TALLINNA TEHNIKAÜLIKOOL

INSENERITEADUSKOND

Virumaa kolledž

2D mängu kaardi genereerimine Unity platformil mängu “Soul never Die” näitel

Telemaatika ja arukad süsteemid

Üliõpilane: Dmitri Vorobev

Üliõpilaskood: 178465

Juhendaja: Natalja Maksimova, lektor

Kohtla-Järve, 2021

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.

Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

"...." ..... 20.....

Autor: .....

/ allkiri /

Töö vastab rakenduskõrgharidusõppe lõputööle/magistritööle esitatud nõuetele
"...." ..... 20.....

Juhendaja: .....

/ allkiri /

Kaitsmisele lubatud
"...." ..... 20.....

Kaitsmiskomisjoni esimees .....

/ nimi ja allkiri /

LIHTLITSENTS LÕPUTÖÖ ÜLDSUSELE KÄTTESAADAVAKS TEGEMISEKS JA REPRODUTSEERIMISEKS

Mina Dmitri Vorobev (sünnikuupäev: 20.10.1998)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

Virumaa Kolledži publikatsioonide veebirakenduse loomine, mille juhendaja on Natalja Maksimova,

1.1. reprodutseerimiseks säilitamise ja elektroonilise avaldamise eesmärgil, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta kolmandate isikute intellektuaalomandi ega isikuandmete kaitse seadusest ja teistest õigusaktidest tulenevaid õigusi.

TalTech Inseneriteaduskond Virumaa kolledž

LÕPUTÖÖ ÜLESANNE

Üliõpilane: Dmitri Vorobev, 178465

Õppekava, peaeriala: EDTR17/17 - Telemaatika ja arukad süsteemid

Juhendaja(d): Lektor, Natalja Maksimova, natalja.maksimova@taltech.ee

ettevõte, telefon, e-post

Lõputöö teema:

(eesti keeles) 2D mängu kaardi genereerimine Unity platformil mängu "Soul Never Die" näitel

(inglise keeles) Generating a 2D game map in Unity using the example of the game "The Soul never Dies"

Lõputöö põhieesmärgid:

1. 2D mängu "Souls never Die" kaardi genereerimine

Lõputöö etapid ja ajakava:

Nr	Ülesande kirjeldus	Tähtaeg
1.	Mängumootoris Unity mänguloogika loomise kirjeldus	01.03.2021
2.	Generaatorite uurimine, töö teaduskirjandusega	15.03.2021
3.	Mängu "Souls never Die" maastiku ja stseenide genereerimine vastava stsenaariumile	29.03.2021
4.	Testimine ja testide analüüs	12.04.2021
5.	Lõputöö kirjutusosa vormistamine	10.05.2021

Töö keel: eesti keel **Lõputöö esitamise tähtaeg:** "..28..".mai 2021.a

Üliõpilane: ..... ".....". ..... 20.....a
/allkiri/

Juhendaja: ..... ".....". ..... 20.....a
/allkiri/

Konsultant: ..... ".....". ..... 20.....a
/allkiri/

Programmijuht: ..... ".....". ..... 20.....a
/allkiri/

SISUKORD

EESSÕNA

Käesoleva lõputöö eesmärgiks on 2D mängu kaardi genereerimine. Lõputöö teema oli valitud lähtudes mängu "Souls never Die" loomises osalemise tõttu. Töö koosneb mängu kaartide genereerimise teoreetilisest ja praktilisest osast. Teoreetilises osas vaadeldakse protseduurilise genereerimise viisid ja pseudojuhuslike arvude generaatori algoritm Xorshift. Algoritm on rakendatud programmeerimiskeeles C# ja mänguplatformil Unity loodud mängu pseudolõpmatu kaardid.

Lõputöö teema aitas sõnastada lõputöö juhendaja Natalja Maksimova. Põhilised algandmed koguti teadusartiklitest ja dokumentatsioonidest. Suur tänu juhendajale Natalja Maksimova abi ja õpetuste eest.

Võtmesõnad: protseduuriline genereerimine, pseudojuhuslike arvude generaator, Unity, rakenduskõrgharidusõppe lõputöö

LÜHENDITE JA TÄHISTE LOETELU

RPG	role-playing game
C++	programmeerimiskeel
C#	programmeerimiskeel
Java	programmeerimiskeel
PHP	Personal Home Page
JavaScript	programmeerimiskeel
Xorshift	juhuslike arvude generaatori algoritm
RNG	Random number generaator, juhuslike arvude generaator

SISSEJUHATUS

Videomängud on tänapäeval muutuvad üha populaarsemateks. Mäng on meie jaoks üks võimalusi lõõgastumiseks ja vaba aja veetmiseks**.** Pärast koroonaviiruse puhangut pidid inimesed veetma kodus tohutult aega, mis ainult suurendas huvi mängude vastu, mis tõi paljude mängude müügi kasvu. Mängutööstuse kogumaht oli 2020. aastal 174,9 miljardit dollarit, mis on 19,6% suurem kui 2019. aastal [1].

Kui varem pöörasid inimesed põhitähelepanu suurte tootjate projektidele, siis nüüd on ilmunud tohutu hulk väikeseid stuudioid, mis loovad kvaliteetse ja huvitava toote. Näiteks mäng „*Minecraft*" [2], mis on selle ilmumisest alates muutunud väga populaarseks hoolimata asjaolust, et selle tegi vaid üks inimene - Markus „Notch" Persson. Hea näide on ka stuudio Supergiant Games [3], mis on loonud hulga populaarseid projekte.

Autoril tekkis huvi väikese mõttekaaslaste rühmaga mänguprojekti loomise vastu. „*Soul never Die*" on 2D platvormimäng *rogulike* žanris. *Rogulike* on arvutimängude žanr. Klassikalise *roguelike* iseloomulikud tunnused on juhuslikult genereeritud tasemed ja tegelase surma pöördumatus - tema surma korral ei saa mängija mängu laadida ja peab uuesti alustama. Genereerimist on vaja selleks, et mängija oleks huvitatud mängu uuesti ja uuesti läbimisest. Kuna selles žanris mängutasemed on juhuslikult genereeritud, siis autor valis seda isikliku eesmärgiks, ülejäänud rühmakaaslased tegelesid animatsiooni ja disainiga. Põhinäitena vaatasime mängu „*Hollow Knight*" [4] mis on samuti 2D platvormimäng.

Kaardigeneraatori kood on sõltumatu teisest mängu osadest ja rühmakaaslaste tööst. Juhuslikul viisil mängutasemeid võib genereerida mis tahes piltidel, kuid lõpptulemus sõltub disaineri tööst.

Uurimisobjekt on arvutimäng.

Uurimisaine on arvutimängudes objektide genereerimise meetodid.

Eesmärk - läbitava mängu loomine kaardi juhusliku genereerimisega.

