Pathfinder 2E Character Generator

Itsearviointi

Jyväskylän yliopiston ohjelmointityö
Riku Lehkonen

TIIVISTELMÄ

Tekijä	Riku Lehkonen	**Vuosi** 2020
Työn nimi	Pathfinder 2E Character Generator	
Työn ohjaaja	Jonne Itkonen	

TIIVISTELMÄ

Pelihahmon luominen Pathfinder (2E) pelissä on pitkä prosessi, joka vaatii pelaajaa tarkistamaan useita yksityiskohtia sen eri vaiheissa. Prosessissa on tärkeää tietää miten pelimekaniikat toimivat ja miten kyseisen hahmon rotu, luokka ja tausta vaikuttavat sen toimintaan pelimaailmassa. Tämän vuoksi ohjelmointityötä varten rakennetaan ohjelma tekemään kyseinen prosessi automaattisesti.

Ohjelmointityössä ohjelmointikielenä toimii C# ja käyttöliittymän rakentamisessa käytetään Unitya. Teknisen toteutuksen rakenteessa hyödynnetään luokkien jakoa UI:n, Hahmon ja XML -rakenteiden välillä. Versionhallintana toimii Github.

Ohjelmointityön tavoitteena on kehittää ymmärrystä generoinnin vaatimuksien rakentamiseen, XML tiedostojen hyödyntämiseen tiedonlähteinä, Unityn yhteistyöhön C# kanssa, sekä tukea pitkäaikaisen ohjelmoinnin vaatimuksien ymmärrystä.

Avainsanat Pathfinder 2E, Character Generation, Ohjelmointityö

Sivut 12, joista liitteitä n. 1 sivu

Sisällysluettelo

Liitteet

1 KUVAUS

Pelihahmon luominen Pathfinder (2E) pelissä on pitkä prosessi, joka vaatii pelaajaa tarkistamaan useita yksityiskohtia sen eri vaiheissa. Prosessissa on tärkeää tietää miten pelimekaniikat toimivat ja miten kyseisen hahmon rotu, luokka ja tausta vaikuttavat sen toimintaan pelimaailmassa. Tämän vuoksi ohjelmointityötä varten rakennetaan ohjelma tekemään kyseinen prosessi automaattisesti.

Keskeisenä osana ohjelmointityötä on luoda järjestelmä, jonka perusteella ohjelma voi generoida toimivia hahmoja, joita ei voida selkeästi erottaa pelaajan luomista hahmoista. Ohjelmointityön tuntimäärän vuoksi keskitytään ensisijaisesti toimivan hahmon luomisen rakentamiseen, joka sisältää muun muassa hahmon taidot, rodun ja kyvyt.

Toimivan hahmon luomisen jälkeen keskitytään siihen, miten kyseisten ominaisuuksien valintaan ja painotetaan automaattisessa hahmon luomisessa, jolloin ohjelma pystyisi kehittämään oikeaoppisia hahmoja. Valitettavasti ohjelmointityön työajan määrän vuoksi tässä ohjelmointityössä ei ehditä hyödyntämään minkäänlaista koneoppimista aikaisemman tekoälyn käytännönkokemuksen puutteen vuoksi. Tämän vuoksi generoituja hahmoja ohjataan ainoastaan heikon painotuksen avulla, joka vaikuttaisi toimivan riittävästi.

