HAMK
Hämeen ammatti-
korkeakoulu

OpenAI GPT-4o:n hyödyntäminen Unityn kenttäsuunnittelussa

Merkkigrafiikkakartasta 2D-pelimaisemaksi

Ammattikorkeakoulututkinnon opinnäytetyö
Tieto- ja viestintätekniikka, insinööri (AMK)
Kevät, 2025
Samppa Sirnö

HAMK
Hämeen ammatti-
korkeakoulu

Tiivistelmä

Koulutus	Ammattikorkeakoulun opinnäytetyö, tieto- ja viestintätekniikka	
Tekijä	Samppa Sirnö	Vuosi 2025
Työn nimi	OpenAI GPT-4o:n hyväksikäyttö Unityn kenttäsuunnittelussa. Merkkigrafiikkakartasta 2D-pelimaisemaksi.	
Ohjaaja	Petri Kuittinen	

Tämän opinnäytetyön tavoitteena on toimia oppaana OpeanAI-tekoälypalvelun API-liittymän käyttöön Unity-pelimoottorin kanssa peliohjelmoinnissa ja kenttäsuunnittelussa. Tätä varten tehtiin Python-koodilla kenttägeneraattori, joka tuottaa merkkigrafiikalla pelikenttiä yksinkertaista koepeliä varten. Tutkimuksen ydin oli siinä, että suuria kielimalleja ei ole suunniteltu symboliseen grafiikkaan, kuten juuri karttojen tekemiseen, mihin se tässä tutkimuksessa valjastettiin.

Työn raportissa käydään lävitse mitä tällaisessa ohjelmoinnissa on otettava huomioon. Tuloksena on, että laajat kielimallit pystyvät hyvätasoiseen symboliseen grafiikkaan, mutta niitä on siinä opastettava hyvin yksityiskohtaisesti. Kyse on eräänlaisesta selväkielisestä ohjelmoinnista (englanniksi). Tämä ohjelmointi on lyhyempää ja helpompaa kuin tavanomaisilla ohjelmointikielillä mutta myös virheherkkää. Tuloksena oli, ettei tämäntyyppistä ohjelmointia kannata tehdä yksittäiseen kenttään, mutta jos halutaan tehdä kymmeniä kenttiä, on tämä keino erittäin tehokas. Kenttä on ennen kaikkea tietoisesti suunniteltu, ei sattumanvaraisesti kuten usein proseduaalisella, käskyihin perustavalla ohjelmoinnilla tehdyt kentät ovat.

Tutkimukseen kuului myös hyvin pieni haastattelututkimus. Tässä testattiin, onko pelikentät helposti erotettavissa ihmisen ja koneen suunnittelemiksi. Tulos oli, että ei. Tämä oli siis eräänlainen Turingin testi tutkittaessa osaako ihminen erottaa koneen ja ihmisen luovan työn tulokset.

Suurin osa opinnäytetyöstä koostui itse pelin ohjelmoinnista ja grafiikan suunnittelusta, sekä oikeanlaisen kehotteen muokkaamisesta kenttägeneraattoria varten. Peli on ladattavissa (18.3.2025) osoitteesta https://github.com/Samppa72/Lopputyosirno/releases/tag/lopputyo_Sirno.

Ohjelmapaketin nimi on Lopputyo_Sirno.zip.

Avainsanat	Peliohjelmointi, pelisuunnittelu, suuret kielimallit, tekoäly, tietokonegrafiikka
Sivut	24 sivua ja liitteitä 13 sivua

HAMK
Hämeen ammatti-
korkeakoulu

Abstract

DP	University of applied sciences, Information and Communication Technology
Author	Samppa Sirnö
Subject	The use of OpenAI 4o-model in Unity level design. From textual graphics to 2D-level environment.
Supervisor	Petri Kuittinen

Year 2025

The goal of this thesis is to serve as a guide to using the OpenAI artificial intelligence service API interface with Unity game engine for game programming and level design. For this purpose, a level generator was created using Python code, which generates game levels with character graphics for a simple test game. The core of the research lay in the fact that large language models (LLM) are not designed for symbolic graphics, such as creating maps, which is what this research tried to accomplish.

The report discusses what needs to be considered in such programming. The result was that large language models are capable for good quality symbolic graphics, but they need to be guided in detail for this. It is a type of natural language programming (in English). This programming is shorter and easier than when using conventional programming languages, but it is also more prone to errors. The result was that this type of programming is not advisable for creating a single level, but if tens of levels are to be made, this method is extremely efficient. The level is consciously designed, not randomly made, as is often the case with procedurally generated levels.

The research also included a very small interview study. This tested whether game levels can easily be distinguished as designed by a human or as designed by a machine. The result was that they could not. This was essentially a kind of Turing-test to investigate whether humans can distinguish between the creative work of a machine and a human.

Most of the thesis consists of programming the game itself and designing the graphics, as well as editing the right kind of prompt for the field generator. The game is available for download (March 18, 2025) from https://github.com/Samppa72/Lopputyosirno/releases/tag/lopputyo_Sirno.

The name of the program package is Lopputyo_Sirno.zip.

Keywords	Artificial Intelligence, computer graphics, game programming, game design, Large Language Models
Pages	24 pages and appendices 13 pages

Sisällys

Kuvat

Liitteet

Liite 1. Kyselyn kysymykset

Liite 2. Ohjelman tekniset vaatimukset

Liite 3. Kenttägeneroijan käytön ohje

Liite 4. Pelin lataaminen

Liite 5. Ohjeet pelikenttiin ja pelin pelaamiseen ylipäänsä

Liite 6. Maisemanluonnin pääluuppi C#-koodilla

1 Johdanto

Ajatus paloista rakentuvasta algoritmin muodostamasta kentästä on vanha, 1980-luvulta. Tekoälyn kehittyminen harppauksittain viime vuosina on puolestaan tehnyt sillä tehdystä grafiikasta tavanomaista. Merkkigrafiikkapohjainen kartta esimerkiksi roolipelin pohjana taas oli yleistä vielä 1990-luvulla.

Tässä raportissa kuvataan yksinkertaista peliä, joka yhdistää nämä kaksi: merkkipohjaisella **Large Language Modelilla (LLM)** eli suurella kielimallilla muodostetaan merkkigrafiikkakartta, josta sitten rakennetaan **proseduaalisesti** eli käskyillä valmiista grafiikkapaloista kenttä. Tätä pidettiin vielä äskettäin melko vaikeana toteuttaa, koska suuret kielimallit ovat heikkoja havainnoimaan avaruudellisia tiloja ja kentistä tuli esim. **OpenAI**:n LLM:a käytettäessä usein miten käyttökelvottomia.

Tässä opinnäytetyössä yritetään todistaa päinvastaista. Tutkimuksen tulos oli, etteivät LLM:t ole huonoja ymmärtämään avaruudellisia tiloja, vaan niille pitää antaa tarpeeksi yksityiskohtainen ohje, että ne pystyvät tuottamaan vaikkapa hyvän roolipelin kentän. Jos niille sanoo: ”luo kenttä”, niin kielimallit eivät ymmärrä mitä niiltä halutaan. Tätä kokeiltiin käytännössä kenttien luomisessa.

