevno opravilo, zato se pregled izvede le vsako sekundo (namesto vsako sliko). S tem optimiziramo delovanje igre brez vpliva na igralnost. Ko trupel ni več, GM obnovi igralcu zdravje in energijo, naznani ter sproži nov val, ki vsebuje nove ustvarjalce.

4.1.6 Gospodar igre

Poleg valov sovražnikov objekt gospodar igre (GM) nadzoruje potek igre. Tako omogoča premor igre z nastavljanjem skale časa na 0, prikazuje uporabniški vmesnik in skrbi za pravilno menjavanje scen. Ponavadi je igra sestavljena iz več delov, imenovanih scene, ki si jih lahko predstavljamo kot različne nivoje (čeprav je možno imeti več nivojev v eni sceni, to ni običajna praksa). Tudi krajša igra, kakršna je moja, vsebuje 7 različnih scen, katerih potek je prikazan na sliki 4.8.



Slika 4.8: Potek scen igre.

Vsaka scena ima svojega GM, ki upravlja potek igre v trenutni sceni in ob izpolnjenem pogoju preklopi na drugo sceno. Vsi elementi in objekti scene se ob preklopu izbrišejo in nato ponovno naložijo v prvotnem stanju. To je primerno pri resetiranju glavnega nivoja, saj želimo, da igralec ob smrti začne od začetka. Ob preklopu na zaključni nivo pa želimo, da igralec ohrani svoje pridobljene nadgradnje, zato mu moramo dodati funkcijo DontDestro-

yOnLoad, ki ohrani objekt med preklopom.

Pri tem je potrebno biti pazljiv, da skozi igro obstaja le ena, pravilna verzija igralca. Zato GM glavnega nivoja ob smrti uniči igralca, medtem ko ga GM zaključnega nivoja oživi - po animaciji smrti mu napolni zdravje in energijo, ga postavi na začetno pozicijo in naloži sceno. Tako objekt igralca ostaja isti, čeprav imamo med igranjem občutek, da je umrl.

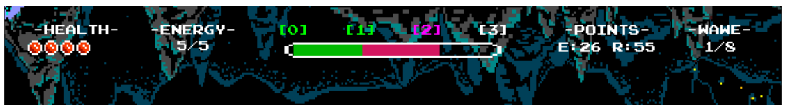
4.1.7 Uporabniški vmesnik

Računalniška igra je programska oprema, ki je v interakciji z igralcem. Da lahko pride do te interakcije, potrebuje uporabniški vmesnik (UV), s katerim lahko človek upravlja z igro in s katerim igra posreduje informacije človeku. Prve igre so imele človeku neprijazen tekstovni uporabniški vmesnik, medtem ko danes večina iger uporablja grafičnega (graphical user interface, GUI). Grafični uporabniški vmesnik igre obsega različne menije in informacije o stanju igre, ki so med igranjem prisotne na zaslonu. Ta del grafičnega vmesnika je v igrah imenovan HUD ali head-up display. Vsaka igra ima drugačen GUI, ki je odvisen od igralnosti, vizualnega sloga in ciljnega računalniškega okolja.

V igralnem pogonu Unity grafični uporabniški vmesnik kreiramo na objektu platno (angl. Canvas). Platno je območje, ki se izriše čez sceno igre. Znotraj njega so vsebovani vsi elementi vmesnika. Vsakemu elementu lahko določimo sidro, ki kontrolira pozicijo elementa med raztezanjem platna. Na to vpliva tudi hierarhija elementov, saj je otrok zasidran glede na pozicijo elementa očeta. Tako sem za sidro HUD-a igralca določil celozaslonski razteg, medtem ko so posamezni elementi zasidrani od levega do desnega zgornjega kota, kar omogoča pravilen razteg pri vseh velikostih ekrana.

HUD igralca (prikazan na sliki 4.9) je sestavljen iz besedil, ki prikazujejo zdravje, energijo, pridobljene točke in številko sovražnikovega vala. Pod besedilom se nahaja pripadajoče vnosno polje, v katerega se napiše trenutno stanje parametra. Zanimiv je element, ki prikazuje napredek igralca skozi stopnje nadgradenj. Sestavljen je iz slike, besedila in dveh drsnikov različnih barv (za vsako kategorijo nadgradnje eden), ki se prekrivata. Skripta premika

drsnik glede na pridobljene točke in, ko doseže mejo naslednje nadgradnje, obarva pripadajočo številko z barvo drsnika, ki jo je dosegel. Tako vizualno sporočimo igralcu razmerje in napredek dveh kategorij točk in katere nadgradnje je odklenil s katero kategorijo.