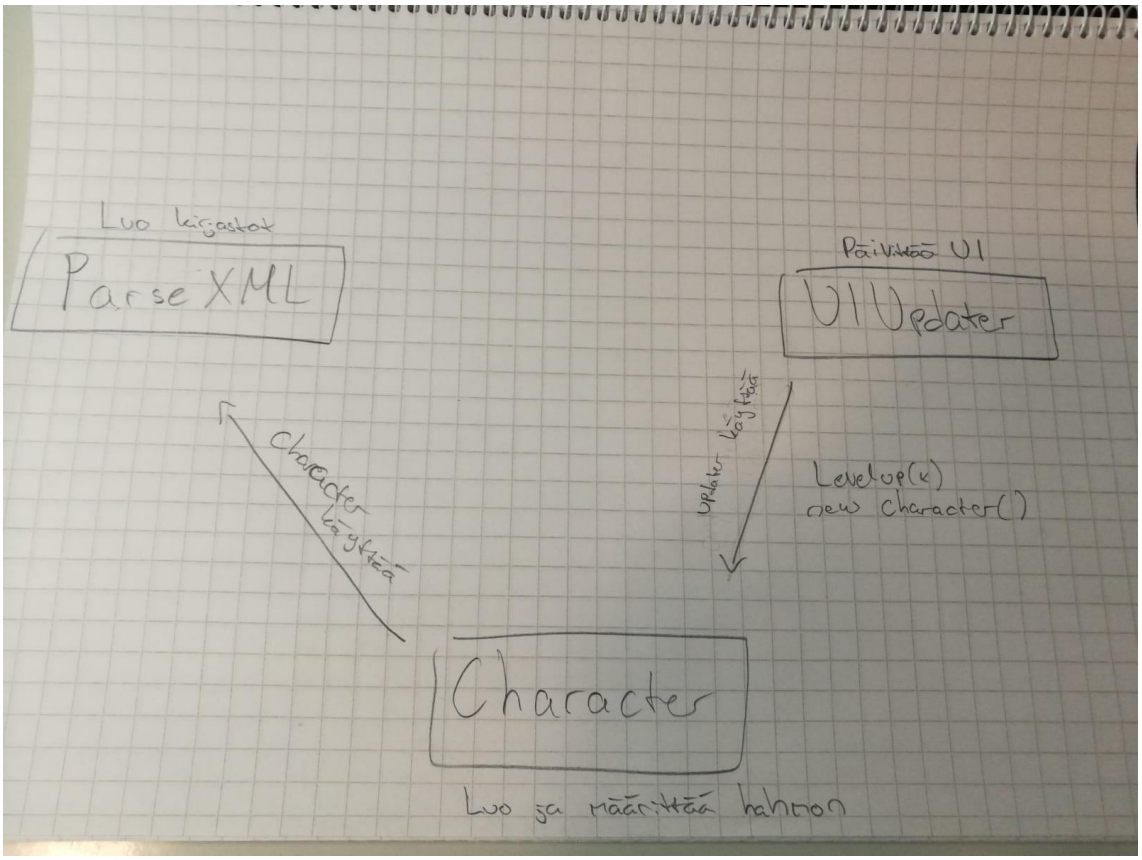
Ohjelmointityön tavoitteena on kehittää ymmärrystä generoinnin vaatimuksiin, XML tiedostojen hyödyntämiseen tiedonlähteinä, Unityn yhteistyöhön C# kanssa, sekä pitkäaikaisen ohjelmoinnin vaatimuksien ymmärrystä.

2 KÄYTÄNNÖN TOTEUTUS

Ohjelmointityössä ohjelmointikielenä toimii C# ja käyttöliittymän rakentamisessa käytetään Unitya. Versionhallintana toimii Github. Teknisen toteutuksen rakenteessa hyödynnetään luokkien jakoa UI:n, Hahmon ja XML -rakenteiden välillä.

XML -tiedostoista luodaan Dictionary olioita, joiden avulla Hahmo -luokka pystyy kehittämään hahmoa tarpeen mukaan. UIUpdate -luokka käyttää Hahmo -luokkaa itse generoitavan hahmon rakentamiseen ja lopulta käyttää Unityssä C# skriptejä kyseisen hahmon tietojen siirtämiseen käyttöliittymään. Unityn skriptit luovat elementtejä UI:n sisälle, UIUpdate kutsuu näitä skriptejä ja siirtää hahmon tietoja näiden avulla.

Joitakin elementtejä on täytynyt kovakoodata myöhemmän järjestelmän ymmärtämisen jälkeen ajansäästön vuoksi, nämä elementeille olisi täytynyt luoda uusia XML -tiedostoja ja toimintatapoja. Lopputuloksena ohjelmalle on toimiva sattumanvarainen hahmonluonti -järjestelmä, jonka luotuja hahmo ei pystytä erottamaan ihmisen luomista hahmoista.