Töö põhiosa on jaotud 3 osaks: 1) vaadeldakse kaardi protseduurilise genereerimise meetodikat; 2) antakse ülevaade pseudojuhuslike arvude kaudu genereerimisest; 3) praktiline rakendamine *Unity*-s [5].

1 PROTSEDUURILINE GENEREERIMINE

Protseduuriline genereerimine on mängude disainimisel väga kasulik tööriist. See muudab mänguarendajate tööd lihtsamaks, võimaldades neil keskenduda mängu teistele aspektidele. Protseduuriline genereerimine on keeruline süsteem paljude tingimuste ja parameetritega [6]. Selle abiga luuakse tohutud suured virtuaalsed maailmad, üksikud tegelased, tasemed, süžeed ja isegi mängureeglid.

1.1 Genereerimise rakendamine mängudes

Kaardi genereerimine kasutati algselt lauamängudes. Mängu "Advanced Dungeon & Dragons"[7] jutustaja (mängija) lõi iga käigu ajal oma maa-alust käiku ja maastikku, sest täringu viskamine on juhuslik sündmus.

Hiljem viis *Don Worth* selle idee 1978. aasta videomängu „*Beneath Apple Manor*" Apple 2 jaoks [8]. Kahe aasta hiljem loodi mäng „*Rogue*"[9] millest sai *rogulike* žanri alusepanija. Mängudes kasutatakse kaardi protseduurilist genereerimist, mis võimaldab mängu iga kord uue loona läbida. Kõik toad, koletised ja aarded genereeriti juhuslikult uue mängu igal käivitamisel.

Tänapäeval kombineeritakse *roguelike* teiste mängužanritega: platvormimängudega nagu mängus „*Rogue Legacy*" [10], rollimänguga (RPG) - mängus „*Diablo*" [11], strateegiatega – mängus „*FTL: Faster Than Light*"[12], dünaamilise protsessiga – nagu mängus „*The Binding of Isaac*" [13] и „*Hades*" [14]. Kõikides nendes mängudes kasutatakse protseduurilist genereerimist erineval määral mängutasemete või varustuse, tegelasrelvade ja vaenlaste jaoks.

Protseduuriline genereerimine võimaldab mängudes luua mitmesuguseid heliribasid. Näiteks mängus „*No Man's Sky*" [15] kasutatakse albumit, mille on grupp *65daysofstatic* [16] salvestanud, ja selle muusikakompositsioone mängitakse süžee võtmehetkedel. Mängus kasutatakse põhiliselt juhuslikult kombineeritud viiside fragmente. Algoritm võtab helisid viiside kogust ja ühendab need kokku, et need sobiksid hetkega.

Protseduurilist genereerimist kasutatakse sagely otsingumängude esemete otsimissüsteemides. Näiteks „*Borderlandsi*" mänguseeria põhineb protseduurilise genereerimise süsteemil, mis võimaldab luua üle miljoni ainulaadse relva ja muu varustuse.

Samuti kasutatakse protseduurilist genereerimist filmides visuaalselt huvitavate ruumide kiireks loomiseks. Maastikud „*The Mandaloriaanis*" [17] luuakse lennult,

kasutades modifitseeritud mootorit *Unreal Engine 4* [18] ja LED-e tulemuse projitseerimiseks, mis filmitakse reaalajas, et näidata täpseid peegeldusi platsil.

1.2 Genereerimise omadused

Protseduuriliseks genereerimiseks nimetatakse mängusisu automaatset loomist eelnevalt kirjutatud algoritmide abil. Loodava sisu põhijoon on see, et mäng peab olema läbitav - mängija peab olema suuteline läbima loodud taseme algusest lõpuni, kasutada genereeritud relva, ronida genereeritud redelil.

Protseduuriline genereerimine peab vastama teatud tingimustele.

Loovus ja usutavus: niisuguse sisu genereerimine, mis näeb välja nagu inimese loodud, mitte generatori.

- Mitmekesisus: sellise sisu loomine, mis oleks võimalikult erinev erinevatel käivitustel ja seda vaadates ei tunneks mängija igavust.
- Kiirus: sõltuvalt ülesandest erinevad nõuded millisekunditest kuudeni, kuid üldiselt peab sisu olema loodud õigeaegselt, et see vastaks mängimise vajadustele.
- Usaldusväärsus: kindlaksmääratud kriteeriumide täitmise garanteerimine, näiteks alati tagada mängijale võimalust läbiga labürindi väljumiseni.

1.3 Genereerimise plussid ja miinused

Plussid:

- Protseduuriline genereerimine võimaldab luua hiigelsuuri maailmu, mis võtaks inimese töötamiseks liiga kaua aega. Seetõttu võivad paljud meie ajal toodud mängud näidata mängijale ilusaid ja suuri lokatsioone.
- Protseduuriline genereerimine võib arvutis oluliselt ruumi kokku hoida. Selle näite võib tuua mängust „*No Man's Sky*", kus mäng loob umbes kaheksateist kvintiljonit planeeti, kuigi see ise kaalub veidi üle kümne gigabaiti.
- Protseduuriliste genereerimistega mängud võimaldavad mängijal saada iga kord, kui ta neid läbib, täiesti uue kogemuse nähtust.
- Protseduuriline genereerimine teeb väga suurt tööd, võimaldades indie-stuudiotel luua kvaliteetseid mänge lühema ajaga.

Miinused:

- Protseduuriline genereerimine ei saa lõputult midagi uut luua. Valikud lõpevad alati ja mängija näeb juba vanu esemeid või maastikku.

- Protseduuriline genereerimine nõuab väga hoolikat häälestamist, kuna see võib genereerimisel väga kergesti vigu teha ja seeläbi mängukogemust rikkuda. Samuti on peaaegu võimatu kõiki genereerimise kombinatsioone ette arvutada ja alati on võimalus teha viga, mis võib mängu kas rikkuda või mängijaid lõbustada.
- Vähendatakse töökohade arv suurtel stuudiotel, kuna protseduuriline genereerimine teeb palju tööd ja ei vaja suurt hulka inimesi sellega töötamiseks. [19]

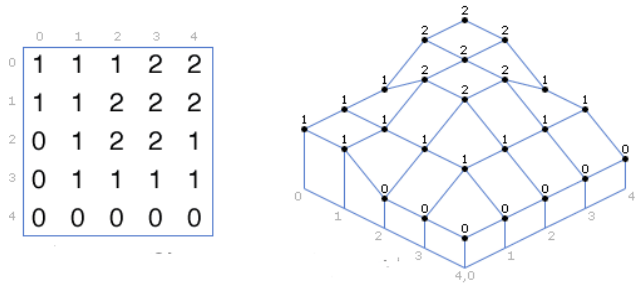
2 MAASTIKU PROTSEDUURILISTE GENEREERIMISTE TÜÜBID

On olemas protseduuriliste genereerimiste palju liike, kuid neist eristuvad eelkõige kolm:

- kõrguste kaartide genereerimine;
- mürakaartide põhjal genereerimine;
- plaaditud maastiku genereerimine.