Slika 4.9: HUD igralca.

Poleg menijev in HUD-a je v igri prisotno še okno dialoga, s katerim igralcu posredujemo zgodbo med igranjem ali animacijo. Sestavljeno je iz slike ozadja, na katerem leži tekstovno polje, v katerega se izpisuje dialog. Skripta, ki kontrolira izpisovanje, vsebuje polje nizov (Strings), ki se nato v zanki s časovnim zamikom postopoma izpisuje, črko po črko. To skupaj z zvokom ustvari dinamičen efekt, ki je zanimivejši od preprostega prikaza besedila. Ko se niz zaključi, z gumbom sprožimo naslednji izpis.

4.2 Izgled

Končni izgled igre je viden na sliki 4.10.

4.2.1 Omejitve

Omejitve so pri kreiranju umetnosti ključnega pomena. Zaradi njih je umetnik primoran postati kreativen in čimbolj izkoristiti sredstva, ki jih ima na voljo. Vsak medij vsebuje svoje omejitve, ki oblikujejo izgled nastalih del. Slog Pixel art, ki definira izgled moje igre, se je porodil zaradi strojnih omejitev zgodnjih konzol in računalnikov. Pri kreiranju bom okvirno upošteval lastnosti konzole NES (Nintendo Entertainment System) iz leta 1985 [36].

Zgodnje konzole so imele (v primerjavi z današnjimi) zelo malo pomnilnika; NES ima le 4 KB. Ta pomnilnik je moral zadoščati za potrebe CPE in grafične

Slika 4.10: Zajem zaslona igre.

enote. NES ima resolucijo 256x240 pikslov. Če bi želeli, da v pomnilniku shrani trenutno sliko, ki jo bo izrisal na zaslon, bi za vsak piksel potrebovali 1 bit (tako bi imeli monokromatsko sliko, le iz dveh barv), bi potrebovali 7680 bajtov pomnilnika - skoraj dvakrat več, kot ga ima na voljo. Kako je lahko potem NES omogočal igranje iger v barvah? Z iznajdljivimi rešitvami so inženirji zaobšli strojne omejitve sistema in s tem določili prepoznavne lastnosti naprave.

Ključna lastnost je razdelitev zaslona na ploščice (velikosti 8x8 pikslov) in bloke (velikosti 16x16 pikslov). Grafična enota PPU vsaki ploščici določi sliko, ki je shranjena v kaseti igre. To sliko nato obarva, tako da vsakemu bloku določi eno izmed štirih barvnih palet. Barvna paleta lahko vsebuje največ štiri barve, pri čemer mora biti ena od njih barva ozadja - v vsakem trenutku imamo torej na voljo 13 barv. S tem prihranimo veliko pomnilnika, saj omejimo število možnih barv, s katerim lahko pobarvamo posamezni piksel v blokih. To povzroči tudi značilen kvadratast izgled, ki bo prisoten tudi v moji igri.

Poleg tega NES, za potrebe optimizacije, uporablja *sprite*. V primerjavi z današnjim poimenovanjem, ki označuje vse 2D slike v igri, je pravi *sprite* specifičen 2D grafični element, ki potrebuje pripadajočo strojno opremo. Je

8x8 pikslov velika bitna slika, neodvisna od mreže ploščic in blokov. Upravlja jih grafična enota, ki jim spreminja atribute (pozicijo, barvo) med samim izrisom na zaslon. Zaradi 32 bajtov velikega medpomnilnika lahko PPU prikaže le 8 *spritov* v eni liniji izrisa (scanline). Če to ne velja, začnejo *spriti* utripati. Tega ne bom upošteval pri moji igri. Držal pa se bom omejitve, ki narekuje, da je lahko posamezni *sprite* pobarvan le z eno od štirih možnih palet. Podobno kot pri blokih, ena paleta lahko vsebuje največ 4 barve, kjer je ena od njih barva ozadja. Za vse *sprite* na zaslonu imamo tako na voljo 12 različnih barv.

Barve izbiramo iz glavne NES palete, ki ima na voljo paleto 54 barv barvnega prostora YPbPr. Ta je računalniško generirana s kombiniranjem štirih stopenj svetlosti (komponenta Y) in mešanjem modre (komponenta Pb) in rdeče barve (komponenta Pr). Posledica je zelo hladna paleta, z velikimi koraki med odtenki in odsotnostjo prave rumene barve. To paleto sem uporabil zaradi njenega značilnega retro izgleda. Prikazana je na sliki 4.11.

Slika 4.11: Paleta barv NES naprave.