Liite 1 Pääluokkien suhteet

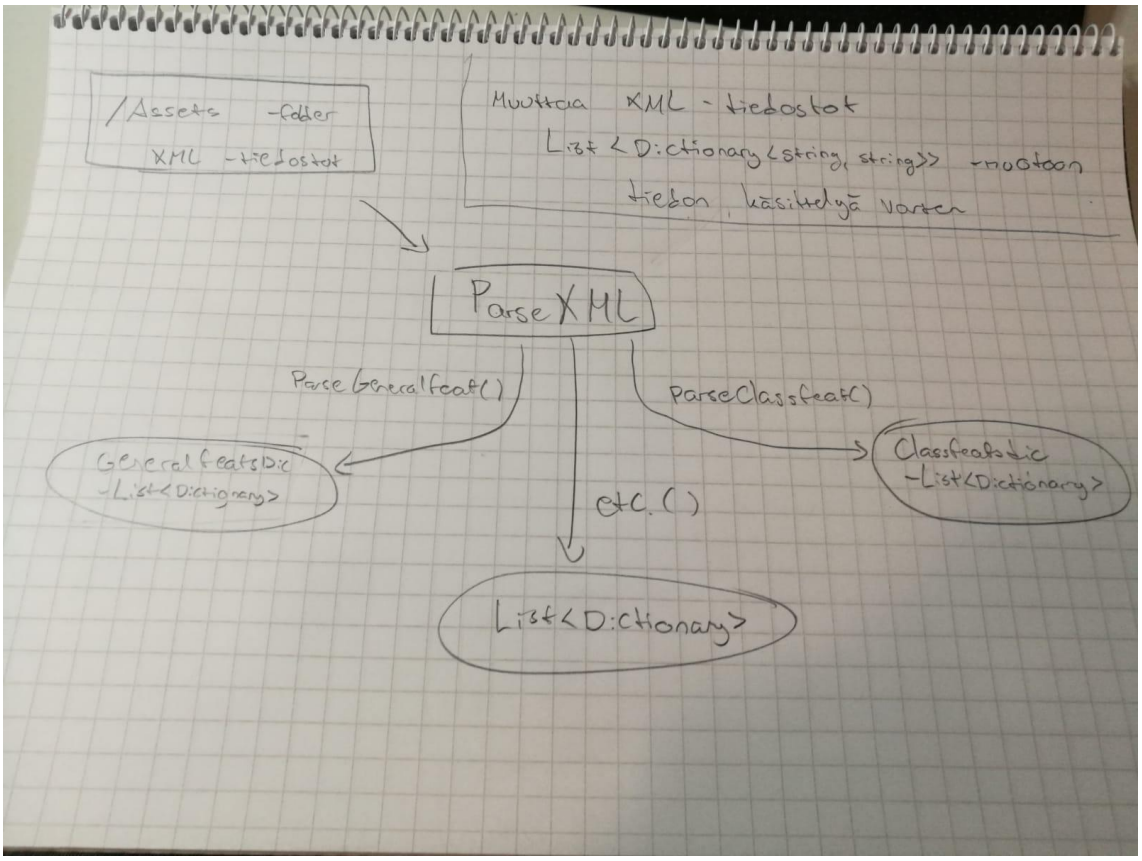
2.1 **ParseXML sisäiset metodit**

ParseXML toimii XML -tiedostojen muuttajan dictionary ja listDictionary muotoon, joiden avulla muut luokat ymmärtävät XML -tiedostoissa olevat tiedot. ParseXML:n metodeja ei käytetä muuten, kuin ohjelman avaamisvaiheessa, jossa se muuttaa XML -tiedostot haluttuun muotoon. Tämän jälkeen kyseisiä tietoja käsitellään aina kutsulla ParseXML:n kautta.

ParseXML metodit toimivat identtisesti toistensa kanssa, mutta ne vaativat eri toimenpiteitä riippuen mikä XML tiedosto on kyseessä. Tässä ohjelmointityössä testattiin eroja yhden ison XML tiedoston suodattamisessa tarpeen mukaan (proficiencies, feats, ancestries, advances) ja useiden pienempien XML -tiedostojen käytössä, joiden kutsuminen täytyy erikseen määritellä (progressions).

ParseXML:n tuottamat Dictionaryt sisältävät tietoa string -formaatissa, ja niissä käytetään erikoismerkkeinä ’|’, ’-’ ja ’/’. Jokainen erikoismerkki