Kysymyksiä, jotka tässä tutkimuksessa pyritään selvittämään ovat siis: ”onko GPT-4o kykenevä tuottamaan pelillisesti toimivia karttoja merkkigrafiikalla?”, ”pystyykö näin tekoälyllä tuotetut kentät erottamaan ihmisen tekemistä?” Tässä yhteydessä tehtävän ohjelman tekemän kartan luontiin tarvittiin käskysarja selkoenglannilla. Tämän käskysarjan luonti oli keskeinen osa tutkimusta. OpenAI:ta kutsutaan tätä käskyä käyttäen. Tutkimuksessa tämä **Python**-pohjainen **OpenAI API**:n kutsuminen on integroitu **Unity-pelimoottorilla** ja **C#:**lla tehtyyn yksinkertaiseen etsintäpeliin. Kenttiä voidaan luoda OpenAI:n API-keylla eli avaimella. Kenttägeneraattori tekee .txt-tyyppisen tiedoston, jota voi sitten pelissä pelata.

Näin on käännetty ympäri proseduaalinen ohjelmointi. Ihminen ei tee kenttäsuunnittelua vaan vastaa yksityiskohdista. Itse kenttien rakenteen luo LLM. Kysymys on, että jos ajatusta viedään eteenpäin, tarvitaanko ihmistä välttämättä edes laadun tarkastukseen.

Tässä tutkimuksessa siis kysyttiin, poistuuko ihmisen osuus kokonaan. Kokeiltiin kunnon kartan luomista ja siihen tarvittiin monta yksityiskohtaista englanninkielistä käskyä. Tulos

oli, että kyse on eräänlaisesta ohjelmoinnin tyypistä. Kysymys oli syrjäyttääkö AI ihmisen, vai helpottaako se vain karttojen sarjatuotantoa.

Tässä raportissa käytiin myös yleisesti 2D-grafiikan luontia Unity 6:lla ja tehtiin käyttäjätesti. Testiotos on todella pieni, vain neljä osanottajaa, mutta toisaalta se mahdollistaa yksityiskohtaisemman palautteen. Isoin kysymys testissä oli: erottaako testihenkilö koneen tekemät kentät ihmisen suunnittelemista?

Koska tämä opinnäytetyö on tehty suomeksi, on pyritty pysymään myös suomen kielessä eli välttämään pahimpia ”finglish”-ilmaisuja. Tässä oli suurena apuna Janne Joensuun opinnäytetyö 3D-alalla käytettävästä sanastosta (Joensuu, 2016).

2 OpenAI:n käyttö pelisuunnittelussa opinnäytetyönä

2.1 Tämän opinnäytetyön idea

Tämän opinnäytetyön idea on kokeilla mm. **ChatGPT** -tekoälypalvelussa käytetyn OpenAI-tekoälymoottorin **GPT-4o**-version käyttöä roolipelikartan luomiseen Unity-pelimoottorin maiseman luomiseen. Tarkoitus on, että tämä on eräänlainen opas tämän tyyppistä tekniikkaa käyttäville.

Suuria kielimalleja on pidetty huonoina ymmärtämään ja tuottamaan symbolista grafiikkaa, kuten tässä työssä karttoja. Niitähän ei ole varsinaisesti koulutettu tähän. Esimerkkinä keskustelusta LLM:ien kyvystä tehdä kaavakuvia on tuore tutkimus, jossa pyrittiin arvioimaan testillä, pystyvätkö LLM-mallit ymmärtämään ja kuvittelemaan visuaalisia objekteja siis kuvien semanttista sisältöä. (Qui ym., 2024, ss.1-4).

Tässä kiinalais-amerikkalais-eurooppalaisessa tutkimuksessa testattiin suurten kielimallien kykyä ymmärtää esimerkiksi SVG-vektorigrafiikkaa. Tässä tutkimuksessa todettiin, että LLM-mallit epäonnistuivat tässä tehtävässä dramaattisesti, vaikka se olit ihmisille helppo ymmärtää (Qui ym., 2024, s.10).

Tässä käsillä olevassa tutkimuksessa symbolisessa grafiikassa kyse on 2D-kartasta. Tässä harkittiin myös 3D-kartan tekemistä, mutta koska pohjana käytettävä tekstikartta on kaksiulotteinen, katsottiin että se demonstroituu paremmin 2D-maisemana Unity-pelimoottorissa. Unityssahan on valmiina työskentelytila 2D-pelien tekemiseen. Sinänsä

3D-kartan luominen onnistuisi esimerkiksi luomalla kaksi karttaa toinen korkeudelle ja toinen sisällölle, tässä halutaan kuitenkin pysyä mahdollisimman yksinkertaisessa ohjelmassa.

Idea tekstigrafiikalla tehtyjen karttojen käyttöön juontaa jonnekin 1900-luvulle, jolloin sillä tehtyjä pelejä oli paljon. Tämä tuodaan nykyaikaan tulkitsemalla se valmiiksi tehdyiksi komponenteiksi. Tämän tyyppistä suunnittelua on esimerkiksi monissa lentosimulaattoreissa, joissa maisema koostuu valmiista ”tiilistä” kootusta mosaiikista. Esimerkiksi ainakin vanhemmissa Il-2 Sturmovik -lentosotapeleissä maisemat oli luotu valmiista vaihdelluista tiilistä, joissa oli peltoa ja metsää ja niihin lisätyistä tarkemmista kohteista, jotka olivat ainutlaatuisia oikeitten karttojen perusteella tehtyjä kohteita taloineen kaikkineen (vaikkapa Stalingradin kaupunki): ”In Il-2 the ground textures are divided into square tiles” [”Il-2-pelissä maakamaran tekstuurit ovat jaettu neliötiiliin”], kuten eräs pelaaja toteaa pelin foorumilla (BlackHellHound1, 2016).

Näitä on myös tutkittu Suomessakin esimerkiksi vuonna 2024 tehdyssä opinnäytetyössä (Särkiniemi, 2024). Tässä ei siis käydä läpi sinänsä proseduaalista pelikenttien kehitystä muuta kuin miten se toteutuu tekoälyn suunnitteleman kentän toteuttamisessa. Proseduaalista kenttäsuunnittelua on tehty lukemattomia kertoja sinänsä. Esimerkkeinä voidaan mainita vaikka avaruusseikkailu Elite 1980-luvulta jatko-osineen sekä tuoreempana No Man’s Sky, myös avaruusseikkailupeli, jossa on lähes rajaton määrä maailmoita. Näitä ei olisi olemassa ilman proseduaalista kenttäsuunnittelua. (Särkiniemi, 2024, s.7).

Klassisin esimerkki merkkigrafiikalla tehdystä, satunnaisuuteen perustuvasta pelistä on Rogue vuodelta 1980. Siinä merkit symbolisoivat luolastoja, joista pelaajan täytyi löytää ulos. (Särkiniemi, 2024, ss.10-12). Tässä tutkimuksessa toteutetun pelin idea on hyvin samankaltainen, löytää lohikäärme avoimesta kentästä. Grafiikka on pohjaltaan kartta, joka on merkkigrafiikkaa, josta generoidaan sitten pelaajalle näkyviä 2D-grafiikkakenttiä.

