TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Aleksander Ozerov 206175IADB

Dungeon Crawleri veebimängu serveritarkvara arendus

Bakalaureusetöö

Juhendaja: Einar Kivisalu
Magistrikraad

Tallinn 2023

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Aleksander Ozerov

11.05.2023

Annotatsioon

Selle lõputöö eesmärk oli luua serveritarkvara Dungeon Crawler veebimängule. Mängu idee on võimaldada mängijatel uurida koopaid, muutes samal ajal oma tegelased tugevamaks, samuti võimaldada mängijatel kasutada oma rühmas teisi mängijategelasi. See töö kirjeldab projekti analüüsiosa, kuidas mängu visioon kujundati, analüüsides olemasolevaid mänge Dungeon Crawleri žanris ja praktilist osa, mis selgitab, kuidas tarkvara arendati, milliseid tööriistu kasutati ja miks, samuti serveri API rakendamist.

Selle projekti eesmärk oli luua serverirakendus, mis töötleks mängu loogikat ja pakuks API lõpp-punkte tulevase kliendirakendusega suhtlemiseks. Nende eesmärkide saavutamiseks loodi tarkvara C# programmeerimiskeeles, kasutades Entity Frameworki ja ASP.NET Core raamistikke. Lõputöö peamised eesmärgid saavutati.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 5 peatükki, 18 joonist, 0 tabelit.

Abstract

Server Software Development for the Dungeon Crawler Web-Based Game

The purpose of this thesis work was to create server software for a web-based Dungeon Crawler game. The game idea is to let players explore dungeons while making their characters stronger, as well as allow players to use other player characters in their group. This work describes the analysis part of the project, how the game vision was designed, analyzing existing web games in Dungeon Crawler genre, and the practical part that explains how the software was developed, what tools were used and why, as well as the implementation of server API.

The goal of this project was to create a server application that would process the game logic and provide API endpoints for communication with future client application. To achieve these goals the software was created in C# programming language with the use of Entity Framework and ASP.NET Core frameworks. The main goals of the thesis were achieved.

The thesis is in Estonian and contains 31 pages of text, 5 chapters, 18 figures, 0 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
BLL	*Bussiness Access Layer*, äriloogika kiht
Character	Mängija tegelane
DAL	*Data Access Layer*, andmekiht
DnD	*Dungeons & Dragons*, laua-rollimäng
Domain	Domeen
DTO	*Data Transfer Object*, andmeedastusobjekt
Dungeon	Koobas
Effect	Efektid, mis muudavad tegelaste statistikat
HP	*Health Points*, Tervisepunktid
HTTP	*HyperText Transfer Protocol*, hüperteksti edastusprotokoll
Item	Mängu ajal leitavad esemed
JSON	*JavaScript Object Notation*, andmevorming/süntaks andmete salvestamiseks ning vahetamiseks
JWT	*JSON Web Token*
MP	*Mana Points*, Mana punktid
RPG	*Role-playing game*, rollimäng
Skill	Oskused, mida tegelased saavad efektide saamiseks kasutada

Sisukord

Jooniste loetelu

1 Sissejuhatus

Tänapäeval on Dungeon Crawleri žanris väga vähe online-veebimänge. See on väga märgatav, kuna on raske leida veebimängu, kus saaksite teiste mängijatega suhelda lisaks oma edusammude jagamisele.

Seda silmas pidades sündis idee kaasaegsest Dungeon Crawler RPG veebimängust. Selle projekti eesmärk on luua paljudele mängijatele uus veebimängukogemus, kirjeldades samal ajal arendusprotsessi. Selle projekti eesmärk on alustada täiesti uue veebimängu arendamist Dungeon Crawleri žanris, mis võimaldaks mängijatel saada huvitava veebikogemuse.

Selles töös kirjeldatakse paljusid veebimängude arendamise aspekte, näiteks mänguprotsessi hoolikat analüüsi. Analüüsitakse mängu põhifunktsioone ning arengu piire.

Mänguloogika koodi kirjutamise praktiline osa on olemas, et anda ülevaade veebimängude arendamisest ja selle käigus tekkida võivatest probleemidest.

2 Projekti eesmärk

Selle projekti eesmärk on luua serverirakendus dungeon crawler veebimängu jaoks ja saada kogemusi selle kohta, mis tunne on mängu arendada. Serverirakendus tuleb kujundada hooldatavust ja uute funktsioonide lihtsat arendamist silmas pidades. Idee on luua server, mis tegeleks mänguloogika ja interaktsioonidega, samuti seadistada API lõpp-punktid, mida saaks hiljem kasutada veebimängu kliendirakenduse loomiseks.

Mängus eeldatakse, et mängijatel on vabadus oma iseloomu vastavalt oma vajadustele arendada ja neil pole kunstlikke piiranguid nagu teistel selle žanri mängudel. Tavaliselt piiravad RPG-mängud mängijaid kindlate klassidega, mis täidavad mängus teatud rolli. Selles mänguprojektis on mõte lasta mängijatel oma tegelasroll ise määratleda, kasutades õppimisoskusi, suurendades statistikat tasemetõusuga ja hankides tugevdavaid esemeid.

2.1 Metoodika

Selle projektiga parimate tulemuste saavutamiseks on vaja üksikasjalikku metoodikat. Lähenemine toimus järgmistes etappides:

Esiteks on juba olemasolevate Dungeon Crawler RPG-mängude analüüs, alustades kõige varasematest mängudest, millele järgnevad kaasaegsed tõlgendused ja mängud, mida peetakse žanris headeks. Neid analüüsides oleks lihtsam määratleda vajalikud funktsioonid ja mõelda, mis mängu jaoks kõige paremini sobiks. Mängutööstus jõudis kaugele ja nüüd on palju erinevaid mänge erinevates žanrites. See võimaldab analüüsida suurt valikut mänge.

Pärast olemasolevate mängude analüüsi tuleb määratleda nõutavad ja toredad funktsioonid. Need on funktsioonid, mis määratlevad, kuidas mäng töötab ja mida mängijad võiksid seda mängides kogeda.

Järgmine samm oleks analüüsida, milliseid tehnoloogiaid parema tulemuse saavutamiseks kasutada.

Pärast kogu analüüsi tegemist tuleks alustada serveritarkvara arendamist. Tarkvara on välja töötatud kõiki eelnevaid analüüse silmas pidades, et saavutada hea tulemus.

Kui tarkvara loetakse tööversiooni olekusse sisenenuks, tuleks teha ulatuslik testimine, et teha kindlaks, kas kõik tarkvara funktsioonid töötavad nii, nagu on ette nähtud, et tagada kogemuste edukas edastamine mängijatele.

3 Analüüs

Kõigepealt tuleks selgeks teha, mis on *Dungeon Crawleri* žanr. Selle keskmes on *Dungeon Crawleri* mäng RPG, mis on tavaliselt fantaasiakeskkonnas, kus mängijad võtavad seiklejate rolli ja uurivad labürindikeskkondi, võitlevad koletiste või muude vaenlastega, leiavad aardeid, väldivad püüniseid ja lahendavad mõistatusi. *Dungeon Crawleril* on üldiselt konkreetne eesmärk, mille nimel mängijad tülitsevad: võita koopa lõpus kaabakas, päästa printsess koopast või lihtsalt otsida sügavale koopasse peidetud aardeid. [1]

Žanr sai algselt alguse lauamängudest 1975. aastal. Üks esimesi Dungeon *Crawleri* lauamänge kandis nime *Dungeon!* ja oli paljuski sarnane *DnD-ga*. Mängijad leiaksid end koopaid uurimas ja koletistega võitlemas ning mida kaugemale nad lähevad, seda ohtlikumaid koletisi ja väärtuslikumaid aardeid. [1]

Peaaegu kõik mängud, olgu need siis laua- või videomängud, järgivad samu ühiseid reegleid: seal on tegelane või tegelaste rühm, kes uurivad koopasse, võitlevad koletistega, kes blokeerivad nende tee ja kellel on lõppeesmärk, mida nad üritavad saavutada. See on väga oluline osa, mängijad ei investeeriks mängu, kui neil poleks selget eesmärki, mille nimel töötada.

3.1 Olemasolevate mängude analüüs

Dungeon Crawleri mängude erinevatest olemasolevatest rakendustest parema ülevaate saamiseks analüüsitakse nii arvuti- kui ka veebimänge.