2.1 Kõrguskaartide genereerimine

Kõrguskaartide genereerimine on piltide konstrueerimine, mis põhineb kahemõõtmelisel massiivil, mille elementideks on maastikualade määratlemise kõrgusväärtused. Joonisel on kuvatud kõrguskaart ja selle põhjal ehitatud maastikuala. [20]



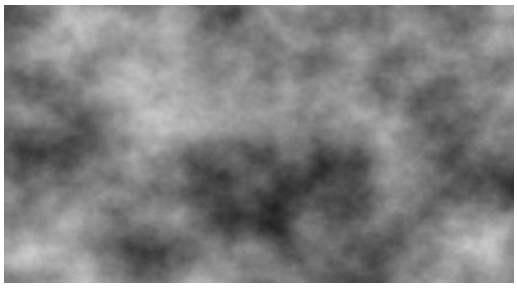
Joonis 2.1 Kõrguskaartide [20]

See tehnika sobib hästi 3D-maastikupiltide loomiseks, kus numbrid või kõrgused on juhuslikud täisarvud antud vahemikust. Negatiivsed arvud tähistavad madalikke ja positiivsed numbrid kürgustikke. Tavaliselt salvestatakse kõrguskaart pildifailidesse (*.jpg), sest pilte esitatakse pikslite maatriksina ja halltoonides värvid kodeeridakse naturaalarvuga vahemikus 0 – 255. Kõrguskaartide salvestamine pildifailis võimaldab igal ajal muudatusi teha. Pilte hoitakse ühevärvilisena, kuid võib kasutada ka muid värve. Kõrguskaarte kasutatakse laialdaselt 3D-objektide tekstuurimisel modelleerimisprogrammides. Näiteks Blender 3d modelleerimisprogrammis sügavuse visualiseerimiseks on vaja kõrguskaardid: kühmukaart (*bump map*) või normaalkaart (*normal map*), kus 1 – hästi krobeline (valge värv) ja 0 – ei ole (must värv), halltoonidele vastavad vahemikus 0…1 reaalarvud.

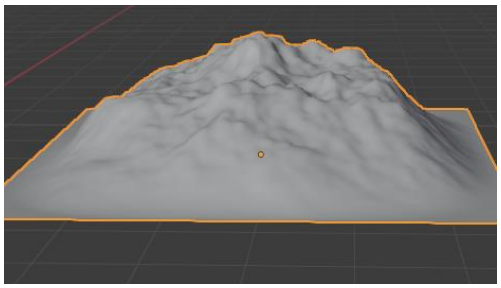
Joonis 2.2 Kühmukaart ja selle rakendamine tekstuuril Blender'i programmis

2.2 Kaartide loomine mürafunktsioonidest

Müra on arvutigraafikas juhuslike arvude generaator [21]. Mürakaardi abil saab luua 3D- ja 2D-kaarte ning tekstuure. Mürakaartide abil genereerimiseks kasutatakse sageli Perlini müra. Perlini müra on mitmemõõtmeline algoritm, mida kasutatakse protseduurilisel genereerimisel, tekstuurides, maastiku genereerimisel, kaartide genereerimisel, pinna genereerimisel ja tippude genereerimisel [21]. Näiteks on pildil mürakaart, mida saab kasutada 3D-maastiku loomiseks. Heledates piirkondades on maastik kõrgem, tumedates on madalam. Numbrivahemik on sel juhul vahemikus 0 kuni 1.

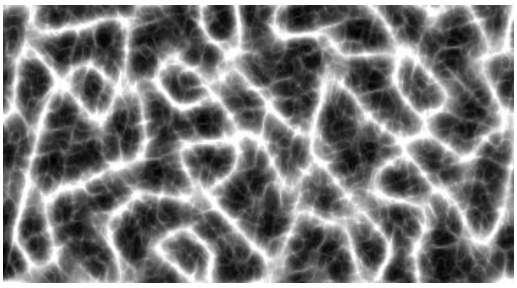


Joonis 2.5 Mürakaart [21]



Joonis 2.6 Mürakaardi järgi modelleeritud maastiku näide

See mürakaart sobib koobaste võrgu loomiseks. Koobaste kaartide jaoks sobivad paremini hargnevate käikudega joonised (Joonis 2.7).



Joonis 2.7 Mürakaart [21]

2.3 Plaaditud maastiku genereerimine

Plaaditud maastiku genereerimine on kogu ala jagamine eraldi osadeks. Kaardi koostamise protsess teostatakse aladega või alade rühmadega manipuleerimisega, määrates neile värvi või numbri. Seda tüüpi genereerimine erineb ülalkirjeldatust selle poolest, et maastiku variandid on arendajad eelnevalt loonud.

Maastiku mitme tüübiga töötades võib erinevate variatsioonide arv ületada 300 või enamat plaati. Sellise generaatori töö demonstreerimiseks sobib mäng *„Civilization"* [22], kus iga alustatud mäng genereerib uue kaardi heksagoonide abil.

Joonis 2.8 Genereeritud kaart heksagoonide abil [22]

3 PSEUDOJUHUSLIKE ARVUDE GENERAATOR

Pseudojuhuslike arvude generaator on algoritm, mis genereerib arvude jada, mille elemendid on üksteisest peaaegu sõltumatud ja alluvad etteantud jaotusele [23]. Determineeritud algoritm on algoritm, mis tagastab samade väljundväärtuste samade sisendväärtuste korral[24]. Pseudojuhuslik arv on determineeritud algoritmi abil saadud arv, mida kasutatakse juhusliku arvuna[24].

Kasutatakse juhul, kui peab looma sisu eelnevalt loodud elementide põhjal. Selle genereerimise näidet võib näha mängus „*The Binding of Isaac*". Arendajad on loonud algselt mängutoad ja kogu vajaliku sisu ning mäng paigaldas need iseseisvalt juhuslikus järjekorras. Tänu sellisele genereerimisele saab mängija iga mängu korduva läbimise ajal uue mängukogemuse, kuid samal ajal säilitavad arendajad kõrge kontrolli avaldatava sisu lõpliku kvaliteedi üle. Autor valis antud generaatori oma töös ja selles peatükis on uuritud generaatori algoritmi lähemalt.