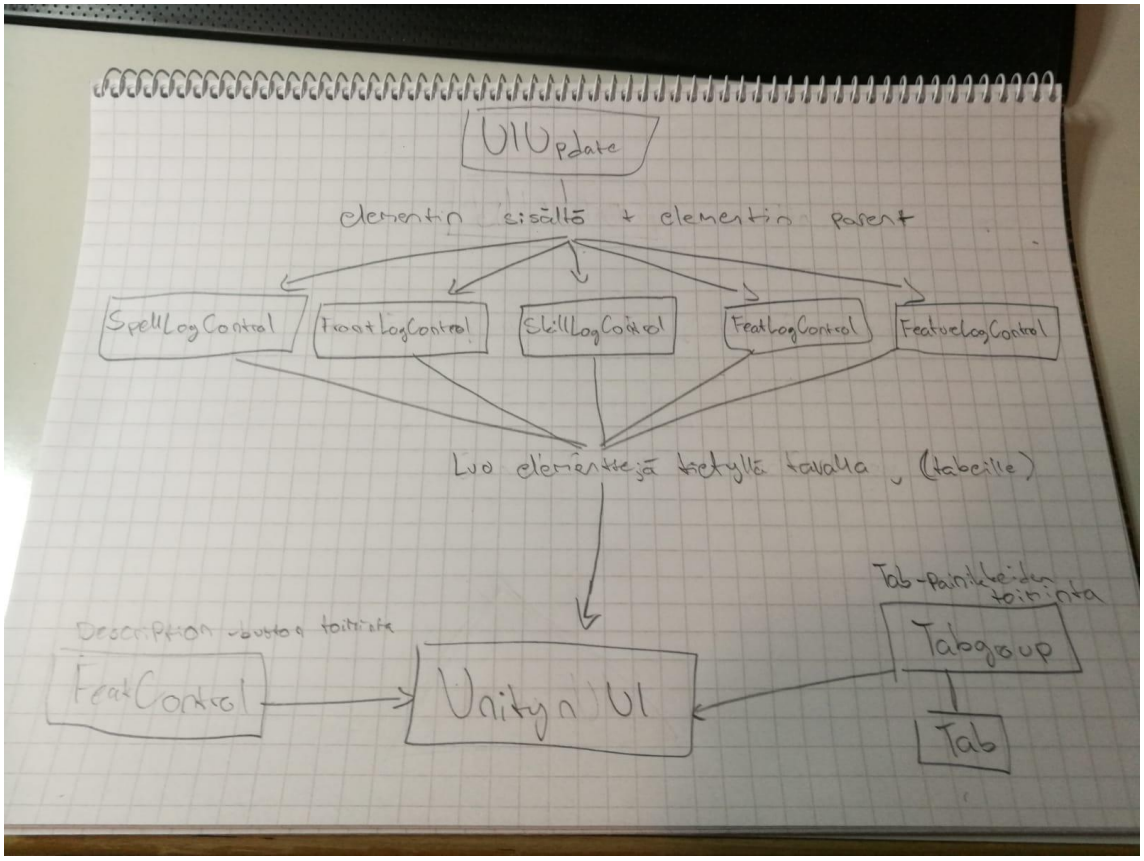
toimii ohjelman kannalta eri tavalla. ’|’ toimii erottelijana, jossa sen vasemmanpuoleinen string on määrittelijä, ja oikeanpuoleinen määriteltävän arvo. ’-’ -merkki toimii OR -ehtona, jota käytetään usein vaatimuksissa, tai useissa valinnoissa hahmon kehityksen aikana. ’/’ -merkki taas toimii AND -ehtona, joka on usein yhteydessä samanaikaisesti ’-’ -merkin kanssa. Tämän rakenteen avulla, yksi string -muuttuja sisältää kaikki tarvitut tiedot.



Liite 2 ParseXML metodit

2.2 **UIUpdate sisäiset metodit**

UIUpdate sisältää metodit jokaiseen Unitya kontrolloivaan luokkaan. Näissä luokissa määritellään mitä eri luokan komennoille annetaan tietona. Unityn kontrolloivat luokat toimivat ainoastaan elementtien lisääjinä, kun taas UIUpdaten puoli määrittelee mitä tietoa kyseisiin elementteihin pitää lisätä. Tämän vuoksi käytännön tasolla jokainen metodi UIUpdatessa on periaatteessa oma luokkansa Unityn skriptien toiminnallisuuden vuoksi. Tämän vuoksi tässä vaiheessa kuvataan UIUpdate toiminnallisuutta korkeammalta tasolta luokkien suhteista, eli UI:n toimintatavoista. Alla on kuva metodien toimintatarkoituksista, UI:n näkökulmasta.



Liite 3 UI toiminta

2.3 **Character luokan -metodit**

Character luokassa muokataan kyseistä hahmoa. Käsittely tapahtuu LevelUp() -metodit avulla, joka tuottaa prosessin automaattisesti joka levelille, kyseisen levelin classin progression dictionaryn avulla. Kaikki tieto mikä hahmoon lisätään, tulee ParseXML -luokan tuottamista Dictionary tai List<Dictionary> olioista. Hahmo sisältää sisäisesti käytettyjä tietoja, kuten HPPerLevel, class, background, ym. ominaisuuksia. Hahmon tiedot luokan ulkopuolelle otetaan selville sen metodien avulla, joitakin yksityiskohtia lukuunottamatta.