Se mikä tekee näin luoduista kentistä erilaisia verrattuna vanhoihin proseduaalisesti luotuihin pelikenttiin on sattuman osuuden minimointi. Tekoäly luo ja ”ajattelee” luodessaan kenttiä. Tämän vuoksi kenttien laatu voi olla erittäin hyvä. Eli tämän työn idea eroaa näistä edellä mainituista peleistä siinä, että karttojen tekijänä ei ole (ainakaan suoraan) ihminen tai satunnaislukugeneraattori (jota siis ei ole tässä käytetty maiseman monipuolisuuden takaamiseksi) vaan tekoäly ja nimenomaan tekstien tuottamiseen ei symbolisten grafiikkojen tuottamiseen koulutettu tekoälyversio.

Kuten Särkiniemi toteaa, on proseduaalisessa ohjelmoinnissa ollut keskeistä sattumanvaraisuus (Särkiniemi, 2024, s.3). Tässä tutkimuksessa ideana on nimenomaan päästä eroon sattumanvaraisuudesta (paitsi tietenkin siinä määrin kuin tekoäly siihen perustuu) ja ottaa tilalle **intentionaalisen** rakennelman, jossa voisi näkyä älyn suunnitelmallisuutta.

2.2 Aiheen merkitys, tutkimuksen tavoite ja rajaus

Merkitys tässä tutkimuksessa löytyy käytännöstä, eli siitä miten merkkikartoista voidaan luoda maisemia, jotka ovat pelattavia, ei pelkästään katsottavissa. Siinä tutkitaan mitä OpenAI:n ja Unityn yhteensovittaminen vaatii, mitä se käytännön ohjelmoinnissa on ja mitkä ovat sen rajoitukset. Ensimmäinen kysymys on, onko tällainen ChatGPT:n tapainen tekoäly ylipäätänsä sovelias grafiikkakenttien tuottamiseen.

Tutkimuksen kohteena on kaksi asiaa, eli merkkigrafiikkakartan luominen LLM-ohjelmalla, joka ei ole varsinaisesti tarkoitettu grafiikan tekoon, mutta joka ymmärtää paremmin käskyjä ja pystyy tuottamaan kartan, jonka voi käyttää pelikentän luomiseen helpommin kuin valmiin grafiikan. Toinen asia mikä tutkitaan, onkin sitten kentän luominen tällaisesta kartasta ja se miten hyvä tällä tavoin luotu kenttä on. Tätä varten järjestetään pieni testi, jossa on neljä koehenkilöä, joilta yritetään saada selville erottavatko he koneen tekemät kartat ihmisen suunnittelemista ja ovatko koneen luomat kartat ylipäänsä kelvollisia pelikenttien perustoja.

Näitä operaatioita varten on luotu peli, Lohikäärmeen tappaja, jossa on ideana viiden minuutin sisällä löytää kentästä lohikäärmeen luola. Pelaajalla on tieto etäisyydestä tähän, mutta itse kartassa on usein tilanne, jossa ei tiedetä miten vuorista ja muista osista koostuvia esteitä, jotka estävän suoran pääsyn päämäärään, on kenttään sijoitettu. Se mikä on lähellä, voi vaatia pitkän kiertotien. Halutaankin tietään ovatko koneen luomiin karttoihin perustuvat pelit mielenkiintoisia, tarpeeksi vaikeita, ylipäänsä mahdollisia läpäistä ja ehjiä karttoja, joissa ei ole virheitä ja peli on toimiva, esimerkiksi sankari pystyy liikkumaan kentällä eikä ole eksynyt vuoristoon tai lohikäärme ei ole vuorien saartama eli mahdoton tavoittaa. Kysymys on siis siitä, pystyykö OpenAI-tekoälypalvelun LLM GPT-4o -malli luomaan kentän, kuten sen olisi luonut ihminen selkokielellä annetusta käskystä.

Tämän työn tavoite on myös havainnollistaa C#:in, Unity 6:n ja tekoälyn käyttöä. Siitä selviää mitä proseduaalinen ohjelmointi, merkkitiedostojen käyttö grafiikan luonnissa, 2D-grafiikan luonti ym. seikat tarkoittavat, eli asioita, joita tässä raportissa läpi käydään.

Työn tulos selviää lataamalla valmiiksi käännetty versio Lohikäärmeen tappaja -pelistä (katso liite 2) ja sen testaamisella. Mukaan ei laiteta alkuperäistä lähdemateriaalia, jonka laajuus on noin 7 gigatavua, vain Pythonilla ohjelmoitu kartanluontiohjelma on kokonaisuudessaan *zip*-paketin mukana **Kartta**-kansiossa. Peli on ladattavissa osoitteesta https://github.com/Samppa72/Lopputyosirno/releases/tag/lopputyo_Sirno. Lopputyo_Sirno.zip tiedostona.

Halutessaan lukija voi kokeilla itse luoda tällä Python-ohjelmalla karttoja, ohjeet ovat mukana. Se vaatii kuitenkin tiettyä perehtymistä ja luottokortin. Se vaatii OpenAI:n API-avaimen ostoa ja syöttämistä kartta.py-ohjelmaan, joko suoraan tai API:n ympäristömuuttujana. Ohjelman voi aukaista pelissä painamalla Alt-M näppäinyhdistelmän.

Suositeltavaa on käyttää paketin mukana tulevia n. 60:tä karttaa Kartta-kansiossa, jotka lähes kaikki on luotu kartta.py-ohjelman eri versioilla. Niiden käyttö ei maksa mitään ja niitä voi vaihtaa painamalla Shift-R ja syöttämällä kartan nimen ilman .txt -loppupäätettä syöttökenttään.

Tämä selitetään siksi, että lukija ymmärtää, että OpenAI:n API:n käyttäminen peleissä ei ole ongelmatonta. Vaikka tekijä olisi laittanut omista varoistaan hankkimansa avaimen mukaan peliin, kuten alun perin oli suunnitelma ja joka olisi huomattavasti helpompaa, ei OpenAI tähän olisi suostunut, vaan sulkenut tämän avaimen. **OpenAI:n avaimia ei saa luovuttaa muille,** organisaation avaimet ovat sitten eri asia.

Alkuperäisen materiaalin koon vuoksi, sekä sen vuoksi, että se sisältää **copyright**-suojattua materiaalia, joihin on ostettu oikeudet, ei sitä aleta jakamaan kokonaisuudessaan. Jos olet kiinnostunut jostakin yksittäisestä asiasta, joka ei tässä raportissa ilmene, esimerkiksi miten sankarin liikkuminen on koodillisesti toteutettu, toimitan materiaalin, kuten C#-koodin, mielelläni. (Sähköpostiosoite tätä varten: samppa.sirno@gmail.com).

2.3 Käytetyt tutkimusmetodit

Tämän raportin tehtävä on **demonstroida** mitä kaikkea tällaisen projektin luominen vaatii. Se on kuvaus tästä luontiprosessista, joka toivottavasti myös auttaa niitä, jotka työskentelevät samanlaisten ohjelmointiongelmien kanssa. Tutkimusmetodi on siis ennen kaikkea **deskriptiivinen** eli kuvaileva. Toivottavaa on, että tämä raportti toimii jonkinlaisena ohjekirjana LLM-malleilla luotujen karttojen tekemiseen ja niiden käyttöön vaikkapa roolipelin pohjaksi.