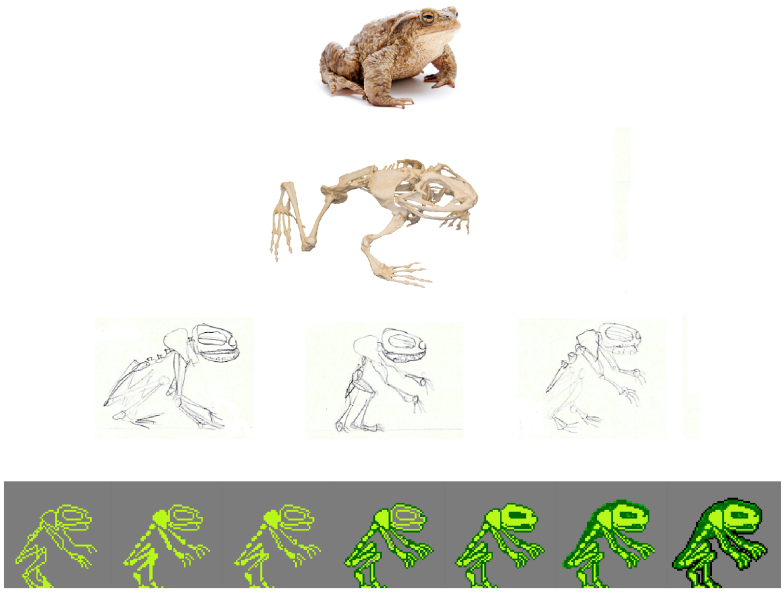
Zaradi razdelitve je večina objektov v NES igrah dimenzij večkratnika števila 8. Podobno je v moji igri: ozadje in nivo bo sestavljeno iz blokov velikosti 16x16 pikslov. Takšne dimenzije bo tudi igralec, medtem ko bodo sovražniki velikosti od 24x16 do 64x64 pikslov.

Kljub pomembnosti omejitev, ki sem jih določil, se jih nisem držal brezpogojno. Uporabil sem sivo, modro in rumeno barvo, ki niso prisotne v originalni paleti. Pri igralčevem *spritu* sem uporabil 6 barv, namesto običajno

štirih. Menim, da to ne pokvari izgleda, saj igra ohrani enoten, prepoznavno retro stil. To pa je bil namen uporabe omejitev.

4.2.2 Risanje

Proces risanja se začne z določanjem oblike. Zaradi omejitev in sloga Pixel art, je potrebno stilizirati in določiti želene karakteristike objekta, ki ga želimo upodobiti, tako da bo nemudoma prepoznaven. Pri tem si pomagamo z referenčnimi slikami, ki smo jih izbrali v predprodukciji. Primer oblikovanja sovražnika je prikazan na sliki 4.12.



Slika 4.12: Potek oblikovanja sovražnika, od začetne referenčne slike do končne računalniške risbe.

Nato je objekt potrebno narisati na računalniku. Pri večjih slikah, kot je ozadje, sem naredil risbo na papirju, ki sem jo nato občrtal v programu (slika 4.13). Za manjši objekt, kot je slika sovražnika ali delček nivoja, je hitreje narisati silhueto direktno v programu.

Sama oblika objekta vpliva na naše dojemanje: objekt, ki vsebuje ostre

Slika 4.13: Skica na papirju in končna slika v igri.

like, deluje zlobno in agresivno, medtem ko mehke linije upodabljajo prijaznost in dobroto. To je vidno v igri. Celotni nivo ima oster izgled, z različni kapniki predstavlja nevarno okolje za našega junaka. Ta je sestavljen iz kvadratov, ki izražajo trdnost in odločnost, primerno za upodobitev škrata. Nasprotno imajo sovražniki večinoma zaobljene linije, saj so mehki, organski in sluzasti.

Objektu nato postopoma dodajamo podrobnosti. Glede na velikost slike določimo najmanjši del objekta, ki ga bomo narisali. Vse manjše podrobnosti zanemarimo ali jih samo nakažemo s spremembo barve ali senčenjem.

Barva je zaradi omejitev pomemben likovni element Pixel art sloga. V igri deluje kot legenda, ki igralcu razločno sporoča, kaj je nivo, kaj sovražnik, junak, ozadje ali ospredje.

Pri tem je koristno poznati osnove barvne teorije, na primer kompozicije barv. Sovražniki v igri so zelene barve, ki je komplementarna vijolični barvi igralca. To ustvari velik kontrast, ki nam pomaga, da v množici razločimo igralca. Nivo je obarvan v analognih barvah modre in vijolične, ki so si sorodne in ustvarijo umirjen ter enoten izgled, na katerem izstopajo ostali objekti. Iz razpoložljivih barv izberemo barvno lestvico odtenkov, s katero bomo obarvali te skupine objektov (slika 4.14).