Pseudojuhuslike arvugeneraatori realiseerimiseks Unity'is kasutati klassi *Random.Range*. *Range* kasutatakse seetõttu, et meil on eelnevalt genereeritud tubade valik. *Random* – see staatiline klass pakub mõningaid lihtsaid mängulisi viise pseudojuhuslike arvude genereerimiseks [25]. Klassis kasutatakse algoritmi *Xorshift*128 (128-biti). *Xorshift* on pseudojuhuslike arvugeneraatorite klass, mille avastas Ameerika matemaatik ja arvutiteadlane George Marsaglia. Joonisel nr 3.1 on toodud *Xorshifti* 32-bitise versiooni rakendamine. Genereeritakse juhuslikud arvud, kasutades üht algarvu, 3 nihet ja XOR loogikaoperaator [26].

```
long seed = System.nanoTime();

public long randomLong()
{
    seed ^= (seed << 21);
    seed ^= (seed >>> 35);
    seed ^= (seed << 4);
    return seed;
}
```

Joonis 3.1 xorshift algoritmi koodi näidis Java's

Arvud arvutis on digitaalsel kujul ja koosnevad bitijadadest. Teades, et 1 bitt on ühekohaline binaarne arv, mis märgistakse 0 või 1 abil ja 2-bitt arvud on kahekohalised binaarsed numbrid (00, 01, 10, 11), mis annavad kokku $2^2 = 4$ võimalikku olekut. 32-bitises versioonis kasutatakse 32-bitine arv, mis koosneb 32 kohtadest ja annab kokku 2^{32} võimalikku olekut.

Juhuslike arvude generaator xorshift (xorshift RNG) toodab 2^{32}-1 täisarvu x või 2^{64}-1 paaride x, y järjestuse või 2^{96}–1 kolmikute x, y, z või üldjuhul 2^{32n} – 1 n numbrirühma jada kasutades tehet VÕI (xor) enda nihutatud versiooniga.[27]

C++ (samuti C#, Java, PHP, JavaScript) programmeerimiskeeles tähistatakse XOR-toimingut sümboliga ^. Põhitoiming on vasakute nihete korral y ^ = (y << a), parempoolsete nihete korral y ^ (y >> a). Selliste xorshift toimingute kombineerimine erinevate nihete ja argumentide jaoks tagab ülikiire ja lihtsa juhuslike arvude generaatori [27].

Edasi on toodud antud koodi seletus ja arvutuskäik lihtsamatel arvudel.

Välistavaks VÕI (XOR) nimetatakse loogikaoperatsioonit, mis omandab väärtuse „tõene" ainult siis, kui ainult üks argumentidest on tõene ja tähistatakse $A \oplus B$ või $A\ XOR\ B$. Allpool on XOR operatsiooni tõetabel (Tabel 3.1) ja näide, kuidas tehe töötab täisarvudel.

Tabel 3.1 XOR operatsiooni tõetabel

A	B	AND	OR	**XOR**
0	0	0	0	**0**
0	1	0	1	**1**
1	0	0	1	**1**
1	1	1	1	**0**

Oletame, et x =10 ja y= 8, siis tehing $x = x\ XOR\ y$ annab tulemuses x =2. Numbrid teisendatakse binaarkoodiks ja pärast loogilise tehingu sooritamist tagastatakse kümnendarvuline tulemus.

Arvu esitamise binaarkoodiks (numbritest 1 ja 0 koosnev järjend) algoritm on järgmine

1. numbri on vaja pidevalt jagama 2-ga, kuni viimane jääk või tulemus jõuab nullini;
2. kirjutada saadud tulemused vastupidises suunas.

Arvu 10:2= 5 (jääk on 0); 5:2= 2 (jääk on 1); 2:2=1 (jääk on 0); 1:2=0 (jääk on 1).

Arvu 8:2 = 4 (jääk on 0); 4:2=2 (jääk on 0); 2:2=1 (jääk on 0); 1:2=0 (jääk on 1).

Kirjutades saadud jäägid vastupidises suunas saame 1010 – arvu 10 ja 1000 – arvu 8 binaarses süsteemis. Välistava VÕI (XOR) tehe tulemus on esitatud tabelis 3.2.

Tabel 3.2 Tehe 10 XOR 8 tõetabel

$x = 10$	1	0	1	0
$y = 8$	1	0	0	0
$x \oplus y$	0	0	1	0

Binaarsest süsteemist arvu kümnendsüsteemi teisenduse valemi (1) järgi [28]

$$10 \dots 01_{\smile n\ numbrit\ n-bitt} = 1 \cdot 2^{n-1} + 0 \cdot 2^{n-2} + \cdots + 0 \cdot 2^1 + 1 \cdot 2^0 \qquad (1)$$

tagastame kümnendarvulise tulemuse: $0010_{bin} = 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 2_{dec}$.

Nihketehete puhul nihutatakse registris oleva arvu kõiki järke korraga [29,lk 32] ja väljanihkuvad bitid kaovad ja tühjad kohad täidetakse 0- ega. Nihe 1 võrra vasakule suurendab kümnendsüsteemis arvu kaks korda, nihe paremale vähendab kaks korda[30] ja üldjuhul arvutatakse järgmise valemi järgi

$$\begin{aligned} x \ll y &\Leftrightarrow x \cdot 2^y \\ x \gg y &\Leftrightarrow \frac{x}{2^y} \end{aligned} \qquad (2)$$

Näiteks, tehe $x\hat{} = (x \ll 2)$ tulemus on 34, kui $x = 10$. Tehete järjekord on 1) nihketehe; 2) XOR.

Kui lahendada kümnendsüsteemis, siis 1) $10 \ll 2 = 10 \cdot 2^2 = 40$; 2) arv 40 binaarsüsteemis on juba 8-bitt arv: 00101000. Kuna 00101000 on 8-bitt arv, siis kirjutatakse ka arv 10 8-bitt arvuna $10_{dec} = 1010 = 00001010$, kui lisada enne neli nulli. Välistava VÕI operatsiooni tulemus on $00100010_{bin} = 2^5 + 2^1 = 34_{dec}$.

x	0	0	0	0	1	0	1	0
$x \ll 2$	0	0	1	0	1	0	0	0
$x\hat{} = (x \ll 2)$	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**
n	7	6	5	4	3	2	1	0

Arv, mis kasutatakse arvutiprogrammeeritav genereerimise algoritmis ei ole nii väike ja tavaliselt võrdub süsteemiajaga. Näides 3.1. on kasutatud *Java* meetod nanoTime (), mis tagastab süsteemitaimeri praeguse väärtuse nanosekundites [30]. Selle töö kirjutamise ajal tagastas arvu

$$1364888288603109_{dec} = 0100110110010101101111010001100110011010101111100101_{bn}$$

Unity dokumentatsioonist ei ole selge, mis arv võetakse arvuti operatsioonsüsteemist [25], kuid generaatori algoritmi nimetus ja tööprintsiibi kirjeldus on esitatud.

4 UNITY

Unity on populaarne, kasutajasõbralik ja tasuta, mistõttu on see mugav väikestele arendajate meeskondadele. Unity on kasutajasõbralik, sest ei kasutajale ei ole vaja teada kõrgtasemel programmeerimist, mängu objektid, meetodid on hea dokumenteeritud ja on olemas valmis ressursid (*Unity Assets*) mängu loomiseks. Kuna rühma eesmärgiks on oma mängu loomine, siis me ei kasutanud valmis ressuursse.

4.1 Kaardigeneraatori töö

Kaardi juhuslik koostamine toimub plokkide paigutamise teel üksteisele nii, et nende vahele tekib läbipääs ja seinad ei takista ühest ruumist teise jõudmist.