Tässä ohjelma jakautuu kahteen osaan: kenttien suunnitteluun tarkoitettuun Python-kieliseen kenttägeneraattoriin ja C#-kielellä ja Unitylla toteutettuun pääohjelmaan, joka tulkitsee kartat. Molempien ohjelmien tärkeimmät koodin osat ovat luettavissa tässä raportissa (kuva 1 ja liite 6).

Lisäksi käydään läpi itse komentokäsky, jolla AI osaa luoda halutun laisen kentän, sekä se mitä vaikeuksia tällaisen käskyn luomisessa on ja mitä siinä otettava huomioon. Käskyn listaus löytyy kuvasta 2 (englanniksi) ja 3 (suomennettuna).

Erikseen on sitten toteutettu pieni kvalitatiivinen kysely, johon palataan. Tämä on eräänlainen Turingin-testin pienoisversio, siinä kysytään koehenkilöiltä, onko kenttä ihmisen vai tekoälyn suunnittelema. Tai mikä on mielikenttä kuudesta koekentästä, joista kaksi on ihmisen, loput AI:n suunnittelemia. Tämän lisäksi koehenkilöt saivat vapaasti kertoa muita vaikutelmia pelistä.

Kyselyn kysymykset löytyvät liitteestä 1. Ohjeet itse ohjelman lataamiseen ja käyttöön, sekä järjestelmävaatimuksiin löytyvät liitteistä 4,5 ja 2.

3 2D-pelien kenttäsuunnittelu Unity-pelimoottorilla ja OpenAI:lla

3.1 Unity-ohjelmoinnin yleiset teoriat ja periaatteet

Tässä yhteydessä ei käydä lävitse yleistä pelien tekoa Unityllä, siis 2D ja 3D-peleille yhteisiä työvaiheita. Ei käydä esimerkiksi skenen, eli pelikentän käsitettä, ei pelissä liikkumista, ei kameroita tai valaisua tai käännösprosessia valmiiksi peliksi.

Aihepiiristä on lukuisia hyviä kirjoja, oma suosikkini on Jukka Selinin kaksiosainen oppimateriaali Unity-ohjelmointikursseille (Selin, 2024). Siinä on selkeällä suomen kielellä ja kuvin sekä koodiesimerkein opastettu Unity-ohjelmointiin.

3.2 2D-pelien suunnittelun poikkeaminen muista pelityypeistä Unityssa

> The biggest difference, I've noticed, 3D projects import textures just as textures. 2D projects, any imported texture is immediately converted into a Sprite.
>
> Other than that, they're almost entirely interchangeable. I'm working in 3D to make a 2D game.
>
> [Suurin ero, olen huomannut, on että 3D projektit importoivat tekstuurit pelkkinä tekstuureina. 2D projektit sen sijaan muuttavat sen tekstuurit välittömästi spriteiksi.
>
> Muuten ne ovat käytännössä samanlaisia. Työskentelen parasta aikaa 3D-tilassa tehdäkseni 2D-pelin.]
>
> (Deeeds, 2018).

Kuten nimimerkki Deeeds toteaa, Unityssa on siis valmiina 2D-pelien tekoon suunniteltu tila. Itse asiassa tämä on vain muunneltu 3D-tila (vrt. myös. Selin, 2024, ss.176-177), 2D-ohjelmoinnista Unitylla: (Unity Technologies, 2024). Perspektiiviä ei ole vaan suora ja pysyvä kuvakulma. Tehtäessä lohikäärmettä käytettiin pohjana valmista 3D-**prefabia** (käyttövalmiiksi tehtyä animaatiomallia), joten siinä on hiukan nähtävissä kolmiulotteisuutta ikään kuin efektinä. Peli itsessään on kuitenkin 2D-peli. Toki esimerkiksi pelaajahahmon on animoitu spritien (**sprite**, hahmomalli 2D-peleissä) pohjalta kahdeksaan eri suuntiin kävelevänä, mikä nyt on tietysti normaalia 2D-pelisuunnittelussakin ja hahmo pystyy myös kävelemään myös puiden latvojen takaa kuitenkin törmäten niiden runkoon.

2D-peleissä on siis ”lattea” **ortograafinen perspektiivi**. Samoin koska hahmot ovat lähes kaikki samalla tasolla, niiden järjestys hierarkiassa (**order in layer**) on tärkeä (Selin, 2024, s.177). Tämä takaa sen, että hahmo ei voi kävellä järven lävitse ja että se menee toisaalta puiden takaa. Samoin esimerkiksi lohikäärmeen pesä koostuu kolmesta osasta, jo mainitusta vuori-spritesta, sekä 3D-prefabista, joka on kultarahapino sekä lohikäärmeen

prefabista, jossa lohikäärme tekee erilaisia liikkeitä, herää, lentää hiukan ja menee nukkumaan (ja lopulta kun sankari saapuu paikalle, kuolee).

Taustalla on siis nurmikkotaso, jonka päällä on spriteistä koostuvat ”levy” ja lohikäärmeen tapauksessa aarre ja lohikäärmeen prefab. Näiden asettelu, valaistus, valmiiden animaatioiden käsittely (lohikäärmeen kuolema), osin animaatioiden teko valmiista spriteistä (sade) ym. jäävät valmiiksi ostettujen palojen käyttäjälle eli pelin tekijälle.

Hahmojen fysiikan hoitavat Rigidbody2D ja Collider2D (BoxCollider2D ym.) -fysiikkamallit. Ne huolehtivat siitä, että toisiinsa törmätessä 2D-spritet eivät mene vain toistensa lävitse vaan voivat pysähtyä niihin, työntää niitä eteenpäin, kaataa niitä ja jos painovoima on voimassa tippuvat ne alas. Toiminnot ovat hyvin samanlaisia kuin 3D-elementeissä tietysti vain kaksiulotteisina. (Selin, 2024, s.194).

Esimerkkinä tästä pelissä sankarihahmo törmää pensaaseen, jonka se huitaisee mennessään pois edestä. Puut on tehty kahdesta spritesta ja **colliderista**, törmäyssensorista, runko ei anna periksi, eikä hahmo pääse sen läpi, latva taas peittää hahmon lehtien taakse, mutta sen läpi voi kävellä. Kuten sanottua näistä puista ja pensaista tehtiin 40 yksikköä kertaa 40 yksikköä kokoinen laatta, jota monistettiin kartan näyttäessä ”F”, eli forest, eli suomeksi metsä. Kartan luontihan tapahtui englannin kielellä.

Painovoima on säädetty sopivaksi ja samoin esineitten painot siihen nähden realistisiksi. Painovoima ei siis vedä objekteja alaspäin, jos näin tehdään puut, järvet, kylät ja muut tippuvat yhtenä kasana kuvaruudulla alas ruudulta. Ei ole siis käytetty painovoimaa vaan **static**-määritettä, tehden ne liikkumattomaksi tai niin painaviksi, etteivät elementit pensaita ja kukkia lukuun ottamatta heilu (Selin, 2024, s. 195). Puut saisi kyllä näyttävästi kaatumaan tai raahautumaan hahmon mukana. Niiden massa on kuitenkin niin suuri, etteivät ne törmätessä heilu. Hahmon liikkuminen on tehty määräämällä sille vakiovoima, joka on riittävä raahaamaan pensaita mutta ei puita, taloja, järviä tai vuoria mukanaan. Näitä kaikkia ominaisuuksia voi kokeilla ja saada aikaan pienellä vaivalla näyttäviäkin 2D-fysiikkaefektejä.