3.1.1 Videomängude analüüs

Oleks õiglane alustada esimesest teadaolevast videomängust *Dungeon Crawleri* žanris. Varaseimad teadaolevad *Dungeon Crawl* arvutimängud ilmusid PLATO süsteemis 1975. aastal. Mängu nimi oli *pedit5*, kuid seda tunti ka kui *Dungeoni*. See oli tugevalt inspireeritud lauamängust *Dungeons & Dragons*, mängijad lõid tegelase, kelle statistika sarnaneb *DnD* omaga (tervisepunktid, tugevus, osavus jne), ja seejärel juhatasid selle tegelase läbi koopa. Koobas ise oli selles mängus alati sama, samas kui koletiste kohtumised ja leitud aarded tekkisid tegelase loomisel. Koobas renderdati kahemõõtmelise ülalt-alla vaatena ja seda juhiti klaviatuurikäskudega. [2] Vangikongi uurides võisid mängijad kohata koletisi ja neil oleks kolm võimalust: võidelda, loitsu välja võluda või põgeneda. Kui mängijal ei õnnestu ära joosta, edeneks lahing tegelase ja koletise vahel automaatselt ilma mängija panuseta. [3]

3.1.2 Veebimängude analüüs

Tänapäeval pole palju veebimänge, mis pole suunatud lastele ja *Dungeon Crawleri* žanris pole neid peaaegu üldse. Üks veebimängudest *Dungeon Crawler* on *Dungeoneers*. Selles mängus mängite ühena kolmest kangelasest: *Human Swordsman, Dwarven Brawler* ja *Elven Huntress*. Kui olete seiklust alustanud, sisenete lahingutsooni, mis koosneb kuusnurksetest ruumidest. Kui te esimest korda tsooni sisenete, peidetakse teid kohalolevate koletiste eest, kuid iga kord, kui liigutate ühte ruumi, on võimalus, et koletised märkavad teid ja ründavad teid. [4]

Selle mängu lahingud on pöördepõhised, teie ja koletised vahelduvad toimingute tegemisega. Tegelane saab kasutada mõõka, et võidelda koletistega, kes asuvad selle ruumi kõrval, kus nad seisavad, kasutada vibu, et võidelda koletistega kuni kahe ruumi kaugusel tegelasest, kasutada kilpi kaitse suurendamiseks või kasutada maagilisi kirjarulli. Kõiki neid toiminguid tehakse täringurulliga, et teha kindlaks, kas see on edukas või mitte, koletist rünnates veeretavad koletised ka täringut, et teha kindlaks, kas nad saavad kahju.

Lahingutsoonis viibides on võimalik leida varustust, mis muudab iseloomu tugevamaks, näiteks parem mõõk, vibu või kilp. Ümberringi lebavad ka kerimised ja võlujoogid, mida saate eelise saamiseks kasutada.

Pärast kõigi koletiste alistamist või neist lahingutsoonis mööda hiilimist on valida, millist teed sihtkohta jõudmiseks valida, kuid kõik need viivad järgmisse tsooni. Iga seikluse lõppeesmärk on draakoni leidmine ja alistamine. Draakon on tugevaim koletis, kelle võitmiseks vajate häid ettevalmistusi.

Mängu võib pidada ka *Roguelike'i* žanriks, mis on *Dungeon Crawleri* mängudes väga levinud. See tähendab, et iga kord, kui seiklust alustate, lähtestatakse kogu teie stardivarustus ja peate algusest peale edasi liikuma. Sellistel mängudel on õnne element, et määrata seikluse eduka lõpuleviimise võimalused. Kui ebaõnnestute, võib järgmine kord olla parem.

3.2 Visioon

Visioon on luua *Dungeon Crawler* mäng, kus mängijad saavad võidelda koletistega, saada esemeid ja kasvada tugevamaks. Mängukogemus on peamiselt üksikmängija, kusjuures twist on see, et kõik grupi liikmed on teiste mängijate tegelased.

See tähendab, et arendades oma iseloomu, ei kujunda nad mitte ainult oma mängustiili, vaid aitavad kaasa ka teiste mängijate grupi loomisele. Kuna mängul pole ühtegi eelnevalt määratletud klassi, mida võiks leida teistest mängudest, on mängijatel vabadus määratleda oma tegelaskuju roll: iga asja peale mees, meeskonda kaitsev rüütel, meeskonda tugevdav võlur jne. See võimaldab mängijatel katsetada erinevaid grupidünaamikaid ja leida, mis sobiks kõige paremini erinevate koobaste jaoks.

Koopad, mida mängijad peavad täitma, oleksid eelnevalt määratletud, kuid iga kord erineva tsoonide paigutusega. Mängijad peavad planeerima ja valmistuma vangikongideks, kuna koobastest leitud esemeid ei saa kasutada ja need tuuakse kätte alles siis, kui koobas on täielikult valmis. Kui mängijad lahkuvad koopast või ebaõnnestuvad, saavad nad kogemuspunkte ainult lüüa saanud koletistelt.

Mängijad võitlevad koletistega pöördepõhises lahingusüsteemis. Mängijad saavad valida grupi iga tegelase jaoks toimingud. Iga tegelane võib valida, kas rünnata, et koletisele kahju tekitada ja kaitsta, et suurendada oma kaitsestatistikat. Lisaks nendele tegevustele saavad tegelased valida, kas kasutada õpitud oskusi. Oskused ulatuvad loitsude ründamisest kuni tugevdavate efektideni.

Mängijad arendavad oma tegelasi, tasandades end pärast seda, kui on koletistega võideldes saanud piisavalt kogemuspunkte. Iga taseme tõusuga saavad tegelased oma statistika suurenemise ja täiendavad statistikapunktid, mida mängijad saavad kasutada oma iseloomu edasiseks arendamiseks.

Pood, kuhu mängijad pääsevad koopauurimiste vahel, võimaldab neil raha eest esemeid osta või müüa. Mängijad saavad esemete müümisest või palga saamisest raha teenida ainult siis, kui teised mängijad kasutavad oma tegelaskuju vangikongi lõpuleviimiseks. Esemed, mida mängijad võivad leida, ulatuvad kasutuskõlbmatutest esemetest, mis langevad konkreetsetest koletistest raha eest müüki, kuni tarbekaupadeni (tervendavad, tugevdavad esemed) ja raamatuteni, mis õpetavad uusi oskusi.

3.3 Scope

Selle projekti ulatuse jaoks peaks serverirakendusel olema suurem osa mänguloogikast tehtud:

- Server peaks saama hakkama nii mängijakontode loomise kui ka mängija tegelaste loomisega.
- Käsitsege mänguseansi loomist. Mängijad saavad luua rühma teiste mängijate tegelastega.
- Server peaks suutma koopas võitlusloogikaga hakkama saada. Töötlege mängija sisendit ja genereerige koletiste jaoks toiminguid.
- Käsitsege tegelaste taseme tõusu. Tegelased suurendavad oma statistikat kogemustega.
- Käsitsege mängupoes suhtlemist - mängijad ostavad ja müüvad esemeid.

Tuleb luua API lõpp-punktid, et võimaldada klientrakendustel serveriga suhelda. API peaks võimaldama kontode, tegelaste ja kogu muu mängu edenemiseks vajaliku suhtluse loomist.

3.4 Projekti nõuded

Rakenduste arendamisega parema tulemuse saamiseks tuleks selgeks teha funktsionaalsed ja mittefunktsionaalsed nõuded ning üksikasjalikud kasutajalood.

3.4.1 Funktsionaalsed nõuded

- Mängijad saavad mängule konto luua.
- Mängijad saavad luua oma tegelaste.
- Mängijad saavad vangikongi uurida.
- Mängijad saavad koletistega võidelda pöördepõhises lahingusüsteemis.
- Mängijad saavad võideldes oma tegelasi koopas tasandada.
- Mängijad saavad esemeid lüüa saanud koletistelt.
- Mängijad saavad poes esemeid osta/müüa.
- Mängijad saavad sõbruneda teiste mängijatega, et kutsuda oma tegelasi rühma.

3.4.2 Mittefunktsionaalsed nõuded

- Kasutajapõhine sisu on turvaline.
- API-päringud ei võta liiga palju aega.
- Serveri API-l on nõuetekohane dokumentatsioon.

3.5 Arhitektuur

Rakendus peaks olema ehitatud nii, et see oleks hooldatav ja võtaks hõlpsasti vastu uusi muudatusi arenduses. Sel põhjusel peeti selle projekti jaoks parimaks *Domain, Data Access Layer, Business Logic Layer* lähenemist. See kihiline arhitektuur võimaldab projekti korralikult struktureerida, skaleerida ja hõlpsasti hooldada.