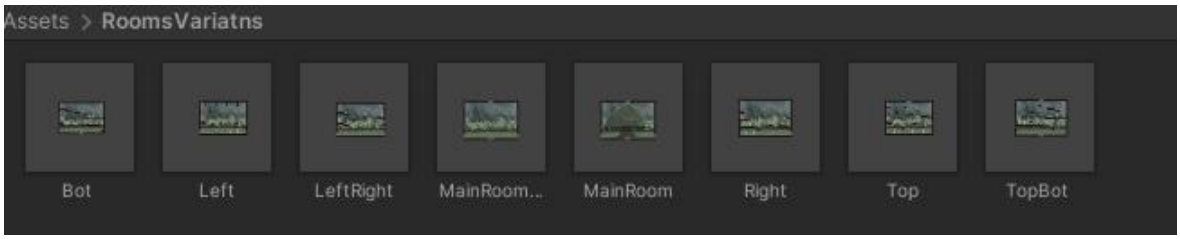
Kaardi vale ja õige ehituse näited on joonisel 4.1. Kaardi vale näide, kus seinad blokeerivad läbipääsu ühest toast teise ja õige ehituse näide, kus seinad ei sega ühest toast teise üleminekut.

Joonis 4.1 Blokeeritud läbipääs ja läbipääs olemas

Generaator ei tohiks jätkata kaardi loomist pärast tuba, millest pole üleminekuid teistesse stseenidesse.

Kaardi koostamisel pannakse sobivad plokid juurde ja generaatori plokkide arvu saab igal ajal muuta, luues uue toa ja lisades selle vastavasse loendisse. Kui lisada loendisse ühe toa mitu korda, suurendab see kaardistruktuuris selle ilmumise sagedust.

Mugavuse huvides on kõik toad salvestatud eraldi kausta.



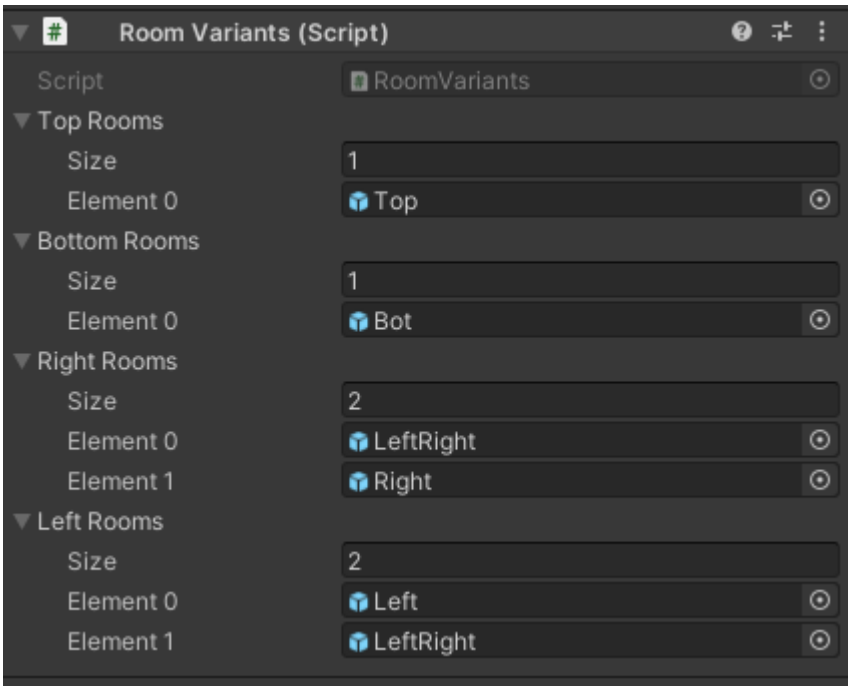
Joonis 4.3 Võimalikud toad

Tubade olemasolu kaustas ei tähenda, et neid kaardi loomisel kasutatakse, kuna selleks lisati koodi abil eraldi funktsioon tühjale esemele Unity-s.

```
public class RoomVariants : MonoBehaviour
{
    public GameObject[] topRooms;
    public GameObject[] bottomRooms;
    public GameObject[] rightRooms;
    public GameObject[] leftRooms;

}
```

Joonis 4.4 Kood



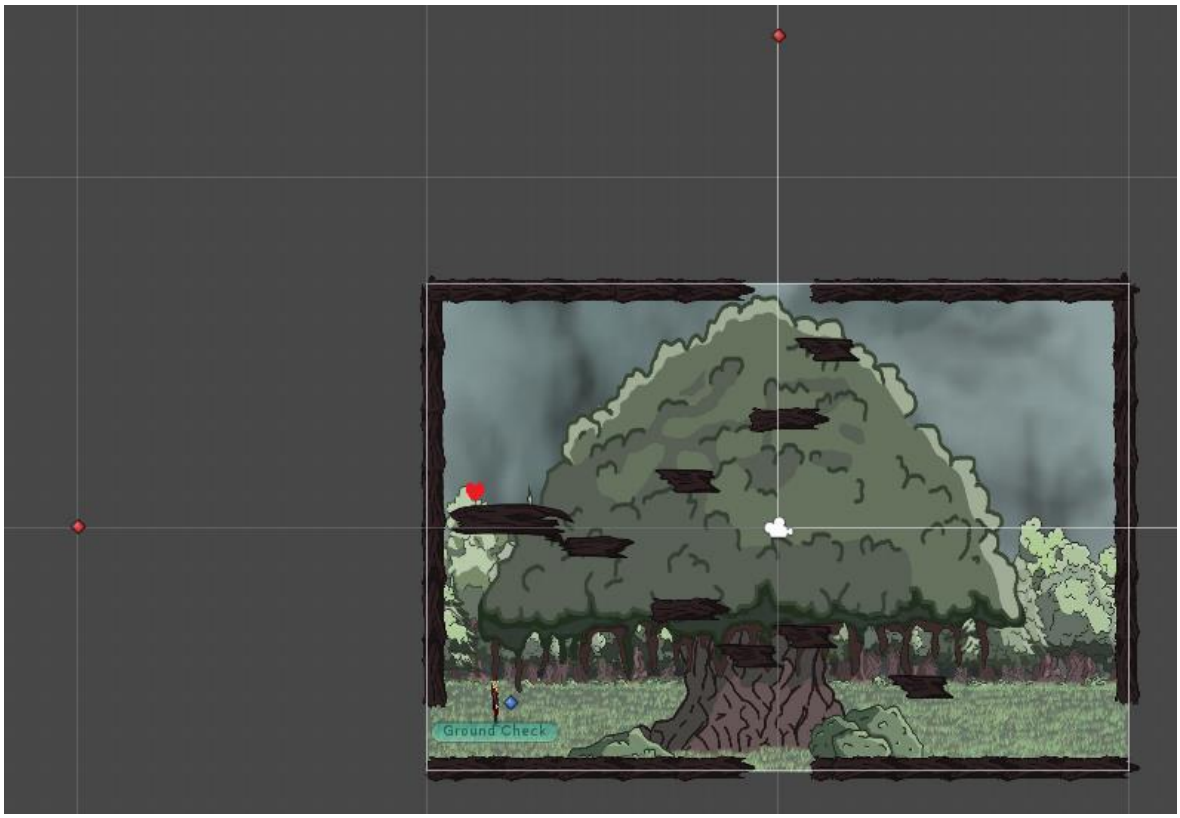
Joonis 4.5 Tubade loetelu

Sõltuvalt läbikäigu asukohast toas lisatakse see eraldi loendisse, millest generaator võtab sobiva toa ja lisab selle kaardile. Näiteks kui panete ruumi loendisse *„Left*

Rooms", lisatakse see tuba käigule, mis viib vasakule, luues seeläbi käigu lisatud tuppa.