Elementeistä koostuva kenttä koostuu siis 2D-spriteista, jotka on ostettu valmiina tai tehty tekoälyn avulla. Tämä on yksi syy minkä takia peli lähdemateriaali ei ole ladattuna **Githubiin**. Alkuperäinen materiaali Unityssa on kooltaan yli 7 gigatavua. Peli itse on valmiina käännettynä vain 500 megatavua ja pakattuna 70 megatavua.

3.3 Unityn ohjelmointivälineet kenttäsuunnittelussa ja tämän pelin kenttäsuunnittelu

Tässä pelissä ohjelman osuus lataa tiedostosta kaksiulotteisen taulukon, jossa eri elementit ovat. Tätä on hiukan jouduttu parantamaan niin, että kartan oikea laita ja alalaita tulevat myös vaaleanpunaisen katkoviivan rajaamiksi (kuva 17). Tässä on käytetty moninkertaista varmistusta. Ensin käsketään AI:n luomaan 40 kertaa 40 kokoinen kenttä, jonka laidat ovat ulkovallin (”O”), ympäröimät. Koska tekoäly ei aina tai lähes koskaan tee tasasivuista kenttää, on pidettävä huoli siitä, että myös lohikäärmeen pesä (”D”) tulee kentän sisälle. Jos siitä huolimatta ei ole tasaneliöinen kenttä käsketään täyttämään ylimääräisiä muurinpaloja ruoholla (”G”) tai metsällä (”F”) ja sitten tekemään ulkomuurin. Ja kentän alareunaan 40 kertaa ”O” -merkkinen rivi.

Näin periaatteessa.

Käytännössä kenttä on silti harvoin neliön muotoinen ja joskus jopa puolta lyhyempi. En ymmärtänyt tätä logiikkaa, kuten ei ilmeisesti tekoälykään. Niinpä jouduttiin vielä Unitylla tehdyssä C#-ohjelmassa erikseen lisäämään kentällä oikean ja alareunan. Tämä yleensä jo auttoi, mutta ei aina. Joissakin kentissä ulkomuurissa oli aukkoja. Niistä jopa pääsi läpi. Kenttä vain on niin suuri, ettei sankarihahmo päässyt ainakaan testaukseni mukaan tippumaan ruohikolta viiden minuutin aikana.

Kaikkien elementtien alkuperäiset mallit itse sankaria lukuun ottamatta ovat ulkomuurin vasemman alareunan kulmauksesta alaspäin. Kukin mallina käytetty spriteistä koostuva mallilaatikko vaati siis 40 kertaa 40 yksikköä. Näitä laatikoita, joista kenttä koostuu, on siis kuusi erilaista. Kun ne olivat allekkain, oli helppo (ainakin periaatteessa) laskea niiden asema ruohokentällä X- ja Y-koordinaateilla sen mukaan missä kohtaa taulukkoa ne olivat.

Myös lohikäärmeen pesä oli tällainen hahmo. Valitettavasti tekoäly ei sitä läheskään kaikkiin kenttiin osannut sijoittaa. Tämän takia C#-koodi käy läpi kenttämuuttujan ja jos se ei löydä ”D”-merkkiä, se lisää oikeaan alaneljännekseen satunnaisesti tämän merkin, mikäli se osuu ruoho- tai metsä- merkin kohdalle. Pienten korjailujen jälkeen harvoin 60:stä kentästä jos koskaan, peli oli mahdoton, eli lohikäärme olisi ollut vuorten täysin ympäröimä. Tämä huomioitiin Python-koodissa, jossa käskettiin ensinnäkin ruoho- ja metsäkenttien olemaan aina toisiinsa yhteydessä ja myöhemmissä versioissa tässä englanninkielisessä käskyssä myös olemaan muodostamatta ”laaksoja” vuoriston keskelle.

Kentän sankari sijoitettiin samalla periaatteella myös satunnaisesti ruoho- tai metsäneliöön kentän vasempaan yläneljännekseen. Koska hahmo oli alun perin eri kohdassa kuin muut spritet ja siitä oli vain yksi instanssi, joka sijoitettiin valmiiseen kenttään, hahmo välillä sijoittui selvästi vuoristoon, josta se kuitenkin ”valui” fysiikkamoottorin mukaisesti ruohikolle. Tätä kohtaa joutui hiomaan paljon. Kuten sanottu sekä sankarihahmo, että lohikäärme olivat satunnaisesti sijoitettuja, mutta käytössä oli **siemenluku**. Näin lohikäärme ja sankari sijoittuivat samalla kentällä samaan kohtaan.

Joissakin tapauksissa kentän muodostaminen epäonnistui täysin, jolloin peli antaa virheilmoituksen ”Viallinen kenttä” tai joissakin tapauksissa kenttä oli mahdoton. Jos kaupallista versiota jostakin pelistä tällä tekniikalla aiotaan tehdä, niin kenttien täytyy olla etukäteen joko ihmisen tai algoritmin tarkastamia. Tämä vie pohjaa pois ajatuksesta automaattisesta kentän luonnista pelin yhteydessä.

Tekoälyähän voidaan luokitella juuri tämän ominaisuuden, eli ihmisen roolin mukaan. **Vahva tekoäly** ei tarvitse ihmisen puuttumista, kun taas **heikko tekoäly** tarvitsee käyttäjän toimintaa, jotta se toimii. Tässä yhteydessä on siis tarkoitus tehdä mahdollisimman vahva tekoäly eli sellainen, joka ei vaadi interventiota suorituksen aikana muuta kuin pelin käyttäjän eli eräällä tavalla asiakkaan roolissa. (Heiskanen, 2021, s.4). Yleisesti ottaen kentät kuitenkin toimivat.

3.4 OpenAI ja sen ohjeistaminen sekä muut tekoälyohjelmat

Tässä tutkimuksessa kokeiltiin myös claude.ai, https://gemini.google.com/, google.ai.studio sivuja ja tutkittiin LLama Darkidol -AI:a. Esimerkiksi **Claude** tuotti lähes yhtä hyvän kartan kuin ChatGPT. Se selvästi kuitenkin ymmärsi käskyn ja pystyi jopa luomaan esimerkiksi vuoristoryhmiä jne.

Google Gemini ei joko ymmärtänyt englanninkielistä käskyä eikä kysymystä ylipäänsä tai tuotti käyttökelvottoman kartan, kuten pelkästään ruohokentän, joka oli ulkomuurin ympäröimä. Sillä siis ei pystytty luomaan minkäänlaista karttaa.

Google.ai.studiossa käskyn täytyy olla erilainen kuin OpenAI:n käyttämä. Se tuotti juuri sellaisen käyttökelvottoman kartan, jollaisia kaikkien LLM:ien tekemien karttojen on väitetty olevan.