Slika 4.14: Igralec in sovražnik ter njuni pripadajoči barvni lestvici.

V HSV modelu je barva sestavljena iz treh dimenzij:

1. *hue* ali barvnost, odvisna od valovne dolžine s katero razlikujemo posamezno vrsto barve (na primer rdečo) od drugih.

2. *saturation* ali nasičenost. Visoko nasičena barva je živa, medtem ko nenasičena predstavlja odtenek sivine te barve.

3. *value* ali svetlost, ki določa, koliko bele ali črne barve vsebuje odtenek.

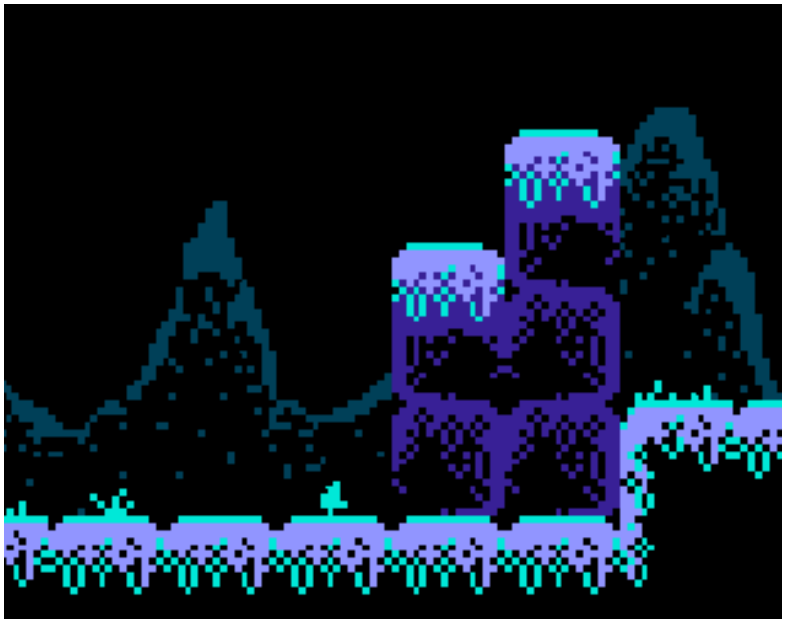
Pri izbiranju odtenkov je dobro, če spreminjamo vsaj 2 barvni dimenziji. Za barvo, s katero sem senčil, sem poleg spremembe svetlosti pomaknil barvnost proti vijolični, pri osvetlitvi določenega dela objekta pa sem izbral barvo, ki se približa rumeni.

Obstaja še četrta dimenzija barve: temperatura. Hladne barve vsebujejo veliko modre, tople pa oranžno-rdeče barve. Ozadje nivoja je hladnejše in temnejše barve kot ospredje, odnos med lastnostmi barv zaznamo kot globino. Poleg tega je bolj oddaljen sloj ozadja narisan manj podrobno, z manjšim številom barv, dokler se na zadnjem sloju ne vidi le obris gore z lučmi mesta. Takšen efekt poudari občutek perspektive.

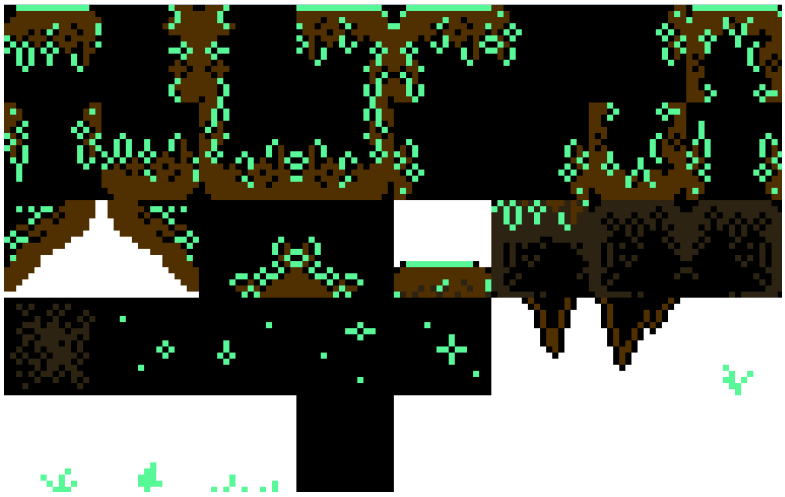
Včasih omejeno število barv ne zadošča našim potrebam. Rešitev je uporaba tehnike, imenovane razpršenje (angl. dithering), kjer s kombiniranjem dveh obstoječih barv ustvarimo gladek prehod in videz nove vmesne barve. Poznamo več vrst razprševanja, v moji igri sem uporabil vzorčno in naključno razprševanje. Ploščice nivoja vsebujejo vzorčno razprševanje, kjer sem barvi kombiniral z risanjem majhnih geometričnih vzorčkov, ki prehodu dodajo okras in zanimivo teksturo. Pri naključnem razprševanju sem kombiniral pike dveh barv brez pravil ali vzorca in s tem ustvaril bolj naraven prehod v sliki ozadja. Vidno na sliki 4.15.