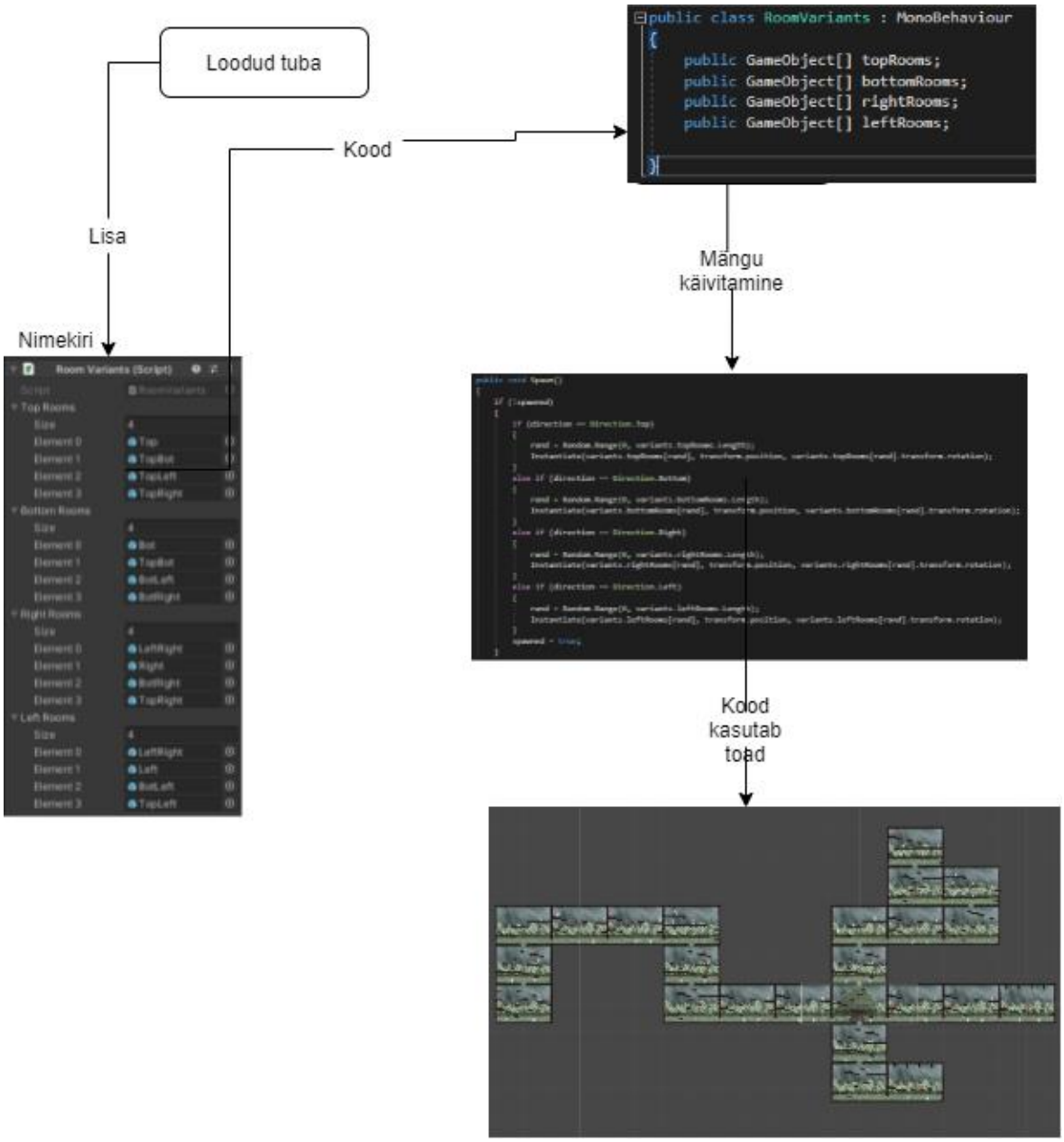
Selles variandis on tubade arv minimaalne, et näidata, kuidas väikese tubade arvuga kaart võib muutuda. Teoreetiliselt ei ole kaardile lisatud võimalike tubade arv piiratud. Kuid mida rohkem tube pakutakse generaatorile valida, seda suuremaks osutub kaart, mis pikendab genereerimise aega. Praktikas võib liiga paljude tubade arvuga kaardi genereerimine võtta kaua aega.

Genereerimise alguses on meil üks tuba (Joonis 4.6), millest ehitatakse teeradasid teiste ettevalmistatud stseenide abil. Tubade ümber on sihtpunktid (joonisel on punased rombid), kuhu on seatud, millisel küljel asub juba genereeritud tuba. Nende sihtpunktide abil ühendatakse toad õigesti.



Joonis 4.6 Esimene tuba

Kaardi loomise etapid kajastab skeemi 4.7.