Alla on lyhyt listaus jokaisen metodit toiminnasta, sen tarvitsemista parametreista ja mitä muita metodeja se hyödyntää prosessin aikana. Ohjelmassa itsessään on paljon yksityiskohtaisempi kommentointi jokaisen metodin toimintaan, koska kyseisen pelikohteen toimintatavoissa on erikoisuuksia, joita ei kannata kirjata tässä jokaiseen metodiin esille. Oletuksena on, että alakappaleet auttavat ymmärtämään miten metodit toimivat yhdessä, eikä se, miten kyseiset metodit toimivat yksityiskohtaisesti, koska tällöin kommentit ohjelmakoodissa olisivat turhia.

- **UpdateCharStat**(string stat)
  - Kasvattaa hahmon statteja
  - Käyttää:
    - UpdateMods() – päivittää modifierit

- **CapitalizeFirstChar**(string str)
  - Ensimmäinen kirjain isoksi

- **LowerFirstChar**(string str)
  - Ensimmainen kirjain pieneksi

- **ParseRequirement**(string requirement)
  - Tulkkaa AND -ehdon ja tutkii vaatimuksien bool arvot
  - Käyttää:
    - ParseRequirementOR() – tarkistaa OR vaatimuksen
    - CheckRequirement() – tarkistaa vaatimuksen

- **ParseRequirementOR**(string requirement)
  - Tulkkaa OR -ehdon ja tutkii vaatimuksien bool arvot
  - Käyttää:
    - CheckRequirement() – tarkistaa vaatimuksen

- **CheckRequirement**(string requirement)
  - Tutkii vaatimuksen bool arvon

- **CheckFeats**(string searchedFeat)
  - Etsii, onko hahmolla parametrin feat

- **GetStat**(string statName)
  - Antaa statin arvon

- **GetMod**(string statName)
  - Antaa statin modin arvon

- **GetGeneralFeat**(string featName)
  - Etsii halutun featin tiedot GeneralFeatDic:sta

- **FindClassFeat**(string featName)
  - Etsii halutun featin tiedot ClassFeatDic:sta

- **GetFeats**()
  - Antaa hahmon featit

- **GetSkills**()
    - Antaa hahmon skillit

- **GetFeatures**()
    - Antaa hahmon ominaisuudet

- **IncreaseSkill**(string skillName)
    - Tulkkaa annetun stringin
    - Kasvattaa skilliä tulkinnaa mukaisesti

- **LevelUp**(int level)
    - Toimii hahmon alustajana ja levelin kasvattajana
    - Jos lvl = 0, alustaa hahmon, muuten kasvattaa
    - Käyttää:
        - RandomClass() - alustus
        - FindProgression() - kehitys
        - LevelUp() – rekursiivinen tason kasvatus

- **RandomClass**()
    - Generoi hahmolle uuden classin
    - Etsii kyseisen classin kasvatustiedoston (method)
    - Päivittää hahmoa classin alustuksen mukaisesti (method)
    - Antaa classille uniikin class -featin
        - Hoitaa erikoistoimenpiteet uniikille featille
    - Käyttää:
        - FindProgressionDic() – progression etsintä
        - ApplyProficienciesEffects() – classin alustus
        - ApplyFeatSkills() – initial feat efektit

- **ApplyProficienciesEffects**()
    - Käy läpi proficiencies -tiedoston tiedot, päivittää hahmoa sen alustuksen mukaisesti ja generoi tarvittavia ominaisuuksia sen eri vaiheissa (Ancestry, Background, FourBoosts), samalla ottaen huomioon pelin perisäännöt valintojen eri vaiheissa.
    - Käyttää:
        - IncreaseSkill() – kasvattaa skilliä
        - GetMod() – antaa modifierin
        - RandomizeNewLanguages() – antaa lisäkieliä
        - CheckFeats() – tarkistaa onko hahmolla feat
        - RandomAncestry() – alustaa rodun
        - RandomBackground() – alustaa taustan
        - ApplyFourBoosts() – viimeistelee statit

- **ApplyFourBoosts**()
    - Lisää viimeiset neljä stattien kasvatusta pelisääntöjen mukaisesti
    - Käyttää:
        - UpdateCharStat() – päivittää statteja

- **RandomizeNewLanguages**(int langAmount)
    - Etsii hahmolle uusia ja uniikkeja kieliä annetun määrän mukaisesti

- **ApplyInitialFeatSkills**(string inFeat)
    - Lisaa aloitusfeatin ominaisuuksia, kuten skilleja ja featteja
    - Käyttää:
        - IncreaseSkill() – kasvattaa skilliä
        - FindClassFeat() – etsii kyseistä feattiä class dictionarysta

- **RandomAncestry**()
    - Generoi hahmolle rodun ja tekee hahmoon muutoksia sen perusteella (method)
    - Käyttää:
        - ApplyAncestryEffects() – alustaa rodun efektit
        - AddAdvancement() – lisää kehityksiä