Domain on kiht, kus kirjeldatakse kõiki andmebaasi olemeid, nende sisu ja seoseid teiste olemitega.

Data Access Layer (DAL) on kiht, kus rakendatakse andmete salvestamise ja edastamise protsessi. DAL pakub *Domain* kihile liideseid ja abstraktsioone, et suhelda salvestusprotsessidega. DAL kapseldab andmete salvestamise keerukust ja pakub lihtsat ja järjepidevat liidest BLL-iga. Selles kihis luuakse repositooriumid.

Business Logic Layer (BLL) on koht, kus rakendatakse projekti põhiloogikat. See kiht vastutab peamise mänguloogikaga tegelemise ja API-kihi andmete esitamise eest. Selles kihis luuakse teenused.

REST API kontrollerid on viimane kiht, mis vastutab serveri ja kliendi vahelise suhtluse eest. API kiht hoolitseb andmete edastamise eest kliendile ja serverile.

4 Praktiline osa

Selle projekti tehnoloogiate valimisel võeti arvesse nii minu, töö autori, kogemusi kui ka tehnoloogia õppimise keerukust. Olemasolev dokumentatsioon oli samuti väga suur tegur. Parim valik selle projekti jaoks oleks tehnoloogiad, mis pakuksid suurt paindlikkust ja hooldatavust, samuti nõuetekohast dokumentatsiooni ja kogukonna tuge arenduse käigus tekkida võivate probleemide korral.

Selle projekti programmeerimiskeele valimisel võeti arvesse ainult neid keeli, milles autoril oli varasem kogemus, sest uute õppimine mõjutaks tugevalt selle rakenduse tegemiseks kuluvat aega. Eespool öeldut arvesse võttes analüüsiti enne töö alustamist järgmisi keeli:

- Python – on objektorienteeritud programmeerimiskeel, mis toetab peale OOP ka teisi programmeerimisparadigmasid, nagu funktsionaalne ja protseduuriline programmeerimine. Pythonit saab kasutada paljudes valdkondades, nagu veebiarendus, andmeanalüüs, masinõpe, automatiseerimine jne. tänu oma ulatuslikule saadaolevate raamatukogude nimekirjale. Üldiselt peetakse seda üldotstarbeliseks keeleks. [5] Pythonil on ka suurepärane raamatukogu veebiarenduseks nimega Django.

Autoril on Pythoniga piisavalt kogemusi, kuid autoril pole Djangoga mingit kogemust. Selle keele valimine tähendaks märkimisväärse hulga aja kulutamist uue raamistiku õppimisele ja nuputamisele, kuidas seda selle projekti jaoks kõige paremini kasutada.

- Java – on objektorienteeritud programmeerimiskeel, seda kasutatakse laialdaselt nii veebiarenduses kui ka desktop- ja mobiilirakendustes. Java-l on väga populaarne veebiarendusraamistik Spring. [6]

  Autoril on head teadmised Java-st ja isegi mõned kogemused Spring raamistikust. See võib olla selle projekti jaoks hea valik, kuid natuke aega tuleks kulutada kevade raamistikuga tutvumisele.

- JavaScript – programmeerimiskeel, mida kasutatakse laialdaselt veebisaidi frontend'is, kuid mida kasutatakse ka mõnes serveripoolses rakenduses. Node.js on hea raamistik, mida kasutatakse veebiserverite loomiseks JavaScriptis. [7]

  Autoril on JavaScriptiga väga piiratud kogemus ja node.js raamistikuga pole autoril üldse midagi. Üldiselt oleks see väga aeganõudev valik ja küsitavate tulemustega.

- C# – on objektorienteeritud programmeerimiskeel, mida kasutatakse nii veebi, desktop, mobiilirakenduste kui ka mängude ja palju muu arendamiseks. C# veebiarenduse ülipopulaarne raamvalik on ASP.NET. [8]

  C# -ga on autoril päris palju kogemusi ja teadmisi. Autoril on ka varem olnud kogemusi ASP.NET raamistiku kasutamisega. See valik oleks tõenäoliselt parim, arvestades keele ja raamistike võimalusi ning rakenduse nõudeid, samuti aega, mis kulub arengu käigus tekkida võivate probleemide lahendamisele.

Lõpuks valiti selle projekti programmeerimiskeeleks C#. Järgmine samm oli valida raamistik, millega serverirakendust arendada. Valik tuleks siin teha raamistiku asjakohasuse, selle hooldatavuse ja paindlikkuse ning olemasolevate dokumentide põhjal. Kuigi ASP.NET võib siin olla ilmne valik, tuleks siiski kaaluda mõningaid alternatiive:

- ASP.NET Core: See on Microsofti välja töötatud avatud lähtekoodiga platvormidevaheline raamistik veebirakenduste ja -teenuste loomiseks. See on

üles ehitatud .NET Core'i käitusajale ja pakub veebiarenduseks tugevat funktsioonide komplekti, sealhulgas MVC, Razor Pages ja Web API tuge. Microsoftil on veebis saadaval ulatuslik dokumentatsioon, mis aitab suuresti kaasa arengule. [9]

- NancyFX: See on kerge avatud lähtekoodiga raamistik veebirakenduste ja -teenuste loomiseks. See on üles ehitatud .NET raamistikule ja on tuntud oma lihtsa ja elegantse lähenemise poolest veebiarendusele, samuti selle toetuse poolest konventsionaalsele konfiguratsioonile. See raamistik loodi alternatiivina ASP.NET, kuid 2020. aastal see suleti ja seda enam ei säilitatud. [10]

Eespool öeldut arvesse võttes valiti selles projektis ASP.NET Core raamistik.

4.1 Andmebaas

Siin on selgitatud, kuidas andmebaas kujundati ja kuidas seda rakendati.

4.1.1 Andmebaasi projekteerimine

Andmebaasi struktuuri kujundamine oli selles projektis eriti oluline osa. Arvesse tuli võtta paljusid asju, nagu hooldatavus, paindlikkus ja jõudlus.

Andmebaasi kujundamisel olid peamised küsimused parim viis hoida andmeid mänguseansi, vangikongide ja kõige sellega seotud kohta.

Disaini järgi on ühes mänguseansis ainult üks tegelaste grupp (1–4 tegelased). Mänguseanss peaks salvestama seansi aktiivseid efekte ja oskuste jahtumist, samuti praeguseid edusamme koopas. Mänguseansil hoitakse ka teavet koopas saadud esemete kohta, mida ei saa kasutada enne, kui koopa on lõppenud. Otsustati ühendada tegelaste grupp ja mänguseansi andmed üheks olemiks nimega *GroupStatus*. Kui tegemist oli *Dungeons*iga, otsustati jagada koopad *DungeonZones*'iks, mis hoiaks teavet selle kohta, millised koletised tal oleksid, ja selle koopa välisvõti, kuhu nad kuuluvad. *Dungeon*idel endil oleks ka tasemenumber, mis deklareeriks koopas kohatud koletiste taset. *Monster* olem on lihtne. See sisaldab andmeid iga üksiku koletise kohta: selle nimi ja statistika, näiteks tervisepunktid, rünnakujõud ja nii edasi.

MonsterList on tabel, mis ühendab Dungeoni tsoonid ja koletised omavahel palju-paljudele suhetes. Sellel on ka arv, kui palju seda tüüpi koletisi on vangikongi tsoonis.

Igal koletisel peaks olema ka nimekiri potentsiaalsetest esemetest, mida see võib langeda. *DropTableItem* tabel on loodud selleks otstarbeks. See salvestab koletise ID-d ja selle eseme ID-d, mida see võib langeda, samuti võimalust ja ilmuda võivate esemete arvu.

Item olemil on nime- ja kirjeldusväljad, samuti kaubakategooria, mis on salvestatud kui Enum, *Skill* ID, kui see ese suudab tegelasele õpetada seda uut oskust, *Effect* ID, kui see ese suudab seda kas kasutades või passiivselt toota. Esemel on ka üldine hind ja minimaalne tase, et tegelane saaks seda poest osta.

Skill on olem, mis kirjeldab ainulaadseid võimeid, mida tegelased saavad kasutada koobastes seikluste ajal. See salvestab oskuste nimed ja kirjeldused, *Effect* ID, mida oskus rakendab, oskuse jahtumine pöörete arvus, enne kui seda saab uuesti kasutada, ja mitu Mana punkti see oskus aktiveerimisel tarbib.