Joonis 4.7 Üldine skeem

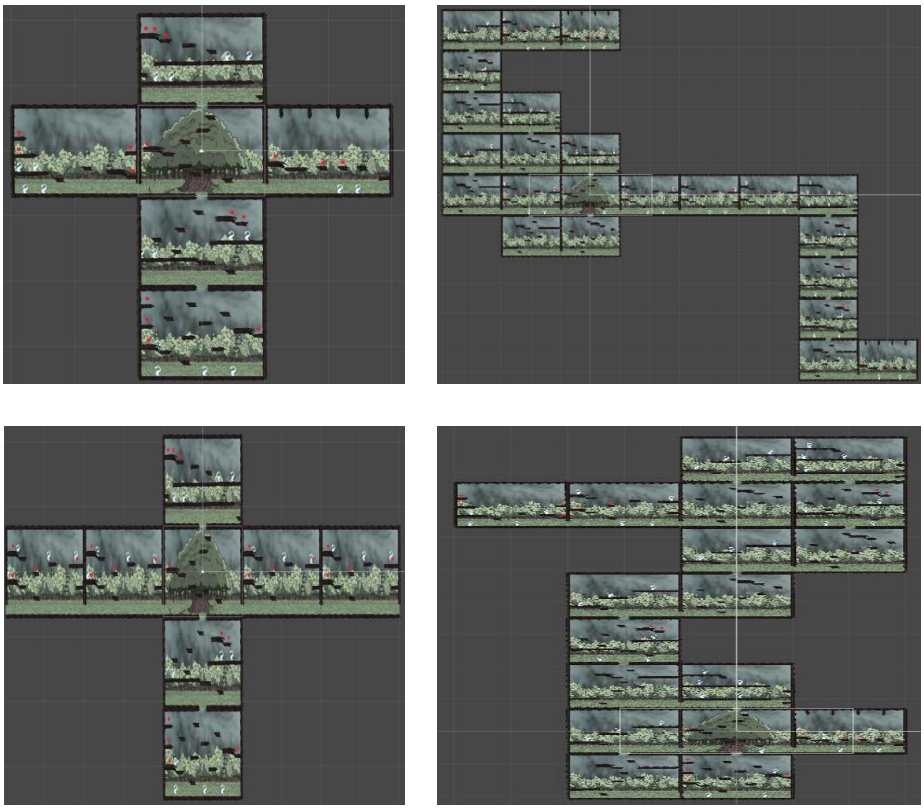
Joonisel 4.8 on esitatud kaardigeneraatori kood. Generaator töötab funktsiooni If kaudu. Esiteks kontrollitakse toa olemasolu sihtpunkti peal. Igal sihtpunktil on märgitud, kummal pool seda tuba asub. Kui tuba puudub, siis käivitatakse kontroll, kuspool sellest sihtpunktist juba loodud tuba asub. Pärast toa asukoha määramist alustatakse tubade nimekirjast juhusliku toa valimist. Kui kood valib juhusliku toa, paigutab ta kaardile ja kood kordub uuesti. Kaardi loomits lõpetatakse kui kõik kaardi läbipääsud on suletud.

```
public void Spawn()
{
    if (!spawned)
    {
        if (direction == Direction.Top)
        {
            rand = Random.Range(0, variants.topRooms.Length);
            Instantiate(variants.topRooms[rand], transform.position, variants.topRooms[rand].transform.rotation);
        }else if (direction == Direction.Bottom)
        {
            rand = Random.Range(0, variants.bottomRooms.Length);
            Instantiate(variants.bottomRooms[rand], transform.position, variants.bottomRooms[rand].transform.rotation);
        }
        else if (direction == Direction.Right)
        {
            rand = Random.Range(0, variants.rightRooms.Length);
            Instantiate(variants.rightRooms[rand], transform.position, variants.rightRooms[rand].transform.rotation);
        }
        else if (direction == Direction.Left)
        {
            rand = Random.Range(0, variants.leftRooms.Length);
            Instantiate(variants.leftRooms[rand], transform.position, variants.leftRooms[rand].transform.rotation);
        }
        spawned = true;
    }

}
```

Joonis 4.8 Generaatori kood

Joonisel 4.9 on esitatud genereerimise abil kaardi loomise mõned näited.

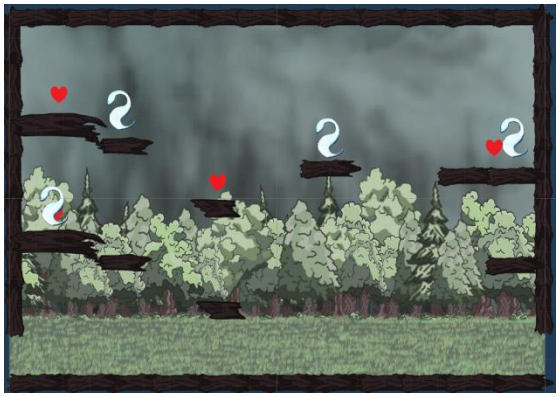


Joonis 4.9 Genereeritud mängu kaardid

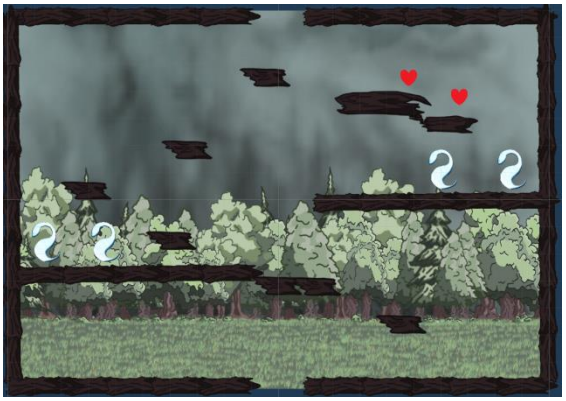
4.2 Generaatori tööks ettevalmistamine

Kuna mängus kasutatakse pseudojuhuslike arvude generaatoril põhinevat generaatorit, kaart koosneb eelnevalt loodud tubadest.

Eelnevalt loodud ruumidesse saab lisada erinevaid mänguelemente, nagu vaenlased, platvormid, esemed, tegelased, stseenid (Joonis 5.1 ja Joonis 5.2). Seda kõike tehakse tubadesse vahelduse lisamiseks.



Joonis 5.1 Näide 1



Joonis 5.2 Näide 2

Esialgu täielikult läbi töötatud tuba, mis juhuslikult lisatakse kaardile, annab arendajale kõrge kontrolli saadud toote üle.

Samuti peab enne loodud toa lisamist kaardigeneraatorisse lisama sellele mänguelemendid. Elemendid sõltuvalt mängu žanrist. Loodavas mängus on vaja peategelase liikumiseks tubadesse lisada platvormid, lahinguvaenlased ja tervist täiendavad esemed. Võib lisada ka dekoratiivseid esemeid, mis mitmekesistavad tube.

KOKKUVÕTE

Tänaseks on arvutimängutööstus kogu maailmas oluliselt laienenud. Nii arvutimänge tootvate ettevõtete kui ka mängijate arv muutub järjest suuremaks, mis on suurendanud huvi selle lõputöö ja videomängude loomise vastu.

Lõputöös kaaluti maastiku juhusliku genereerimise võimalust mitmesuguste tänapäeval kasutatavate populaarsete meetodite abil.

Töö eesmärk oli genereerida mängukaart, mis tehti töö käigus. Töö alguses oli probleem mitme toa kihistamine algustoale. Hiljem leiti lahendus, asetades algustoa keskele täpi, mis hävitab iga selles kohas ilmnenud toa.

Eesmärgi saavutamiseks uuriti kaartide genereerimise meetodeid, mis põhinevad protseduurilisel genereerimisel ja pseudojuhuslike arvugeneraatori kasutamisel. Nende meetodite uurimisel otsustati kasutada pseudojuhuslikku arvugeneraatorit selle valitud žanri mängu loomise mugavuse tõttu. Arendamiseks kasutati *Unity* mängumootorit. Samuti lisati mängule žanri eripära tõttu lahingusüsteem.