- **ApplyAncestryEffects**(Dictionary<string,string> ancestryInfo)
    - Tekee hahmoon muutoksia rodun perusteella (pelin sääntöjen mukaisesti)
    - Käyttää:
        - UpdateCharStat() – kasvattaa stattia

- **RandomBackground**()
    - Generoi hahmolle taustan ja tekee hahmoon muutoksia sen perusteella (method)
    - Käyttää:
        - ApplyBackgroundEffects() – alustaa taustan efektit

- **ApplyBackgroundEffects**(Dictionary<string,string> backgroundInfo)
    - Tekee hahmoon muutoksia sen taustan perusteella
    - Käyttää:
        - UpdateCharStat() – kasvattaa stattia
        - UpdateMods() – päivittää modifierit
        - IncreaseSkill() – kasvattaa skilliä

- **ApplyAdvancementEffects**(Dictionary<string,string> advancement)
    - Lisää hahmolle saadun advancementin efektit hahmolle, kuten kieliä, jumala, uusia feattejä, etc. (method)
    - Käyttää:
        - AddAdvancement() – lisää kehityksiä
        - IncreaseSkill() – kasvattaa skilliä
        - LowerFirstChar() – formatointiin metodi

- **FindProgression**()
    - Etsii hahmon tämänhetkisen tason (levelin) kehitykset, ja lisää niitä (method)
    - Käyttää:
        - AddAdvancement() – lisää kehityksiä

- **FindProgressionDic**()
    - Etsii hahmon kehitysmääritteiden tiedoston, sen classin mukaisesti

- **AddAdvancement**(string advancementName)
    - Lisää kyseisen advancementin, kun sen etsintä on delegoitu toiselle aliohjelmalle. Sisältää featit, skill increase, advancement, etc.
    - Randomgenissä (feat) kaikki valinnat filteröidään vaatimuksien mukaisesti ensin ja siirretään painotuksen kautta lisäykseen(method)
    - Skill Increase rajataan pelin sääntöjen mukaisesti
    - Class kohtainen advancement ja sen efektit lisätään
    - Käyttää:
        - FilterDictionary() – filteröi dictionaryn
        - WeightedChoice() – painottaa valintaa
        - IncreaseSkill() – kasvattaa skilliä
        - UpdateCharStat() – kasvattaa statteja
        - ApplyAdvancementEffect() – lisää kehityksen efektit

- **WeightedChoice**(List<Dictionary<string,string>> dicToFilter)
    - Painottaa mikä feat valitaan sille annetusta listasta

- **FilterDictionary**(List<Dictionary<string,string>> dicToFilter, string dicType)
    - Filteröi annetun listan sen tyypin vaatimuksilla
    - Käyttää:
        - ParseRequirement() – tarkistaa vaatimuksia
        - CheckFeats() – tarkistaa onko feat valittu jo
        - CapitalizeFirstChar() – formatointiin metodi

3 OMAN TYÖN ARVIOINTI

Ohjelmointityön alkuperäisen suunnitelman tavoitteet täyttyivät, mutta eri toimintatavoilla kuin toivoin. Kandityöni aihe oli vahvasti proseduraaliseen tuottamiseen liittyvää ja olen yliopisto-opintojeni aikana aina vahvasti lukenut tekoälyyn liittyvää materiaalia. Valitettavasti ohjelmointityön aikataulun vuoksi en kyennyt kokeilemaan käytännössä tekoälyn liittämistä hahmonluontiprosessiin. Pelin yksityiskohtien ja erikoistapauksien lisääminen ja XML -tiedostojen rakentaminen ja muokkaus vaativat liikaa aikaa, joten itse tekoälyn lisääminen ohjelmaan oli liian pitkä prosessi tämän kurssin yhteydessä.