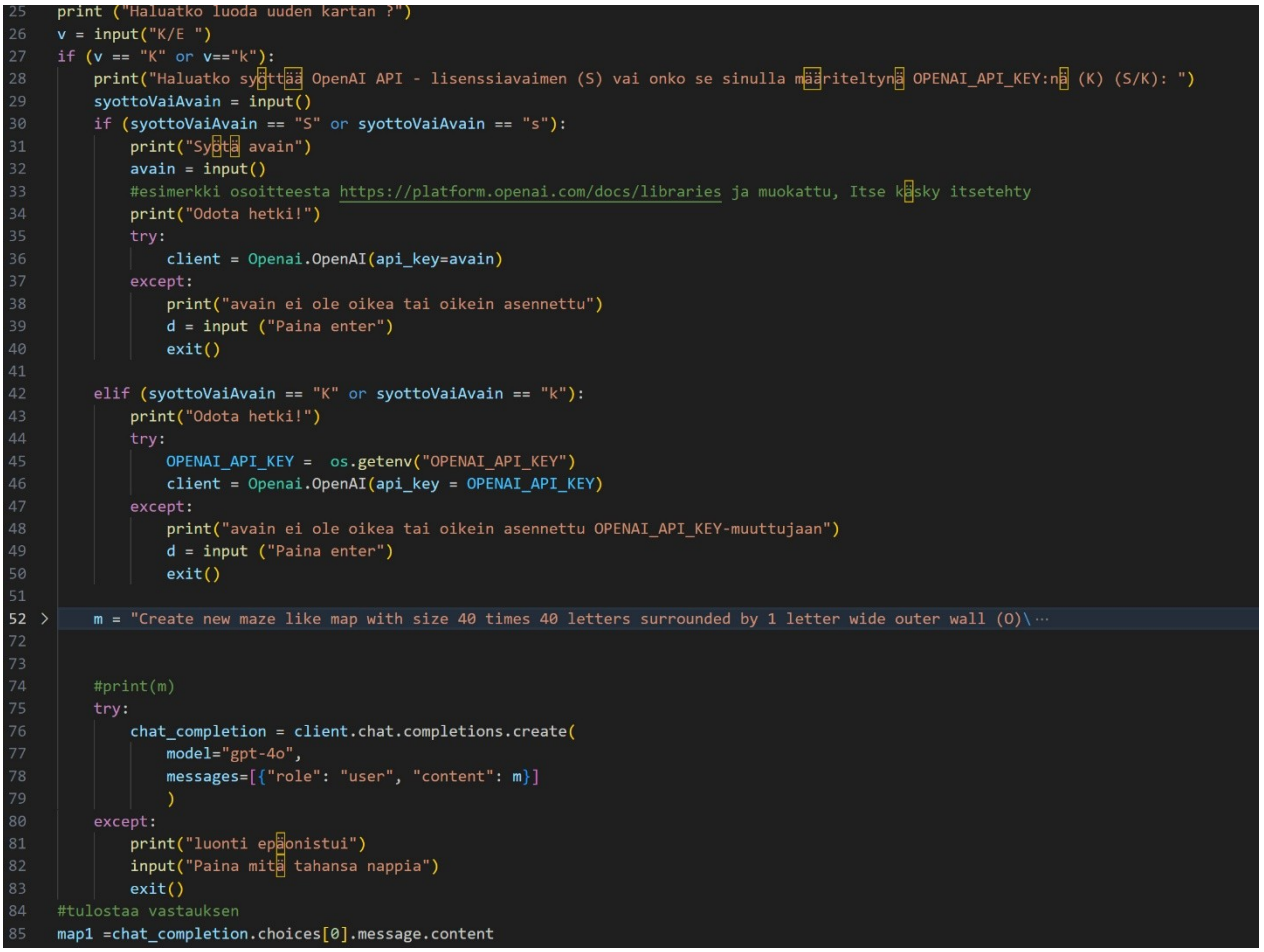
LLama Darkidol AI olisi ollut taas oma projektinsa vaiheessa, jossa oli jo sitouduttu GPT-4o:n käyttöön, sinänsä se olisi varmasti ollut hyvä itse pelikoodin osien tekemiseen. Se onkin käyttökelpoinen itse pelin koodauksessa, eli hieman eri asiassa kuin tässä tutkimuksessa. Päätettiin siis pysyä OpenAI-palvelussa. Toki näitä muitakin malleja olisi voinut tutkia syvemmin.

Tämän tutkimuksen lopuksi kokeiltiin myös o3- ja o4-mini versioita OpenAI:n malleista. Näitä kutsuttiin käskyillä model = ”o3-mini” ja model = ”o4-mini” kartta.py -ohjelmassa model = ”gpt-4o” käskyn sijasta. Molemmat tuottivat kartan, tosin usean minuutin jälkeen, kun 4o tuotti sen parissa kymmenessä sekunnissa. Näistä o3-mallin tuottama kartta oli käyttökelvoton, se jumitti Unitylla tehdyn pääohjelman. Sen sijaan o4-minillä tehty kartta oli täysin käyttökelpoinen ja jopa mielenkiintoinen.

Tässä tutkimuksessa käytetyn kenttägeneraattorin käytön ohjeet löytyvät liitteestä 3. Seuraavassa käydään läpi sen toteutus Python-kielellä kartta.py-ohjelmassa Kartta-kansion alla.

Teknisesti käsky OpenAI:lle annetaan seuraavan Python-koodin mukaan.

Kuva 1: Python-kielellä koodattu karttageneraattorin koodin osa



```
print ("Haluatko luoda uuden kartan ?")
v = input("K/E ")
if (v == "K" or v=="k"):
    print("Haluatko syöttää OpenAI API - lisenssiavaimen (S) vai onko se sinulla määriteltynä OPENAI_API_KEY:nä (K) (S/K): ")
    syottoVaiAvain = input()
    if (syottoVaiAvain == "S" or syottoVaiAvain == "s"):
        print("Syötä avain")
        avain = input()
        #esimerkki osoitteesta https://platform.openai.com/docs/libraries ja muokattu, Itse käsky itsetehty
        print("Odota hetki!")
        try:
            client = Openai.OpenAI(api_key=avain)
        except:
            print("avain ei ole oikea tai oikein asennettu")
            d = input ("Paina enter")
            exit()

    elif (syottoVaiAvain == "K" or syottoVaiAvain == "k"):
        print("Odota hetki!")
        try:
            OPENAI_API_KEY =  os.getenv("OPENAI_API_KEY")
            client = Openai.OpenAI(api_key = OPENAI_API_KEY)
        except:
            print("avain ei ole oikea tai oikein asennettu OPENAI_API_KEY-muuttujaan")
            d = input ("Paina enter")
            exit()

    m = "Create new maze like map with size 40 times 40 letters surrounded by 1 letter wide outer wall (O)\ ···



    #print(m)
    try:
        chat_completion = client.chat.completions.create(
            model="gpt-4o",
            messages=[{"role": "user", "content": m}]
            )
    except:
        print("luonti epäonistui")
        input("Paina mitä tahansa nappia")
        exit()
#tulostaa vastauksen
map1 =chat_completion.choices[0].message.content
```

Tämä Python-kielellä annettu käskysarja käyttää siis avainmuuttujaan tai suoraan ympäristömuuttujasta haetun OpenAI:n käyttöön oikeuttavan API-avaimen hakemaan GPT-4o mallilla tehdyn kartan **message.contentin**, joka sisältää kartan, joka tallennetaan muuttujaan map1.

Koodista on peitetty suurimmaksi osaksi itse englanninkielinen 20 rivinen käsky (m = ”[…]”), sekä itse python-koodista komponenttien lataus ja kartan tallennus, tässä on vain siis se osuus kartta.py -tiedostosta, jossa kartta luodaan.