Effect on olem, mis kirjeldab märkide statistika muutusi oskuse või eseme kasutamisel. Samamoodi on sellel nimi ja kirjeldus, aga ka kestus kordamööda (kui kaua efekt kestab) ja märkide statistika modifikaatorid (positiivsed ja negatiivsed).

Viimase kolme olemi jaoks tehti tabelid InRaidEffect, InRaidSkill ja *InRaidItem*, et luua *GroupStatus*ega palju-mitmele suhe. Tabelites on ka mänguseansile omane teave, näiteks seansil selle konkreetse eseme kohta leitud esemete arv, et vältida kirjete dubleerimist, pöörete arv enne efekti lõppu või pöörete arv enne, kui oskust saab uuesti kasutada.

Märkide andmete salvestamine oli üsna lihtne. *Character* olemil on kogu teave märgi kohta, näiteks tegelaste nimi, statistika, praegune tase ja kogemuspunktid, raha ja kasutaja ID. Otsustati mitte ühendada *Character* ja *Monster* ühte lauda, kuna see mõjutaks jõudlust, aga ka seetõttu, et neil on eraldi rollid. Koletiste ja tegelaste koos hoidmine võib arengu ajal põhjustada märkimisväärseid probleeme.

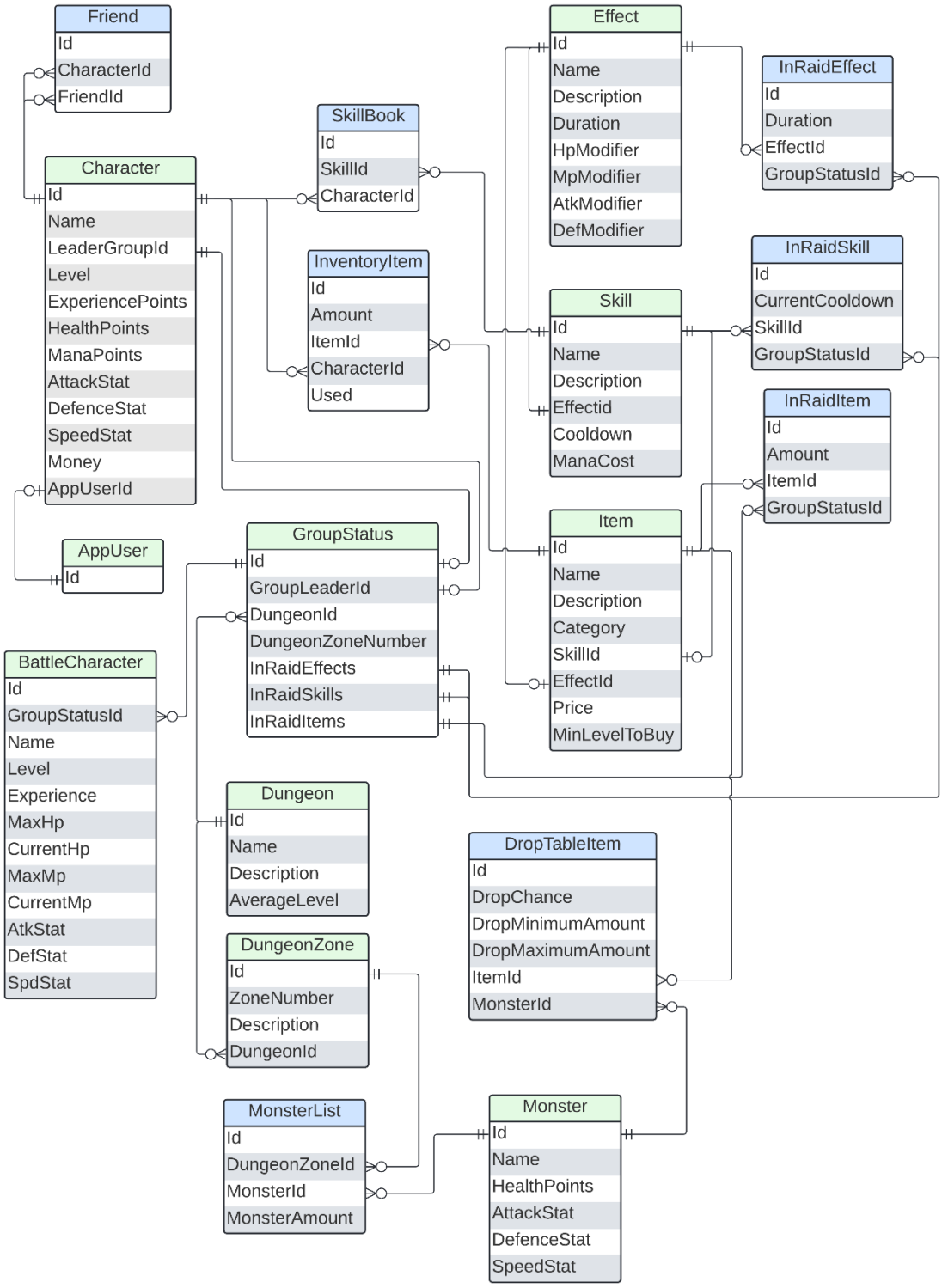
Tabelid *SkillBook* ja *InventoryItem* on selleks, et luua *Character* ja *Item* ning *Skill* vahel palju-palju seoseid. *InventoryItem* on ka lisaväljad, nagu esemete arv, et vältida topeltkirjeid, ja kahendmuutuja, et teha kindlaks, kas seda kaupa kasutati kordamööda, et mitte lubada sama kauba kasutamist mitu korda korraga.

Table *Friend* oli vaja ka selleks, et jälgida tegelasi, keda mängijad saaksid oma gruppi lisada.

Teine probleem, mis tekkis, oli küsimus, kuidas mänguseansi ajal tegelaste tingimusi salvestada. Lahenduseks oli luua tabel *BattleCharacter*, mis väljendaks grupi tegelasi, samuti koletisi, kellega nad võitlevad, ja nende praegust statistikat. Selle tabeli kirjed eksisteerivad ainult siis, kui mänguseanss on aktiivne ja mänguseansi lõppedes kustutatud.

Autentimine toimus ASP.NET Core'i abil ning raamistik käsitles kõiki kontode tabeleid ja andmeid.

Mõned olemid kasutavad *Guidsi primaarvõtmetena* parema turvalisuse ja ainulaadsuse tagamiseks, näiteks *AppUsers*, *Characters*, *BattleCharacters* ja *InRaid*- olemid. Teised olemid kasutavad *int-i* primaarvõtmete jaoks, et tagada parem jõudlus ja vähem salvestusruumi, näiteks *Monsters*, *Items* ja Dungeons. Andmebaasi olemi-suhte diagramm on näha Joonisel 1.



Joonis 1. Andmebaas Olemi-suhte diagramm.

4.1.2 Andmebaasi loomine

Rakendamisel kasutati *Entity Framework Core'i*, kasutades *Code-First* lähenemisviisi.

Code-First lähenemine tähendab, et kogu andmebaasi seadistus tehti nii, et kõigepealt loodi teie koodis klassid, mida nimetatakse mudeliteks, mida seejärel kasutatakse andmebaasis tabelite loomiseks, erinevalt lähenemisviisist *Database-First*, kus kõigepealt kujundate ja loote andmebaasi ning seejärel sobitate klassid andmebaasitabelitega. [11] [12] [13]

Domeenimudelite loomise lihtsustamiseks loodi *DomainEntityId* (Joonis 2) ja *DomainEntityMetaId*. Kuna *Entity Framework Core* nõuab, et igal olemil oleks *primaarvõti*, parandab baasklassi omamine koodi loetavust. *DomainEntityMetaId* on tuletatud domeenist *DomainEntityId*, kuid andmebaasi haldamise parandamiseks on sellel ka lisaväljad *CreatedBy, CreatedAt, UpdatedBy, UpdatedAt*.

```
public abstract class DomainEntityId : DomainEntityId<Guid>,
IDomainEntityId
{
}

public abstract class DomainEntityId<TKey> : IDomainEntityId<TKey>
    where TKey : IEquatable<TKey>
{
    public TKey Id { get; set; } = default!;
}
```

Joonis 2. DomainEntityId klass.