Po mnogih iteracijah končano sliko shranimo v PNG formatu in prenesemo v Unity. Tam se pretvori v tip *sprite*, ki mu določimo lastnosti, kot so PPU in tip filtriranja. Na voljo je tudi urejevalnik, ki nam omogoča, da razrežemo *sprite* na manjše dele in jim določimo fizikalno obliko, kar je še posebej koristno pri ploščicah nivoja.

Namesto, da celotni nivo ročno narišemo v kosu, lahko narišemo različne, 16x16 pikslov velike ploščice, ki sestavljajo paleto nivoja (prikazana na sliki 4.16). Potrebno je biti pazljiv, da se med seboj pravilno in neopazno prilegajo. V Unity nato sceni dodamo objekt mreže, na katero lahko „narišemo“ nivo, kar je fleksibilen in hiter način gradnje nivoja.



Slika 4.15: Primer naključnega (v ozadju) in vzorčnega (v ospredju) razprševanja v igri.



Slika 4.16: Paleta ploščic, s katero gradimo nivo v Unity pogonu.

4.2.3 Animacija

Z animacijo poživimo objekte igre, ne samo igralca in sovražnike, ampak tudi nivo z ozadjem. Pri tem uporabljamo iste tehnike risanja, pozorni pa moramo biti tudi na načela animacije. V knjigi 12 basic principles of animation so leta 1981 Disneyevi animatorji opisali 12 načel, s katerimi dosežemo bolj realistično gibanje. Čeprav so bila namenjena tradicionalni animaciji, veljajo še danes, nekatere sem uporabil tudi pri animiranju sovražnikov in igralca. Na sliki 4.17 so prikazane slike, ki sestavljajo animacije igralca.

Načelo stiskanja in raztega sem uporabil pri teku igralca, ki animaciji doda težo in fleksibilnost. Veliko sem se posluževal načela pretiravanja. Zaradi nizke ločljivosti moramo vsak gib narisati s pretiravanjem, da z omejenim številom pikslov igralcu sporočimo, katero akcijo izvaja objekt. Pri tem nam pomaga tudi načelo pričakovanja, ki narekuje, da gib nakažemo, preden ga izvedemo (priprava na zamah). Poleg risanja je pomembno tudi manipuliranje s časovnimi okvirji. Hitra akcija bo vsebovala manj sličic, medtem ko zamah z mečem prikažemo samo z eno sliko, ki vsebuje razmaz giba. Nasprotno s podobnimi slikami, ki so časovno blizu, ustvarimo občutek počasne, težke akcije (vzgon netopirja).

Da dosežemo naravno animacijo, je potrebno preučiti mehanike gibov, ki jih upodabljamo - tek človeka ali premikanje polža, netopirja.

V Unity končano zaporedje slik sestavimo v animacijo, ki pripada posameznemu objektu. S sistemom Mecanim nato zgradimo mrežo animacij (vidna na sliki 4.18). Med njimi preklapljamo glede na stanja spremenljivk, ki jih določimo v skriptah.

4.3 Zvok

Zaradi igralnosti in izgleda menim, da so za mojo igro še posebej primerni 8-bitni zvoki in glasba, imenovana tudi Chiptune. Chiptune je zvrst elektronske glasbe, ki je močno povezana z igrami in razvojem strojne opreme.

V osemdesetih letih so v naprave začeli vgrajevati programabilne zvočne

čipe, zmožne FM sinteze, ki so lahko generirali nekaj valovnih oblik signala - ponavadi kvadratnega, trikotnega, žagastega in hrup. S spreminjanjem lastnosti, kot je amplituda, so lahko proizvajali različne zvoke.

Ključna lastnost sistemov je bilo omejeno število kanalov, ki so lahko predvajali te preproste zvoke. To je glasbi dalo značilen zvok, vsaka naprava pa je zaradi različnih specifikacij imela svoj prepoznaven ton.