Tuloksena tutkimuksessa on, että pelkästään käskemällä ChatGPT:ä tai OpenAI:ta tekemään kentän tulos on huono. Kuten todettu, tämä johtuu yksinkertaisesti siitä, ettei se tiedä mitä sen pitäisi tehdä. Kun kokeiltiin käskyjen antamista OpenAI:lle, todettiin, että tarvittiin aina vain yksityiskohtaisempia neuvoja. Lopulta kenttä syntyi usean kymmenen rivin käskyillä. Kyse on pikemminkin selkokielisestä ohjelmoinnista kuin minkään maagisen tekoälyn käytöstä. Tämä on lohduttava tieto koodareille, ainakaan heiltä ei ole ihan heti loppumassa työt.

Kenttien tekeminen onnistui kyllä erittäin hyvin OpenAI:lta muutamaa yksityiskohtaa lukuun ottamatta. Käytin kokonaisen vuorokauden, kun yritin saada tekoälyn ymmärtämään, että kentässä pitää olla yksi ja vain yksi lohikäärme. Se synnytti sattumanvaraisesti karttoja, joissa oli yksi, ei yhtään tai lukuisia lohikäärmeitä. En ymmärtänyt tämän logiikkaa. Niinpä yritettiin lopulta tehdä koodin, jossa olisi yksi tai ei yhtään lohikäärmettä ja jälkimmäisessä tapauksessa Unitylla tehty pääohjelma arpoo lohikäärmeen paikan kentän oikeaan alaneljännekseen.

Ylipäätänsä ohjelmassa yritettiin käyttää mahdollisimman vähän sattumaa. Siis mitä AI ei itse itselleen tuottanut. Tämän vuoksi luovuttiin ajatuksesta monipuolistaa grafiikkaa sattumalla ja käytettiin satunnaisuutta vain lohikäärmeen ja päähenkilön paikan arpomiseen, ensin mainittua vain siinä määrin kuin tekoäly ei sitä itse osannut tehdä. Koska lohikäärmeen ja päähenkilön sijainti riippuu paitsi arvasta myös kentästä, ei se anna samoja tuloksia eri kentille, vaikka siemenluvun käyttö sinänsä on täysin **deterministinen** antaen eri koneillakin pelatessa saman tuloksen (Neto_Kokku, 2022).

Kuten todettu, kenttien luonti vaatii kuitenkin ihmisen jälkitarkastuksen siltä varalta, että kenttä on mahdoton läpäistä (sattuu harvemmin) ja ennen kaikkea siksi, ettei se ole liian helppo (näitä sattuu aika usein). Useimmiten kenttägeneroijan tuottama lopputulos on kuitenkin hyvä. Epäonnistuneet kentät voi yksinkertaisesti poistaa Kartat kansiosta (.txt loppuiset tiedostot).

Perinteisessä proseduaalisessa kenttätuotannossahan juuri sattumanvaraisuus on paitsi hyve myös ongelma. Ei ole mitään takuita laadun suhteen onnistumisesta tai epäonnistumisesta, jotka johtuvat seikoista, joita ohjelman kanssa toimiva suunnittelija tai koodari ei ole ottanut huomioon. (Särkiniemi, 2024, s.8). OpenAI:lla tehdyillä kentillä tämä ongelma on huomattavasti pienempi, kenttien laadun ”tarkastaa” jo luomisvaiheessa tekoäly. Ja ainakin tämän pelin suhteen tulee harvemmin vastaan kenttiä, joissa voi vain yksinkertaisesti suoraan mutkatta kävellä lohikäärmeen luo. Päinvastoin useimmat kentät tuottivat päänvaivaa niiden läpikäynnissä jopa tekoälylle käskyt antaneelle taholle.

Kun tätä aihepiiriä lähdettiin suunnittelemaan, odotetiin että OpenAI pystyy tuottamaan hyvätasoisia karttoja yksinkertaisesti sanomalla ”tee pelikenttä”. Näin ei kuitenkaan siis ollut. Aikaa sopivan käskyn luomiseen kesti kymmeniä tunteja ja silti jouduttiin tekemään kompromisseja eli turvautumaan sattumaan erityisesti lohikäärmeen luolan suhteen.

Pelin mukana tulevissa kartoissa on käytetty kymmeniä eri versioita AI:lle annettavista käskyistä. Seuraavassa lopullinen versio käskyistä englanniksi ja suomeksi (osa käskyistä tässä kahdella rivillä):

Kuva 2: Kartan luontikäskyt englanniksi

```
"Create new maze like map with size 40 times 40 letters surrounded by 1 letter wide outer wall (O)\
Begin all map lines with a start (*) sign and end it with an (O) sign.\
There can be only letters O, L, M, G, F, H and D in the map.\
Put a Dragons lair (D) in the map.\
Make lakes (L) to cover 1/10 of the total area and place them on the map.\
Scatter mountains (M) to cover 2/6 of the total area of the map. No more than 10 mountains (M) can be connected.\
If the mountains (M) surround an area make sure there is exit grass areas (G) to the areas outside the mountains(M).\
Make and place three one letter size huts (H) scattered across the map.\
Make the rest of the map 2/6 grass fields (G) and 2/6 of the map with forests (F) scattered so they are connected to each other.
At the start and end of each line has to be outer wall (O).\
Make the map size 40 letters wide, add grass fields (G) or forests (F) if necessary.\
There can be only letters O, L, M, G, F, H and D in the map,
replace all other letters or empty spaces with grass fields (G) or forests (F).\
Add grass fields (G) into the right side of the right outer wall if line is shorter than 40 letters.\
After these, make each line 40 letters wide.\
Fill the extra empty spaces with grass fields (G) or forests(F) or lakes (L) or mountains (M).\
Change any right or left outer wall (O) not on the edge of the map to grass field (G).\
Replace outer walls (O) not in columns 1,40,41,42 or at the edge of the map with grass fields (G) and forests (F).\
Search the map. If there is no Dragons lair (D) in the map, put one Dragons lair (D) in the map.\
If there is more than one Dragons lair (D) in the map, replace it with grass (G), so there is only one Dragons lair in the map.\
End map with sign '#'.\
The answer will begin with the map, nothing else. Don't give any explanations."
```