Mudeli loomine *Code-First* lähenemisviisis on üsna lihtne. *Item* olemi kasutamine näitena (Joonis 3). Atribuutide *Effect* ja *Skill* kasutab *Entity Framework* navigeerimisatribuutidena olemite vaheliste seoste loomiseks ja neid ei salvestata andmebaasi. [14]

```
public class Item : DomainEntityMetaId<int>
{
    public String Name { get; set; }
    public String Description { get; set; }
    public EItemCategory Category { get; set; }
    public int? SkillId { get; set; }
    public int? EffectId { get; set; }
    public Effect? Effect { get; set; }
    public Skill? Skill { get; set; }
    public int Price { get; set; }
    public int? MinLevelToBuy { get; set; }
}
```

Joonis 3. Item entity klass.

Pärast domeenimudelite loomist tuleb konfiguratsiooniosa. *Andmebaasiga* ühenduse loomiseks on vaja DbContexti. Esimene samm on luua klass *ApplicationDbContext* (Joonis 4) ja seejärel lisada see ASP.NET Core rakenduse konfiguratsioonis ulatusega teenusena. [15]

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
        : base(options)
    {
    }
}
```

Joonis 4. ApplicationDbContext klass kood.

Seejärel kasutatakse *mudelite Entity Framework Fluent API* konfiguratsiooni.

Entity Framework võimaldab mudeleid konfigureerida, kasutades andmemärkusi või *Fluent API*-d. Andmemärkustega mudelite konfigureerimine on väga otsekohene ja lihtne, kuna see on vajalik lihtsalt annotatsiooni lisamiseks mudeliklassi. *Fluent API* on keerulisem ja seda peetakse edasijõudnuks, kuna see nõuab konfiguratsioonide täpsustamist *OnModelCreating* meetod *DbContext*. [14]

Mõlemat lähenemisviisi saab kasutada samal ajal, kuid Fluent API kirjutab andmete annotatsiooni konfiguratsioonid üle. Fluent API peamine eelis on see, et see pakub rohkem võimalusi mudelite konfigureerimiseks. Kõike, mida saab konfigureerida andmemärkustega, saab konfigureerida Fluent API abil, kuid mitte kõike, mida saab konfigureerida *Fluent API-ga*, ei saa konfigureerida andmete märkuste abil. Sel põhjusel kasutati selles projektis peamiselt *Fluent API-d* mudelite konfigureerimiseks tänu laiendatud funktsionaalsusele ning koodi loetavamaks ja hooldatavamaks hoidmiseks.

Selleks, et Entity Framework näeks mudeleid, on vaja lisada need DbSet-idena ApplicationDbContext klassi või registreerida see OnModelCreating-is. Selles projektis lisati mudelid DbSet-idena (Joonis 5).

```
public class ApplicationDbContext : DbContext
{
    public DbSet<Character> Characters { get; set; } = default!;
    public DbSet<Effect> Effects { get; set; } = default!;
    public DbSet<Skill> Skills { get; set; } = default!;
    public DbSet<Item> Items { get; set; } = default!;
    ...
}
```

Joonis 5. Mudelite lisamine *DbContexti*.

Kasutades näitena *Item*-i olemit, on Fluent API abil võimalik konfigureerida *Item*-i ja *Effect* / *Skill* vahelisi seoseid. *ApplicationDbContext*-is on vaja üle kirjutada meetod *OnModelCreating Fluent API* kasutamiseks (Joonis 6). *Item* mudeli jaoks oli vaja luua üks-mitmele suhteid *Effect* ja *Skill* mudelitega. Joonisel 6 on näidatud konfiguratsioon, kus on määratud mudelite seos ja määratud võõrvõtmed.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<Item>(it =>
            {
                it.HasOne(i => i.Effect)
                    .WithMany()
                    .HasForeignKey(i => i.EffectId);

                it.HasOne(i => i.Skill)
                    .WithMany()
                    .HasForeignKey(i => i.SkillId);
            }
        );
    }
```

Joonis 6. *Item* olemi konfiguratsioon *Fluent API* abil.

Ülejäänud andmebaasimudelid konfigureeriti selle lähenemisviisi abil.

4.2 Süsteemi arendamine

Kui andmebaas on loodud, on aeg alustada rakenduse enda loomisega. Esimesed sammud olid arhitektuuri ülesehitamine, et tagada tarkvara struktureeritud arendamine ja hoida seda korras. Seejärel lisada autentimissüsteem turvaliseks andmekäitluseks. Pärast seda luua mänguloogika ise, peamiselt see, kuidas serveritarkvara töötleb muudatusi mängu, lahingu ja koletise loogika ajal, samuti omandades uusi objekte koopast, poest ja grupi moodustamisest.

Järgmine samm on andmete edastamiseks vajalike API lõpp-punktide loomine. Ja lõpuks, lisada toetus internaliseerimisele, et lõpptoodet saaksid nautida inimesed erinevatest riikidest ja piirkondadest.

4.2.1 Arhitektuuri ehitamine

Projekti alustati *Domain* kihi rakendamisega. Seal määratleti andmebaasi olemid ja seati nende vahelised seosed.

Järgmisena luuakse DAL-kiht koos olemi raamistiku rakendustega. Siin luuakse liidesed ja hoidlad koos andmeedastusobjektide mapperitega (DTOs).

BLL-is loodi teenused BLL-i ja API-kihi vaheliseks suhtluseks. Ka siin rakendati mänguloogikat.

Lõpuks loodi API kiht, et võimaldada klientidel serveriga suhelda.

Üldiselt näeb struktuur välja selline:

- Base.Domain – abstraktsed klassid domeeni olemite standardimiseks. Internaliseerimise tugiklass.
- Base.Extension – ASP.NET identiteedi laiendused.
- Base.DAL – *Data Access Layer* abstraktsed klassid.
- Base.BLL – *Business Logic Layer* abstraktsed klassid.
- Base.Contracts – abstraktsed liidesed erinevatele kihtidele.

- App.Domain – andmebaasi olemite määratlus.
- App.DAL.EF – *Entity Framework*-i ja andmebaasimudelite konfiguratsiooni rakendamine. Repositooriumide rakendamine.
- App.DAL.DTO – *Data Transfer Objects Data Access Layer*-i jaoks.
- App.Contracts.DAL – Repository liideste rakendamine.
- App.BLL – mänguloogika ja Services'i rakendamine.
- App.BLL.DTO – *Data Transfer Objects Business Logic Layer*-i jaoks.
- App.Contracts.BLL – Service liideste rakendamine.
- WebApp – rakenduse tuum. API kontrollerite rakendamine. Rakenduse seaded ja middleware.
- App.Public.DTO – *Data Transfer Objects API layer*-i jaoks.

4.2.2 Autentimine

Selle projekti väga oluline osa on autentimissüsteemi olemasolu. See on vajalik andmete eraldamiseks kasutajate vahel ja nende turvalisuse tagamiseks, et ainult ühel kasutajal oleks juurdepääs nende tegelasele ja mänguseanssidele.

Autentimine on süsteemile või rakendusele juurdepääsu püüdva kasutaja identiteedi kontrollimise protsess. Tavaliselt tehakse seda nii, et kasutajalt nõutakse turvalise teabe, näiteks kasutajanime ja parooli, esitamist.

Autentimistoe lisamine on ülioluline igas kasutajaspetsiifilise sisuga tegelevas rakenduses. Siin kirjeldatakse, kuidas autentimise tuge projektis rakendatakse.

ASP.NET Core'il on mugav sisseehitatud autentimissüsteemi tugi, mida selles projektis kasutati. Esiteks loodi rakendustel *IdentityUser* ja *AppRole* põhinevad olemid *AppUser* ja *IdentityRole*, et võimaldada lisafunktsioone. Seejärel lisatakse need programmi konfiguratsioonile (Joonis 7) ja andmebaasi konfiguratsioonile (Joonis 8). [16]

```
builder.Services.AddIdentity<AppUser, AppRole>(options => {
options.SignIn.RequireConfirmedAccount = false; })
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();
```

Joonis 7. Identiteedi konfiguratsioon

```
public class ApplicationDbContext : IdentityDbContext<AppUser,
AppRole, Guid>
{
    // DbSets
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        // Model configurations
    }

    public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
        : base(options)
    {
    }
}
```

Joonis 8. *DbContext* identiteediga

Rest API kontrollerite identiteeditoe jaoks tuleks teha täiendavaid muudatusi. Selles projektis kasutatakse JSON-i veebimärke (JWT-sid) API-päringute autentimise käsitlemiseks.