Leta 1985 se je, z izidom Commodor Amige, popularnost Chiptune glasbe povečala, saj je bilo možno zvoke in glasbo ustvarjati v programski opremi Tracker in shraniti v MOD formatu [37]. Ta je vseboval vzorce zvoka in informacije, kako so razporejeni na diskretni časovnici. Tracker je še danes primarni način ustvarjanja Chiptune glasbe, ki je razširjena v retro igrah in računalniški umetnosti Demoscena.

4.3.1 Zvočni efekti

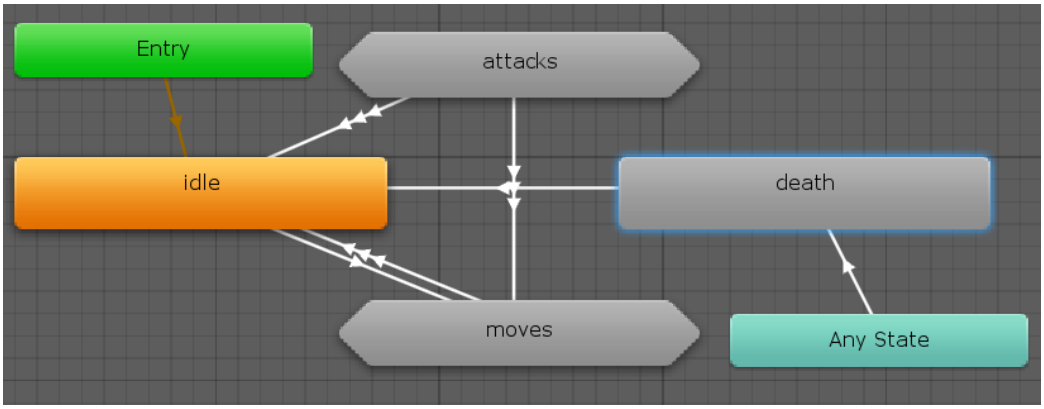
Z orodjem sfxr sem kreiral preproste, kratke zvoke različnih valovnih oblik, ki so značilni za retro igre. Ti poudarijo akcije igralca (na primer strel iz loka) ali mu sporočajo pomembne dogodke (na primer izguba življenja). Poleg tega sem v igro dodal nekaj realističnih ambientnih zvokov (kapljanje vode, šum vetra), ki sem jih pridobil na spletni strani Freesounds.

V Unity objektu dodamo komponento avdio vir, kateremu nato v skripti posredujemo zvočni posnetek, ki ga želimo predvajati.

4.3.2 Glasba

Osnovno Chiptune glasbeno spremljavo bi lahko naredil v enem izmed orodij, kot je FamiTracker. Spremljevalna glasba regulira vzdušje in napatost trenutnega dogajanja igre. Ker nisem glasbeno podkovan, sem se odločil, da uporabim primerno nelicenčno glasbo, na voljo na spletu, ki bo bolje ustrezala tej vlogi.

Slika 4.17: Animacije igralca.



Slika 4.18: Najvišja raven Mecanim sistema animacij igralca, ki skrbi za preklapljanje med glavnimi akcijami igralca.

Poglavje 5

Zaključne faze

Po končani produkciji je potrebno primerno združiti vse dele v zaključeno igro.

5.1 Alfa faza

Prva od zaključnih faz je v industriji poimenovana alfa faza. Vsi elementi igre so implementirani in delujejo, potrebno je le nastaviti njihove lastnosti. V moji igri sem v tej fazi določil obliko črk grafičnega vmesnika, zgradil nivo igre iz prej implementiranih ploščic in prilagodil lastnosti sistema za ustvarjane sovražnikov. Ko lahko igro preigramo od začetka do konca in vsebuje vse željene značilnosti, lahko nadaljujemo z naslednjo fazo.

5.2 Beta faza

V beta fazi se osredotočimo na odpravljanje hroščev in prilagajanje igralčevi izkušnji. V ta namen večje igre testirajo množice testnih igralcev. Mojo igro so testirali prijatelji in družinski člani ter nenazadnje tudi jaz.

Po mnogih igranjih smo odkrili nekaj hroščev, na primer napačna nadgradnja za določeno stopnjo moči in truplo letečega sovražnika, ki je ob smrti včasih padlo čez trkalnike nivoja in tako preprečilo zaključek trenutne sto-

pnje.

Poleg odkrivanja hroščev je pomembno pridobiti mnenja o igri, še posebno negativne kritike. Te nam omogočijo, da odpravimo težave, ki jih sami ne odkrijemo. Igralce moje igre je zmotilo, da ni bilo možno prekiniti dialoga in uvodne animacije. Nekateri tudi niso odkrili sistema odločitev in nadgradenj, saj večina pred igranjem ni prebrala navodil. Vse to so upravičene kritike težav, ki sem jih nato odpravil.