Vaikka työn alussa käytettiin paljon aikaa suunnitteluun ongelmakohtien välttämiseksi, niitä silti tapahtui. Joissakin tapauksissa XML -tiedostojen erittely olisi ollut hyödyksi, kun taas toisissa tapauksissa niiden yhdistely olisi ollut parempi ratkaisu. Jälkikäteen ajatellen olisi todennäköisesti ollut esimerkiksi parempi eritellä Initial Feat -ominaisuudet erilleen kaikesta muusta, koska nämä vaikuttavat hahmoon vahvasti (ovat melkein hahmon omien classien sub-classeja).
Toisena osiona oli Deity ja Spellit, tämän ohjelmointityön aikataulun vuoksi näitä ei lisätty, mutta niiden pohjustaminen XML -tiedostoiksi olisi mahdollistanut edes niiden tietojen kutsun, vaikka ne olisivat tyhjiä. Tein myös päätöksen XML -tiedostojen rakentamisen aikana ”description” -osion tyhjäksi jättämisen, sillä tämän lukeminen ei vaikuta ohjelman toimintaan, vaan on ainoastaan käyttäjäkokemusta varten. Näiden osioiden lisääminen olisi helposti vienyt kymmeniä tunteja aikaa, joten päätin työn laadun vuoksi jättää tämän myöhemmäksi, kurssin ulkopuolelle.
Viimeinen ongelma, joka minulle tuli vastaan, oli tämänkaltainen yhdistäminen Unityn ja C# kanssa. Vaikka olen koodannut molemmilla ohjelmilla ja työskennellyt niiden kanssa, joidenkin erikoistapauksien korjaamisessa meni odotettua enemmän aikaa (TextMeshUGUI näkymätön string elementti), toinen vastaava oli C# puolella yleisen Random() -luokan toimintatapa. Myös oman koneen hajoaminen ja tutkiminen vaikeutti projektia.

Oletin esisuunnitelmassani, että koneoppimisen lisääminen vie paljon aikaa. Teoreettisesti olin oikeassa tämän suhteen, sillä en voinut lisätä sitä lyhyessä ajassa ohjelmaan. Käytännön tasolla tämä kuitenkin tapahtui pelisääntöjen monimutkaisuuksien lisäämisen ja ymmärtämisen vuoksi, sillä nämä veivät koneoppimiselle tarkoitettua aikaa. Muutoin oletin hyvän etukäteisen suunnittelun estävän muut ongelmat, mutta odottamattomia ongelmia on vaikea estää.

En ole henkilökohtaisesti onnistunut kaatamaan ohjelmaani Unityn kautta luodun buildin avulla. Kyseinen executable on Game -kansiossa, jossa on erillinen README.txt käyttöohjeita varten.

Työkalujen valinta, eli github, Unity ja C# auttoivat minua työn aikana, koska olen tottunut työskentelemään näiden kanssa aikaisempien projektien aikana. Unityn käyttäminen käyttöliittymän rakentamiseen auttoi projektissa, ja vahvisti näkemykseni sen hyödyllisyydestä 2D ohjelmiin. Sen avulla on myös mahdollista tuottaa parempia käännöksiä ohjelmaan, jonka vuoksi lopullista projektia on nopeampi testata yksin tai muille jakamalla.

Alkuperäinen esisuunnitelmani aikataulu ei pitänyt, jonka vuoksi siirsin tavoitteeni sen suhteen toisenlaiseksi. Tarkoitukseni oli esisuunnitelmassa tehdä ohjelmointityö mahdollisimman nopeasti kyseisen esisuunnitelman kirjoittamisen jälkeen, mutta Koronan, burnoutin ja yksityisten ongelmien vuoksi en pystynyt aloittamaan työtä haluamallani tahdilla. Näiden tapauksien vuoksi siirsin tavoitteeni syyskuulle ja opettelemaan pitkäaikaista päivittäistä ohjelmointia. Ainoa muutos, jonka olisin tehnyt ratkaisuuni, olisi ollut muutaman vapaapäivän pitämisen, sillä huomasin työskentelyn laatuni heikkenevän huomattavasti jatkuvan kahden ja kolmen viikon koodaamisen aikana.

Kaikilla kursseilla mainitaan esisuunnitelman tärkeydestä, sekä materiaalin valmistelusta. Ohjelmointityö vahvisti näitä teorioita, sillä tarkempi XML tiedostojen määrittely ja esitarkistus olisi nopeuttanut monien bugien korjauksessa, sillä ei olisi tarvinnut arvata mikä on pielessä ja ovatko nämä ongelmat jollain tavalla yhteydessä esimateriaaliin.

4 LÄHDEKOODI

Työssä ei ole käytetty ulkoisia lähdekoodeja. Ongelmakohtien lähestymistavoista on keskusteltu useiden muiden ohjelmoijien kanssa, mutta nämä keskustelut ovat keskittyneet ongelmasta keskusteluun, eivätkä valmiin ratkaisun implementointiin.

5 KÄYTTÖOHJE

Seuraava käyttöohje on myös Game -kansion sisäisessä README -tekstitiedostossa.

Käynnistä peli ajamalla PF2E_Character_Generator -ohjelma

Ohjelmassa on eri välilehtiä sen yläkulmassa, nimetty "Character", "Skills", "Feats", etc. Näiden välilehtien oikealla puolella on myös "Generate" -painike, joka on eri värillä. Tätä painiketta käytetään uuden hahmon luontiin.