JWT-dega toimub autentimine nii: kui kasutaja logib sisse oma õigete kontoandmetega, antakse talle JWT. Seda JWT-d saab seejärel edastada API- päringutega kasutaja identiteedi kontrollimiseks. JWT konfiguratsioonid lisatakse rakenduse seadetele (Joonis 9).

```
"JWT": {
    "Key": "secretkey",
    "Issuer": "cryptsofbuzak.ee",
    "ExpireInMinutes": 7
  },
```

Joonis 9. JWT konfiguratsioon rakenduse seadetes.

Kuna JWT-de kasutamisega kaasneb oht, et kasutajaressurssidele on volitamata juurdepääs, on JWT-de eluiga madal. Kuid selleks, et vältida ebamugavat vajadust sisse logida kogu aeg, kui JWT on aegunud, rakendatakse mugavuse huvides *Refresh Token*-it. *Refresh Token*-itel on pikem eluiga ja neid pakutakse koos JWT loomisel. Need

võimaldavad klientidel värskendada oma JWT-sid, et pääseda juurde kaitstud ressurssidele. Kui JWT on aegunud, saab JWT-ga *Refresh Token*-i saata serverisse, et luua kliendile kasutamiseks uus JWT.

Esiteks tuleb luua olem *Refresh Token* (Joonis 10). See *Refresh Token* genereeritakse kasutaja iga sisselogimise või registreerimisega ja salvestatakse *AppUser*-iga.

```
public class RefreshToken: DomainEntityId
{
    // Current token data
    [StringLength(36, MinimumLength = 36)]
    public string Token { get; set; } = Guid.NewGuid().ToString();
    // UTC
    public DateTime TokenExpirationDateTime { get; set; } =
DateTime.UtcNow.AddDays(7);

    // Expired token data
    [StringLength(36, MinimumLength = 36)]
    public string? PreviousToken { get; set; }
    // UTC
    public DateTime PreviousTokenExpirationDateTime { get; set; }

    // Navigational property
    public Guid AppUserId { get; set; }
    public AppUser? AppUser { get; set; }
}
```

Joonis 10. Refresh Token olem

Lisaks autentimisele rakendatakse ka autoriseerimist, et eraldada kontole juurdepääs. Autoriseerimine on protsess, mille käigus antakse või keelatakse juurdepääs konkreetsetele ressurssidele või funktsioonidele autenditud kasutaja õiguste alusel.

Idee on omada administraatori roll, millel on võimalus muuta, kustutada või lisada *Item*, *Monster*, *Dungeon* jne. andmed, lisaks kasutaja rollile, millel on juurdepääs põhifunktsioonidele.

4.2.3 Mängu loogika

See teema kirjeldab projekti tuuma – mänguloogika rakendamist. Eesmärk on, et kõik mängu jaoks vajalikud funktsioonid oleksid rakendatud. Peamise mänguloogikaga tegelemiseks on loodud kaks klassi: *Core.cs* and *GameSession.cs*. *Core* vastutab mängu üldiste funktsioonide eest, samas kui *GameSessioni* klass tegeleb kõigega, mis on seotud mänguprotsessi endaga.

Esimene funktsioonide komplekt on:

- Mängija tegelase loomine.
- Tegelase tugevdamine kogemustega.
- Mängusessiooni loomine ja grupi moodustamine.
- Vangikongide ruumide genereerimine.

Kõigepealt oli vaja tegelaste loomise loogikat. Esimese rakendusena otsustati, et kõik tegelased algavad sama vaikimisi stat-väärtustega. Tulevikus oleks hea lasta mängijatel valida esialgne statistika koos alglimiidiga ja valida oma tegelasele stardiboonus.

Kui tegelane on loodud, on loogiline rakendada mängija tegelaste jaoks tugevdavat mehaanikut. Kuna tegelased saavad kogemusi ainult koobastes, rakendati loogikat *GameSessionis*. Mängija iseloomu tugevdamine toimus RPG kontseptsiooni *Level up* abil [17]. Kui tegelane saab vaenlaste alistamisest piisavalt kogemusi, "tasandavad" nad ja saavad tugevamaks. Pärast taset üles suurendatakse märkide statistikat fikseeritud väärtuse võrra ning nende HP ja MP taastatakse uuele maksimaalsele väärtusele (Joonis 11). Kuna taseme tõus toimub mänguprotsessi ajal, muudetakse *BattleCharacter'i*, et kajastada lahingu ajal taseme tõusu muutusi. Seda tehakse meelega, et aidata mängijaid lahingute ajal. Näiteks võib tekkida olukord, kus mängija võib sattuda kaotusseisu ja tegelase taseme tõstmine võib lasta mängijal lahingu võita. Selline haruldane meeldejääv hetk avaldab positiivset mõju mängija mängukogemusele.

```
private void LevelUpCharacter(BattleCharacter battleCharacter)
    {
        battleCharacter.Level += 1;
        battleCharacter.MaxHp += 5;
        battleCharacter.CurrentHp = battleCharacter.MaxHp; // Level up
replenish HP
        battleCharacter.MaxMp += 1;
        battleCharacter.CurrentMp = battleCharacter.MaxMp; // Level up
replenish MP
        battleCharacter.AtkStat += 1;
        battleCharacter.DefStat += 1;
        battleCharacter.SpdStat += 1;
    }
```

Joonis 11. Meetod GameSessionis.cs mis vastutab märkide statistika suurendamise eest pärast taseme tõusu.

Järgmiseks tuleb mängusessiooni loomine. See osa on üsna lihtne, luuakse uus GroupStatus objekt, millel on mängu alustanud mängija ID, sõbrategelaste ID-d ja koopa ID, kuhu nad sisenevad. Kui mänguseanss on määratud ja alanud, saabub aeg luua koopasse. Mängu koobaste esialgne rakendusidee on pooleldi fikseeritud lineaarsed koopad. See tähendab, et mängijad liiguvad koopas alati edasi lõpu poole etteantud ruumidega. Kuid ruumid, mida mängijad läbivad, ei ole alati samad. Ruumid, mida mängijad peavad läbima, valitakse saadaolevate ruumide loendist kaalutud võimaluse alusel. Igal *DungeonZone*'il on *Weighti* väärtus, mida kõrgem see on, seda suurem on selle tsooni valimise võimalus. Kaalu väärtus on mugavuse huvides lubatud vahemikus 0 kuni 100.

Kaalutud valiku rakendamist on näha Joonisel 12. Mängusessiooni ajal kutsutakse SelectDungeonZone meetodit kaks korda pärast iga ruumi täitmist, et mängija saaks valida järgmise ruumi, kui praeguse etapi jaoks on vaba ruumi rohkem kui üks. Valitud ruumid eemaldatakse loendist, et vältida mängijale haruldaste ruumide dubleerimist, erinevalt tavaruumidest kaaluga 100.

```
public static DungeonZone SelectDungeonZone(List<DungeonZone> zones)
    {
        int totalWeight = 0;

        // Total weight as a sum of all weights in the list
        foreach (DungeonZone zone in zones)
        {
            totalWeight += zone.Weight;
        }

        int randomNumber = _rnd.Next(0, totalWeight);

        DungeonZone? selectedZone = null;
        foreach (DungeonZone zone in zones)
        {
            if (randomNumber < zone.Weight)
            {
                selectedZone = zone;
                break;
            }

            randomNumber -= zone.Weight;
        }

        if (selectedZone != null && selectedZone.Weight != 100)
        {
            zones.Remove(selectedZone!);
        }

        return selectedZone!;
    }
```

Joonis 12. Kaalutud valiku rakendamise meetod.

Järgmine rakendatud funktsioonide komplekt:

- Esemete lisamine tegelaste inventarile.
- Saadaolevate esemete kasutamine.
- Uute oskuste omandamine.
- Oskuste kasutamine.
- Efektid muudavad tegelaste statistikat.

Esemete lisamine tegelaste inventari tehti, lisades need andmebaasi InventoryItem tabelisse. Konkreetse tegelase iga kordumatu ese oli tabelis rida, millel oli saadaval

selliste eset arvuandmed. Praegusest koopast leitud esemed salvestatakse aga tabelisse InRaidItems ja need teisaldatakse tabelisse InventoryItem alles siis, kui koopas on lõpetatud.

Esemete ja oskuste kasutamine on võimalik ainult koopas viibides. Esemetel on erinevad kategooriad: *Consumable*, *SkillBook*, *Artifact* and *Miscellaneous*. *Consuumable* ja *SkillBook* tüüpi esemed on ühekordselt kasutatavad. *Consumable* esemed annavad rühmale efekti, mis tugevdab või nõrgendab, samas kui *SkillBooki* ese õpetab tegelasele uut oskust. *Artefakt* tüüpi esemed annavad grupile passiivseid efekte, neid peetakse haruldasteks kalliteks esemeteks, mis aitavad rühma. *Miscellaneous* esemeid leidub ainult koopas ja need on mõeldud lihtsalt raha eest müümiseks.