Pomembno je, da s prekomernim prilagajanjem željam igralcev ne ogrozimo celostne vizije igre. Nikoli ne bomo mogli zadovoljiti vseh želj - nekateri igralci igre so se pritoževali, da je igra pretežka, spet drugi, da je prelahka. V igro sem zato samo dodal kodo, ki igralca naredi nesmrtnega. Prav tako nisem implementiral sistema shranjevanja, ker bi to vzelo preveč časa ter uničilo arkadno naravo igre.

5.3 Zlata faza in poprodukcija

Ko je igra naposled končana, preide v zlato (angl. gold) fazo, kjer jo objavimo javnosti in začnemo prodajati.

Po začetni objavi lahko v poprodukciji igro nadgrajujemo z zastonjskimi posodobitvami. To je v času digitalne distribucije postalo še posebno priljubljeno, saj lahko založniki objavijo nedokončane igre z izgovorom, da bodo popravljene naknadno. Prav tako so pogoste naložljive vsebine in mikrotransakcije, ki za manjše plačilo igralcu omogočijo dostop do nove vsebine, neredko celo že obstoječe vsebine, ki je bila namerno izrezana iz igre.

Moje igre nikoli nisem imel namena prodajati, zato sem jo zastonj objavil na spletni strani neodvisnih iger itch.io. Ne načrtujem nobenih posodobitev ali nadgradenj - igra je končana in zaključena celota.

Poglavje 6

Zaključek

V diplomskem delu sem opisal razvoj lastne igre po zgledu ustaljenega procesa razvoja iger, ki se uporablja v industriji. Pri tem sem predstavil vse faze razvoja in pokazal, da je igra produkt, sestavljen iz več enakovrednih vidikov. Razvoj igre je multidisciplinaren projekt, ki zahteva znanje računalništva, likovne umetnosti, glasbe in oblikovanja igralnosti. Za vsako lastnostjo igre se skriva oblikovna odločitev, sprejeta na podlagi nekega tehtnega razloga.

Med razvojem sem naletel na različne ovire, ki sem jih rešil (na primer pravilno prikazovanje pikslov) in težave s hrošči Unity pogona, ki jih včasih ni mogoče odpraviti. Ob vsem tem sem pridobil veliko izkušenj z razvojem iger, ki jih bom uporabil pri naslednjih projektih.

Nekatere funkcionalnosti igre sem opustil, zato se končna igra rahlo razlikuje od začetnega načrta. Če bi imel več časa ali pomoči, bi implementiral zaveznike, ki bi pomagali igralcu v boju, grajenje barikad in možnost pogovora s prebivalci mesta, ki je v trenutni igri predstavljeno le v ozadju. Tudi v večjih produkcijah je pričakovano, da nekatere planirane funkcionalnosti niso prisotne v končnem produktu.

Kljub temu menim, da je igra zaključena celota vseh potrebnih delov in služi kot dober primer razvoja igre.

Literatura

[1] I. Antesic, "Ivan antesic, itch.io profile," Dosegljivo: https://ivan-antesic.itch.io/, [Dostopano: 26. 7. 2018].

[2] Statista, "Film and movie industry - statistics & facts," Dosegljivo: https://www.statista.com/topics/964/film/, [Dostopano: 13. 1. 2018].

[3] E. McDonald, "The global games market will reach $108.9 billion in 2017 with mobile taking 42%," Dosegljivo: newzoo.com/insights/articles/the-global-games-market-will-reach-108-9-billion-in-2017-with-mobile-taking-42/, [Dostopano: 13. 1. 2018].

[4] J. Schreier, *Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made.* Harper Paperbacks, 2017.

[5] B. Sinclair, "Gta v dev costs over $137 million, says analyst," Dosegljivo: http://www.gamesindustry.biz/articles/2013-02-01-gta-v-dev-costs-over-USD137-million-says-analyst, [Dostopano: 13. 1. 2018].

[6] Z. A. Tarn Adams, "Dwarf fortress," Dosegljivo: http://www.bay12games.com/dwarves/, [Dostopano: 13. 1. 2018].

[7] J. Schell, *The art of game design : a book of lenses.* Amsterdam Boston: Elsevier/Morgan Kaufmann, 2008.

[8] I. P. Fletcher Dunn, *3D Math Primer for Graphics and Game Development, 2nd Editione.* CRC Press, 2015.