Ainoa välilehti, joka sisältää tietoa pelin käynnistyksessä on "Character" -välilehti. Muissa välilehdissä on tyhjä lista ja vierityspalkki, jolla listaa voi selata, kun sitä raahaa ylös tai alas.
"Character" -välilehdessä on kohta mihin syottää haluttu taso generoitavalle hahmolle.
Tämä osio hyväksyy ainoastaan numeroita, ja se rajoittaa generoitavan hahmon tasojen 1 ja 20 välille.
Kun haluttu taso on syötetty, paina "Generate" -painiketta, joka on oikeassa yläkulmassa, uuden hahmon luontiin.

Kun uusi hahmo on generoitu, sen tiedot löytyvät pelin eri välilehdistä.

"Feats" -välilehti sisältää tietoja hahmon feateista, kuten:
- nimi
- millä tasolla valittu
- tyyppi
- painike sen kuvaukselle

Painike kuvauksille avaa uuden ikkunan, jolla on otsikko, joka vastaa sen nimeä. Painikkeen uudelleenpainaminen sulkee kyseisen ikkunan. Kaikki uudet ikkunat avautuvat päällekkäin. Aktiivisen painikkeen väri on kirkkaampi kuin muiden, joten käyttäjä tietää mikä ikkuna on auki.

1.0 versiossa uusissa ikkunoissa kuvaukset eivät ole kuvaavia.
1.0 versiossa "Spells" -valilehti sisältää ainoastaan featit, jotka ovat relevantteja loitsujen käyttäjille.

TL;DR:

Kirjoita haluttu hahmon taso "Character" -välilehdessä olevaan kirjoitusalueeseen, ja paina "Generate" -painiketta.

6 YLLÄPITÄJÄN TARVITSEMAT TIEDOT

XML -tiedostojen kasvatuksessa täytyy noudattaa aikaisemmin mainittua formaattia erikoismerkkien avulla: ’|’, ’-’, ’/’.

Ohjelmaa ei ole rakennettu välttämään syntaksi ongelma XML tiedostojen nimeämisessä ja ParseXML -luokassa. Oletuksena on samankaltaisen formaatin pitäminen, oikeaoppisen XML -syntaksin mukaisesti.

7 JATKOKEHITYSKOHTEET

Alla listauksena useita eri jatkokehityskohteita:

- Features -tabin muokkaus Character sivun kaltaiseksi – käyttäjälle yksinkertaisempi.
- Features -tabin languages voi kasvaa alueensa ulkopuolelle, edellä mainittu muutos tabille tuottaisi ratkaisun ongelmaan.
- Highlight menetelmä tabien vaihdoksissa ei kuvaa näkymän muutosta ”Generate” -painikkeen painalluksen aikana. Tämä voi hämmentää käyttäjiä, mutta jokaisella tabilla on otsikko, joka antaa selkeän ymmärryksen katseltavasta näkymästä.
- Valintamahdollisuudet esimerkiksi ancestryn, classin tai backgroundin valintaan generointia varten.
- Initial Feat XML -tiedostojen luominen ja luodaan niiden valinta ja vaikutukset koodiin, kovakoodamisen sijaan.
- Loitsujen valinta niitä käyttäville classeille. (Jätettiin pois valtavan tietokannan rakentamisen vuoksi).
- Mahdollinen valintamahdollisuus generointiin classin initial featin kanssa.
- Export-to-XML ohjelmaan, jotta haluttu hahmo on mahdollista saada talteen.
- Koneoppimisen lisääminen ohjelmaan, painotetun valinnan sijaan.

8 YHTEENVETO

Valmistunut ohjelma on toimiva sattumanvarainen hahmonluontiin käytettävä ratkaisu, mutta se jättää pelaajalle lopullisen tiedon paperille siirtämisen esisuunnitelman mukaisesti. Unityn kautta luotu UI ei ole kaunis, mutta se on käytännöllinen ja antaa käyttäjälle tärkeimmät tiedot ilman ylimääräisiä ongelmia. Aikataulun vuoksi ohjelmointityössä on täytynyt tehdä ratkaisuja ohjelmakoodiin, uusien XML -tiedostojen sijaan, jonka vuoksi joidenkin tiettyjen uusien elementtien lisääminen ohjelmalle vaatii ratkaisuja ohjelmakoodissa, pelkän XML -tiedoston muuttamisen sijaan. Tämänkaltaiset ongelmat ovat kuitenkin nopeasti korjattavissa, tämän projektin ulkopuolella.