Nii esemed kui ka oskused rakendavad rühmale efekti. Efektid võivad muuta erinevat statistikat ja neil võib olla erinev kestus. Mõjud võivad olla rühmale kasulikud või kahjulikud, näiteks võivad need suurendada rühmade rünnakujõudu või vähendada nende kaitsestatistikat. Praeguses teostuses mõjutavad efektid kõiki rühma tegelasi.

Efektid muudavad *BattleCharacter'i* statistikat ainult võitluse ajal, et vältida võimalikke soovimatuid püsivaid muutusi tegelast.

Järgmine oli lahingutega seotud funktsionaalsus:

- Käigupõhise võitlussüsteemi rakendamine.
- Võimaldada mängijatel lahingu ajal toiminguid valida.
- Tegevuste loomine koletistele lahingu ajal.
- Genereerida, millised esemed koletised pärast lüüasaamist maha jätavad.
- Kogemuse saamine koletiste võitmisest.
- Edasiminek järgmisse koopasse.
- Vangikoopa lõpetamine ja leitud esemete salvestamine. Mängusessiooni lõpetamine.
- Poe loomine esemete ostmiseks ja müümiseks.

Käigupõhise võitluse süsteemi arendamisel loodi CharActioni klass (Joonis 13), mis aitab lahingu ajal mängija tegevusi töödelda. Sellel on teave tegelase tegevuse, nende sihtmärgi, sooritatava toimingu ja järjekorra kohta, mis määrab, millal tegevus tehakse võrreldes teiste toimingutega.

```
public class CharAction
{
    public Guid CharId;
    public Guid TargetId;
    public bool IsSkill;

    // One of default actions, or skill id.
    public int ActionId;

    public int Order;

    public CharAction(Guid charId, bool isSkill, int actionId)
    {
        CharId = charId;
        IsSkill = isSkill;
        ActionId = actionId;
    }
}
```

Joonis 13. CharAction klass

Pärast mängija tegevuste määramist genereeritakse juhuslikult koletiste toimingud. Praeguses rakenduses saavad nad rünnata rühma liiget või kaitsta. Pärast seda sorteeritakse tegelaste grupitegevused koos koletiste tegevustega BattleCharacteri SpeedStati alusel. Mida suurem on tegelase kiiruse stat, seda varem nende tegevus toimub.

Lahing jätkub, kuni ühel poolel, mängijate rühmal või koletiste rühmal, on kõik liikmed alla 0 HP. Kui *BattleCharacteri* HP langeb 0-ni, ei saa nad enam lahingus osaleda.

Iga lüüa saanud koletise kohta saavad grupi tegelased kogemusi koletise statistika põhjal. Kogemuste hulk arvutatakse valemi (Joonis 14) alusel, mis võtab arvesse mängija tegelase ja koletise tasemete erinevust ning koletise statistikat.

```
private int ExperienceFromMonster(BattleCharacter battleCharacter,
BattleCharacter monster)
    {
        // DIF = MonLevel / CharLevel
        // (ATK + DEF) * (DIF + 0.5)

        float dif = (float)monster.Level / battleCharacter.Level;
        return (int) ((monster.AtkStat + monster.DefStat) * (dif + 0.5));
    }
```

Joonis 14. Meetod ExperienceFromMonster kogemuste arvutamise valemiga.

Koletiste võitmine autasustab mängijaid esemetega. Igal koletisel on *DropTableItem* tabelis määratletud esemed, mis võivad kukkuda. Seal on määratletud ese, mis langeb, koletis, mis selle kukutab, maksimaalne ja minimaalne arv esemeid, mis võivad langeda, ja võimalus, et ese langeb. Kukkunud esemed genereeritakse, kui kõik ruumis olevad koletised on võidetud ja lisatakse *InRaidItem*-si tabelisse.

Kui kõik ruumis olevad koletised on võidetud, võib mängija liikuda järgmisse ruumi. Kui selle etapi jaoks on saadaval mitu ruumi, on mängijal valida jätkamiseks üks kahest valitud ruumist. Kui koopasse pole enam ruume jäänud, loetakse see lõpetatuks ja kõik tabelis *InRaidItems* olevad esemed teisaldatakse *InventoryItemi*. Seejärel teisendatakse *BattleCharacteri* statistika *Characteri* statistikaks, et salvestada muudatused pärast tasandamist.

Juhul, kui mängijate grupp võideti lahingu ajal või kui mängija otsustas enne selle lõpetamist koopast lahkuda, salvestatakse ainult *BattleCharacteri* statistika ja *InRaidItems* kaotatakse.

Lõpuks rakendatakse kauplust, kus mängijad saavad esemeid müüa või osta. Seal saavad mängijad osta abistavaid esemeid, mis vastavad tegelaste tasemele, või müüa koopast leitud esemeid.

4.2.4 API loomine

ASP.NET Core'il on kontrolleripõhise veebi API tugi [18]. Seda lähenemist kasutatakse projekti API-de loomisel.

Idee on omada API lõpp-punkte, mida klientrakendus saab kasutada serveriga suhtlemiseks andmete saamiseks või saatmiseks. Kliendi ja serveri vaheliseks suhtluseks kasutatakse *App.Public.DTO* objekte.

ASP.NET Core toetab atribuute, mida saab kasutada API kontrollerite käitumise määramiseks. Atribuute kasutatakse ka erinevate API-päringute juurdepääsutaseme määramiseks. Atribuuti [Authorize] kasutatakse selleks, et määrata, millised rollid peavad kasutajal olema, et meetodile juurde pääseda. [19]

API kontrollerite loomiseks on mõned reeglid. Esiteks peab kontrollerite nimi lõppema kontrolleriga, klass peab olema *public* ja kontrollerite nimed peavad olema unikaalsed kõigis nimeruumides. Web API-l on mitmeid toiminguid, mida kasutatakse kontrollerites ja mis on määratud atribuutide abil. Kõige tavalisemad toimingud on *GET, POST, PUT, DELETE*. [20]

Esiteks loodi sisselogimiseks ja registreerimiseks API *AccountController*. Selles kontrolleris luuakse järgmised sisestusmeetodid:

- Konto sisselogimine
- Konto registreerimine
- JSON Web Token värskendamine.

Järgmisena luuakse *Character'i* API-kontroller . Sellel kontrolleril on meetodid:

- Praeguse kasutajategelase teabe hankimine
- Praegusele kasutajale tegelase loomine

Kasutades näitena funktsiooni *GetCharacterByUser* (Joonis 15), kasutatakse meetodi puhul järgmisi atribuute:

- [Produces/Consumes] – see atribuut määrab, millist andmetüüpi see API toiming tagastab/aktsepteerib.
- [ProducesResponseType] – see atribuut määrab, milliseid olekukoode saab API-toiminguga tagastada.

- [HttpGet] – see atribuut tuvastab, et toiming toetab HTTP GET-meetodit.
- [Authorize] – määrab, millised kasutajarollid sellele meetodile juurde pääsevad ja milliseid autoriseerimisskeeme kasutatakse.

```
[Produces("application/json")]
[Consumes("application/json")]
[ProducesResponseType(typeof(App.Public.DTO.v1.Character),
StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[HttpGet("User")]
[Authorize(Roles = "admin,user", AuthenticationSchemes =
JwtBearerDefaults.AuthenticationScheme)]
public async Task<ActionResult<App.Public.DTO.v1.Character>>
GetCharacterByUser()
{
     var character = await
_bll.Characters.FirstOrDefaultAsync(User.GetUserId());

     if (character == null)
     {
               return NotFound();
     }

     var characterPublic =
_mapper.Map<App.Public.DTO.v1.Character>(character);

     return characterPublic;
       }
```

Joonis 15. GetCharacterByUser meetod Character API kontrolleris.

Peamine API kontroller on *GameController*. See kontroller on koht, kus hoitakse peamist sidet. Sellel kontrolleril on meetodid:

- Praeguse mängu oleku saamine.
- Mängu initsialiseerimine.
- Tegelaste gruppi lisamine ja eemaldamine.
- Mängu alustamine ja järgmisse kooparuumi minek.
- Lahingu toimingute seadmine. Esemete ja oskuste kasutamine.