[9] R. Nystrom, *Game Programming Patterns.* Los Gatos: Smashwords Edition, 2014.

[10] J. Gregory, *Game engine architecture.* Boca Raton, FL: CRC Press, Taylor & Francis Group, 2014.

[11] J. Novak, *Game Development Essentials: An Introduction.* Cengage Learning, 2011.

[12] J. Takumi, "The high concept design document," Dosegljivo: thelegendofjohnny.com/tutorials/high-concept, [Dostopano: 7. 6. 2017].

[13] D. Avison and G. Fitzgerald, *Information Systems Development: Methodologies, Techniques & Tools.* McGraw-Hill, 2006.

[14] the Agile Manifesto authors, "Agile alliance," Dosegljivo: https://www.agilealliance.org/agile101/the-agile-manifesto/, [Dostopano: 29. 1. 2018].

[15] U. Technologies, "Unity," Dosegljivo: https://unity3d.com/, [Dostopano: 30. 1. 2018].

[16] E. Games, "Unreal engine," Dosegljivo: https://www.unrealengine.com/, [Dostopano: 30. 1. 2018].

[17] Y. Games, "Gamemaker," Dosegljivo: https://www.yoyogames.com/gamemaker, [Dostopano: 30. 1. 2018].

[18] Crytek, "Cryengine," Dosegljivo: https://www.cryengine.com/, [Dostopano: 30. 1. 2018].

[19] U. Technologies, "Unity - fast facts," Dosegljivo: https://unity3d.com/public-relations, [Dostopano: 30. 1. 2018].

[20] D. Kvarfordt, "Pyxel edit," Dosegljivo: http://pyxeledit.com/, [Dostopano: 30. 1. 2018].

[21] I. L. Software, "Fl studio," Dosegljivo: https://www.image-line.com/flstudio/, [Dostopano: 30. 1. 2018].

[22] A. team, "Audacity," Dosegljivo: https://www.audacityteam.org/, [Dostopano: 30. 1. 2018].

[23] L. E. Nacke, C. Bateman, and R. L. Mandryk, "Brainhex: A neurobiological gamer typology survey," *Entertainment Computing*, vol. 5, no. 1, pp. 55 – 62, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1875952113000086

[24] E. S. Association, "2017 sales, demographic, and usage data - essential facts about the computer and video game industry." [Online]. Available: http://www.theESA.com

[25] E. S. R. Board, "Esrb," Dosegljivo: http://www.esrb.org/, [Dostopano: 20. 3. 2018].

[26] P. E. G. Information, "Pegi," Dosegljivo: https://pegi.info/, [Dostopano: 20. 3. 2018].

[27] TIGA, "Tiga urges pegi to review its pricing policy to help indies," Dosegljivo: http://tiga.org/news/tiga-urges-pegi-to-review-its-pricing-policy-to-help-indies, [Dostopano: 20. 3. 2018].

[28] Y. C. Games, "Feat unlocked: One million copies of shovel knight sold!" Dosegljivo: http://yachtclubgames.com/2016/04/feat-unlocked-one-million-copies-of-shovel-knight-sold/.

[29] V. Corporation, "Steam," Dosegljivo: http://store.steampowered.com/, [Dostopano: 31. 3. 2018].

[30] S. Galyonkin, "Steamspy," Dosegljivo: https://steamspy.com/, [Dostopano: 31. 3. 2018].

[31] P. Ben Kuchera, "Report: 7,672 games were released on steam in 2017," Dosegljivo: https://www.polygon.com/2018/1/10/16873446/steam-release-dates-2017, [Dostopano: 31. 3. 2018].

[32] J. Campbell, *The Hero with a Thousand Faces (Bollingen Series, No. 17)*. Princeton University Press, 1972.

[33] R. L. Stevenson, *Dr. Jekyll and Mr. Hyde (Wordsworth Classics)*. Wordsworth Editions Ltd, 1997.

[34] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience (Harper Perennial Modern Classics)*. Harper Perennial Modern Classics, 2008.

[35] G. Mata Haggis. Storytelling tools to boost your indie game's narrative and gameplay. Dosegljivo: https://www.youtube.com/watch?v=8fXE-E1hjKk. Youtube. [Dostopano: 31. 3. 2018].

[36] Nes hardware explained. Dosegljivo: http://nerdlypleasures.blogspot.com/2015/06/nes-hardware-explained.html. Nerdly Pleasures. [Dostopano: 13. 6. 2018].

[37] Amiga music preservation faq. Dosegljivo: http://amp.dascene.net/faq.php. Amiga Music Preservation. [Dostopano: 13. 6. 2018].