Lõputöö tulemusena on Unity mängumootoril loodud mängu „*Soul never Die*" jaoks 2D-kaardid.

Töö ajal oli märgitud seda, et kõige palju aega võtab kunstniku töö, et joonistada kõik detailid. Kaardi genereerimine võtab see probleem ära, sest selle abiga kunstniku ei ole vaja seda kõik joonistada. See aitab salvestada ressursid mängu rakendamise ajal, mis on väga kasulik kui sellega töötab väike rühm.

Mängu kallal töötamise käigus saadi palju teadmisi, mis võimaldavad tulevikus oma oskusi parendada. Loodud mäng on täielikult funktsionaalne. Tulevikus on selle valdkonna kogemuste kasvamisel võimalik mängu lisada uusi funktsioone ja ideid. Näiteks võib lisada esemeid, mille omadusi saab ka juhuslikult genereerida, ning ülesandeid, milles genereerimisega saab kirjeldusi ja auhinda muuta.

SUMMARY

2D game map generating on unity platform on the example of the game"soul never die"

Dmitri Vorobev

The topic of this work is "2D game map generaton on Unity Platform based on the example of the game" Soul never Die"", describing the conducted research and analysis on this topic and resources with which the work was done by the author.

The relevance of this topic lies in the fact that today the popularity of video games is actively increasing, the reason for that being that people all around the globe are forced to stay at home and explore new activities for themselves because of COVID-19 pandemic lockdown.

The purpose of this work was the desire to try to create a 2D project in the roguelike genre using a pseudo-random number generator, which formed the basis for creating a map generation. A group of people with shared responsibilities worked on this project, but the author of this topic, "2D game map generation on Unity Platform based on the example of the game "Soul never Die"", was specifically researching the topic of creating a passable generated map for the project.

The game named "Hollow Knight" was used as an example for the project basics.

To achieve this goal, the author of this title studied the principle of map generation operation, their types, and the most popular and common models. An also analysis was carried out, which included the study of strengths and weaknesses of this method, and a study of the basics of working with Unity, as well as the main characteristics of the "roguelike" genre.

While studying this topic, a method for generating a map was chosen through the use of pseudo-random numbers.

As a conclusion for the study of this topic, there were generated maps for the game named "Soul Never Die" on the basis of the materials studied and the skills acquired in a practical way.

KASUTATUD KIRJANDUSE LOETELU

1. The Year in Numbers 2020 [Online]
   https://www.gamesindustry.biz/articles/2020-12-21-gamesindustry-biz-presents-the-year-in-numbers-2020 (03. 02. 2021)
2. Ametlik veebileht, "Minecraft" [Online]
   https://www.minecraft.net/ru-ru(03. 02. 2021)
3. Ametlik veebileht, "Supergiant" [Online]
   https://www.supergiantgames.com/(03. 02. 2021)
4. Poe ametlik veebileht, "Steam" [Online]
   https://store.steampowered.com/app/367520/Hollow_Knight/(03. 02. 2021)
5. Ametlik veebileht, "Unity" [Online]
   https://unity.com/(03. 02. 2021)
6. Как процедурная генерация помогает создавать открытые миры" [Online]
   https://dtf.ru/gamedev/169117-kak-procedurnaya-generaciya-pomogaet-sozdavat-otkrytye-miry(05. 02. 2021)
7. Ametlik veebileht, "Dungeons & Dragons" [Online]
   https://dnd.wizards.com/ (05. 02. 2021)
8. Arvutimängude kataloog, "Beneath Apple Manor" [Online]
   https://www.mobygames.com/game/beneath-apple-manor(05. 02. 2021)
9. Процедурная генерация, её история, проблемы и невероятные креативные возможности [Online]
   https://kanobu.ru/pub/436865/(05. 02. 2021)
10. Poe ametlik veebileht, "Steam" (Online)
    https://store.steampowered.com/app/241600/Rogue_Legacy/(05. 02. 2021)
11. Ametlik veebileht, "Diablo 3" [Online]
    https://eu.diablo3.com/ru-ru/(05. 02. 2021)
12. Ametlik veebileht, "FTL" [Online]
    https://subsetgames.com/ftl.html(06. 02. 2021)
13. Ametlik veebileht, "Binding of isaac" [Online]
    https://bindingofisaac.com/(06. 02. 2021)
14. Ametlik veebileht, "Hades" [Online]
    https://www.supergiantgames.com/games/hades/(06. 02. 2021)
15. Ametlik veebileht, "No man's sky" [Online]
    https://www.nomanssky.com/(06. 02. 2021)
16. Ametlik veebileht, "65daysofstatic" [Online]
    https://65daysofstatic.com/(06. 02. 2021)
17. Ametlik veebileht, "Star wars" [Online]
    https://www.starwars.com/series/the-mandalorian(07. 02. 2021)

18. Ametlik veebileht, "UnrealEngine" [Online]

https://www.unrealengine.com/en-US/(07. 02. 2021)

19. Процедурная генерация — поразительное будущее игровой индустрии [Online]

http://texno.info/etc/procedurnaja-generacija-porazitelnoe-budushhee-igrovoj-industrii/(07. 02. 2021)

20. Генерация трехмерных ландшафтов [Online]

https://www.ixbt.com/video/3dterrains-generation.shtml(14. 02. 2021]

21. Нечёткий мир шума Перлина [Online]

https://habr.com/ru/post/441312/(16. 02. 2021)

22. Ametlik veebileht, "Civilization" [Online]

https://civilization.com/(16. 02. 2021)

23. Генераторы случайных и псевдослучайных чисел [Online]

https://moluch.ru/conf/tech/archive/286/13233/(02. 03. 2021)

24. Генератор псевдослучайных чисел [Online]

https://www.sgu.ru/sites/default/files/textdocsfiles/2018/07/09/slepovichev_i.i._generatory_psevdosluchaynyh_chisel_2017.pdf(02. 03. 2021)

25. Random, Unity documentation [Online]

https://docs.unity3d.com/ScriptReference/Random.html(04. 03. 2021)

26. XORShift random number generators [Online]

https://www.javamex.com/tutorials/random_numbers/xorshift.shtml(26. 05. 2021)

27. Marsaglia, George, Xorshift RNGs. Journal of Statistical Software, 2003, vol 8 [E-ajakiri] https://www.jstatsoft.org/article/view/v008i14/0 (26.05.2021)

28. PLC programmis kasutatavad arvsüsteemid [Online]

http://www.tthk.ee/CoDeSys/numbrisusteemid.html(26. 05. 2021)

29. Digitaaltehnika, loengukonspekt, Tallinna Tehnikakõrgkool [Online]

https://moodle.tktk.ee/mod/resource/view.php?id=121770(26. 05. 2021)

30. Left Shift and Right Shift Operators in C/C++, GeeksforGeek, 18.02.2021 [Online]

https://www.geeksforgeeks.org/left-shift-right-shift-operators-c-cpp/ (26.05.2021)