API kontroller kasutab BLL-i mänguloogikat ja tagastab kogutud andmed kliendile tagasi.

4.2.5 Internaliseerimine

Internaliseerimine, tuntud ka kui "i18n", on oluline osa, mis tuleb kaasaegsetesse rakendustesse lisada. Rakenduse kättesaadavus laiemale kasutajaskonnale on hädavajalik mis tahes veebitarkvara, sealhulgas selle mänguprojekti jaoks. Sel põhjusel otsustati sellele mängule lisada internaliseerimise tugi.

Esiteks lisatakse rakenduse sätetele toetatud kultuuride loend koos vaikekultuuriga (Joonis 16) ja teenused lisatakse ASP.NET konfiguratsioonis (Joonis 18).

```
"DefaultCulture": "en-GB",
"SupportedCultures": [
  "en-GB",
  "en-US",
  "et-EE",
  "ru-RU",
  "lt-LT"
]
```

Joonis 16. Rakenduse kultuuri seaded.

Andmebaasiobjektide, näiteks esemete, koletiste, oskuste ja efektide lokaliseerimiseks luuakse uus *LangStr* klass. *LangStr* on *sõnastiku* kohandatud rakendus, mis tegeleb kultuuriloogikaga. Kuna sõnastikke ei saa andmebaasi salvestada, toimub andmete serialiseerimine ja deserialiseerimine Entity Framework [Column(TypeName = "jsonb"] annotatsiooni (Joonis 17) abil, et teisendada see JSON-ideks andmebaasi salvestamiseks.

```
public class Item : DomainEntityMetaId<int>
{
    [Column(TypeName = "jsonb")]
    public LangStr Name { get; set; } = new();
    [Column(TypeName = "jsonb")]
    public LangStr Description { get; set; } = new();
    ...
}
```

Joonis 17. Item olem, mis kasutab LangStri.

```
var supportedCultures = builder.Configuration
    .GetSection("SupportedCultures")
    .GetChildren()
    .Select(x => new CultureInfo(x.Value))
    .ToArray();
builder.Services.Configure<RequestLocalizationOptions>(options =>
{
    options.SupportedCultures = supportedCultures;
    options.SupportedUICultures = supportedCultures;
    options.DefaultRequestCulture =
        new RequestCulture(builder.Configuration["DefaultCulture"],
            builder.Configuration["DefaultCulture"]);
    options.SetDefaultCulture(builder.Configuration["DefaultCulture"]);
    options.RequestCultureProviders = new List<IRequestCultureProvider>
    {
        new QueryStringRequestCultureProvider(),
        new CookieRequestCultureProvider()
    };
});
```

Joonis 18. Internaliseerimisteenused rakenduse konfiguratsioonis.

4.3 Testimine

Arenduse käigus tehti funktsionaalsuse testimine *enamasti käsitsi Postman* abiga. API-päringuid testiti põhjalikult, et veenduda, et need töötavad ettenähtud viisil.

Tarkvara testimine on oluline osa arendusest, mis aitab pakkuda kasutajatele paremat kogemust. Tulevikus on plaanis projekti jaoks teha automatiseeritud testid.

5 Kokkuvõte

Selle projekti eesmärk oli luua serveritarkvara Dungeon Crawleri veebimängule, mis töötleks mängu loogikat. Serverirakendusel peaks olema rakendatud mängu põhifunktsioonid. API lõpp-punktid tuleb tulevaste klientrakenduste jaoks nende kasutamiseks luua.

Võib kindlalt öelda, et selle projekti peamised eesmärgid saavutati. Loodi serverirakendus ja rakendati mängu loogika põhifunktsioonid. Praegune rakendatud mänguloogika võimaldab mängu alustada ja mängida selle põhifunktsiooniga, milleks on ohtlike vangikongide uurimine mängijate vahelise suhtlusega. Kuigi selle mängu online-suhtlust rakendati minimaalsel tasemel, luues põhisüsteemi, saab seda aspekti tulevikus hõlpsasti edasi arendada.

Selle serverirakenduse jaoks loodi vajalikud API lõpp-punktid. Kui see on tehtud, hõlmavad mängu arendamise järgmised sammud kliendipoolse rakenduse loomist, et võimaldada kasutajatel mängu kogeda.

Tarkvara töötati välja nii, et see oleks hooldatav ja seda oleks lihtne edasi arendada. Seda silmas pidades arendatakse projekti edasi, et saavutada suurepärane veebimängu Dungeon Crawleri kogemus.

Kasutatud kirjandus

[1] S. UR, „Chronological List of Dungeon Crawlers,“ 26 Jaanuari 2021. [Võrgumaterjal]. Available: https://boardgamegeek.com/geeklist/260737/chronological-list-dungeon-crawlers. [Kasutatud 2023 Aprill 20].

[2] N. Brewer, „Going Rogue: A Brief History of the Computerized Dungeon Crawl,“ 7 Juuli 2016. [Võrgumaterjal]. Available: https://insight.ieeeusa.org/articles/going-rogue-a-brief-history-of-the-computerized-dungeon-crawl. [Kasutatud 18 Aprill 2023].

[3] R. Rutherford, „The Creation of PEDIT5,“ 31 August 2008. [Võrgumaterjal]. Available: https://web.archive.org/web/20110707162725/http://armchairarcade.com/neo/node/1948. [Kasutatud 18 Aprill 2023].

[4] „Dungeoneers Game,“ Rogue Sword: Strategy & Adventure, 15 November 2022. [Võrgumaterjal]. Available: https://www.dungeoneers.com/. [Kasutatud 20 Aprill 2023].

[5] Python Software Foundation, „General Python FAQ,“ Python Software Foundation, [Võrgumaterjal]. Available: https://docs.python.org/3/faq/general.html. [Kasutatud 21 Aprill 2023].

[6] „What is Java technology,“ Oracle, [Võrgumaterjal]. Available: https://www.java.com/en/download/help/whatis_java.html. [Kasutatud 21 Aprill 2023].

[7] „Introduction to Node.js,“ OpenJS Foundation, [Võrgumaterjal]. Available: https://nodejs.dev/en/learn/. [Kasutatud 21 Aprill 2023].

[8] „A tour of the C# language,“ Microsoft, 02 Veebruari 2023. [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. [Kasutatud 21 Aprill 2023].

[9] „What is ASP.NET,“ Microsoft, [Võrgumaterjal]. Available: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet. [Kasutatud 21 Aprill 2023].

[10] Jarnpher-Rice, „NancyFX Introduction,“ [Võrgumaterjal]. Available: https://github.com/NancyFx/Nancy/wiki/Introduction. [Kasutatud 21 Aprill 2023].

[11] „Entity Framework Core,“ Microsoft, 25 Mai 2021. [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/ef/core. [Kasutatud 24 Aprill 2023].

[12] „What is Code-First?,“ EntityFrameworkTutorial.net, [Võrgumaterjal]. Available: https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx. [Kasutatud 24 Aprill 2023].

[13] „Entity Framework Core,“ EntityFrameworkTutorial.net, [Võrgumaterjal]. Available: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx. [Kasutatud 24 Aprill 2023].

[14] „Creating and Configuring a Model,“ Microsoft, 28 Märts 2023. [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/ef/core/modeling/. [Kasutatud 24 Aprill 2023].

[15] „DbContext Lifetime, Configuration, and Initialization,“ Microsoft, 18 Veebruari 2023. [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/ef/core/dbcontext-configuration/. [Kasutatud 24 Aprill 2023].

[16] „Overview of ASP.NET Core authentication,“ Microsoft, 11 Juuli 2022. [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/authentication. [Kasutatud 8 Mai 2023].

[17] W. J. K. III, „Design Patterns of Successful Role-Playing Games,“ Internet Archive, 2013, pp. 56-58.

[18] „Create web APIs with ASP.NET Core,“ Microsoft, 4 November 2023. [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/core/web-api. [Kasutatud 8 Mai 2023].

[19] „Authentication and Authorization in ASP.NET Web API,“ Microsoft, 5 November 2022. [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/web-api/overview/security/authentication-and-authorization-in-aspnet-web-api. [Kasutatud 8 Mai 2023].

[20] M. H. Mithun Pattankar, Mastering ASP.NET Web API, Packt Publishing, 2017.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Aleksander Ozerov

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Dungeon Crawleri veebimängu serveritarkvara arendus" , mille juhendaja on Einar Kivisalu
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh 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 käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

11.05.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.