Kuva 3: Kartan luontikäskyt suomeksi

```
”Luo sokkelotyyppinen kenttä, joka on 40 kertaa 40 merkkiä kokoinen ja jota ympäröi 1 merkin paksuinen ulkomuuri (O)\
Aloita kaikki rivit alkumerkillä (*) ja lopeta (O) merkkiin.\
Kentässä voi olla vain O,L,M,G,F,H ja D merkkejä.\
Laita Lohikäärmeen luola (D) keskelle karttaa.\
Pistä järvet (L) kattamaan 1/10 karttaa ja laita ne kartalle.\
Ripottele vuoria (M) käsittämään 2/6kentästä. Yli 10 vuorta ei saa olla toisiinsa liittyneinä.\
Jos vuoret (M) ympäröivät alueen, varmista että siitä on uloskäynti ruohona (G) vuorilta (M).\
Ripottele kolme yhden merkin kokoista kylää (H) kartalle.\
Tee lopusta kartasta 2/6 ruohoa (G) ja 2/6 metsää (F) niin että ne ovat yhteyksissä toisiinsa.
Rivin alussa ja lopussa on oltava ulkomuuri (O).\
Tee kartasta 40 saraketta leveä. Lisää ruohoa (G) tai metsää (F) tarvittaessa.\
Kentällä voi olla vain O,L,M,G,F,H ja D merkkejä. korvaa tyhjät merkit tarvittaessa ruoholla (G) tai metsällä (F).\
Lisää ruohoa (F) tarvittaessa ulkomuurin (O) oikealle puolelle niin että rivi on 40 merkkiä leveä.\
Tämän jälkeen tee jokaisesta rivistä 40 merkkiä leveä.\
Täytä tyhjä merkit ruoholla (G), metsällä (F) tai järvillä (L) tai vuorilla(M).\
Muuta mikä tahansa ulkomuuri, joka ei ole oikealla tai vasemmalla reunalla ruohoksi (G).\
Korvaa ulkomuuri joka ei ole sarakkeissa 1,40,41 tai 42 tai kartan reunalla ruoholla (G) tai metsällä(F).\
Tutki kartta. Jos kartalla ei ole yhtä lohikäärmeen pesää (D) laita kartalle yksi lohikäärmeen pesä (D) keskelle karttaa.\
Jos kartalla on enemmän kuin yksi lohikäärmeen pesä (D) korvaa se ruoholla (G) niin,
että kartalla on vain yksi lohikäärmeen pesä (D).\
Lopeta kartta merkkiin ”#”.\
Vastaus alkaa kartalla, ei mitään muuta. Älä anna mitään selityksiä.”
```

Eli kartan luominen vaati 20 hyvinkin yksityiskohtaista käskyä. Yksittäistä karttaa ei tällä tavalla kannata tehdä, mutta jos tehdään useita jopa kymmeniä karttoja samalla komennolla, saadaan hyvinkin erilaisia mutta käyttökelpoisia karttoja.

Kuvaavaa käskyjen tarkkuuden suhteen on, että yhdessä vaiheessa käskettiin koneen tekemään ”some lakes” (joitakin järviä). Tuloksena oli, että kenttä oli puolillaan järviä. Vasta kun käskettiin sen tekemään 1/10 kentästä järviä oli niitä kohtuumäärä.

Perinteinen proseduaalinen kentänluonti tekee itse kentän toteutuksen, tekoälyllä on tässä taas luotu tilanne, jossa ohjelma toteuttaa kentän suunnittelun, joka on perinteisesti erillisen kenttäsuunnittelijan tai ohjelmoijan itsensä tehtävä, ”[…] proseduaalinen generointi korvaa sisällön luomisen, ei sen suunnittelua. Kokemattomat pelintekijät saattavat törmätä tähän.” (Särkiniemi, 2024, s.10).

Tämä on suurin ero mikä tässä tutkimuksessa ja pelissä on toteutettu. Itse asiassa roolit ovat jopa hieman käänteiset. Käskyn antajan tehtävä on huolehtia, että toteutus ja sen yksityiskohdat toimivat, kun tekoäly hoitaa itse kentän suunnittelun.

3.5 OpenAI:n integrointi Unityyn, tekninen toteutus tiedostolla

Kenttäeditorin luonti oli osuus, jossa jouduttiin osittain turvautumaan internetistä löytyviin koodinpätkiin, jotka toki muunnettiin tässä pelissä käyttökelpoisiksi, Itse kartta.py-ohjelma on kyllä omaa tekoani, mikäli siinä ei toisin mainita. Itse englanninkielinen käsky tekoälylle on täysin omaa tuotantoani.

Pythonilla tehdyn kenttäeditorin yhdistää peliin Kartta-kansion sisällä olevat tekstitiedostot, jotka se luo ja jotka itse Unitylla ja C#-kielellä tehty peli lukee. Niiden perusteella ruohoelementissä kiinni oleva maisemaskripti muodostaa niistä kentän lukemalla kartasta kullakin kohdalla olevat symbolit (L-järvi, H-kylä, M-vuori, G-ruoho, F-metsä ja D-lohikäärmeen pesä). Symbolit ovat englannin kielestä, kuten koko kenttäeditorin käsky OpenAI:lle.

OpenAI on siis maksullinen. Sen API-avain siis joko syötetään suoraan kenttäeditoriin tai asetetaan Pythonin ympäristömuuttujaan. Jos avainta ei anneta tai annetaan väärä avain, poistutaan pythonohjelmasta ilman että mitään tapahtuu.

Tässä kysytään lisenssiavain, mikäli sitä ei ole ohjelman ulkopuolella määritelty pysyvästi, eli OPENAI_API_KEY:n arvona järjestelmän muuttujana. Tai pythonin sisältä esimerkiksi lataamalla ympäristön (.env) arvot. (gbiz123, 2024). Tähän on ohje liitteessä 3.

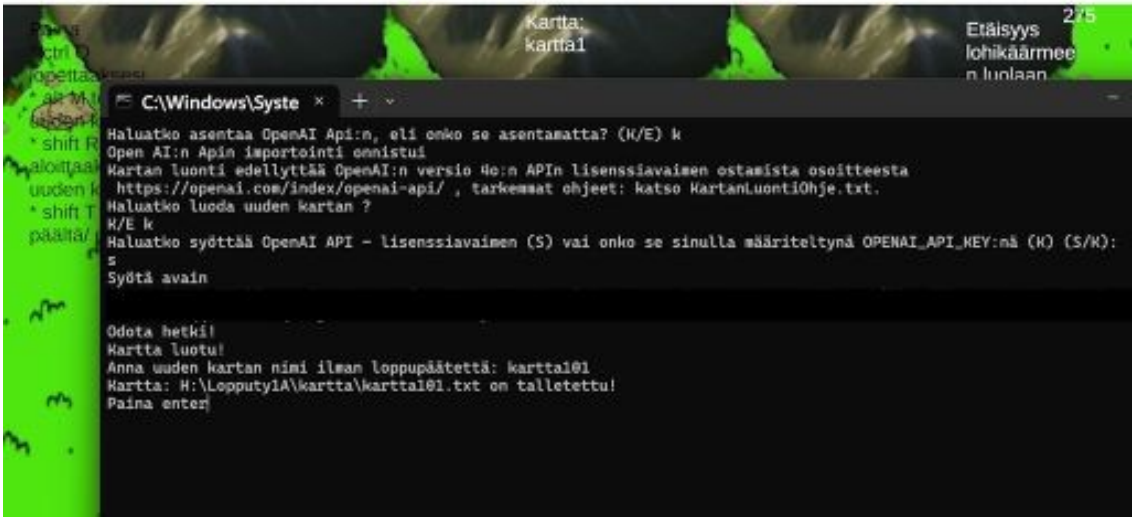
Talletetaan siis salainen avain Pythoniin **ympäristömuuttujaan**. Johonkin tämä avaimen arvo on toki joka tapauksessa talletettava, muuten se jää käyttäjän varaan.

Kuvissa 4 ja 5 näytetään ruudut, jotka ilmestyvät näytölle, mikäli painetaan Alt-M missä tahansa vaiheessa peliä. Tämä ruutu on siis erillinen pythonilla luotu ohjelma kartta.py,

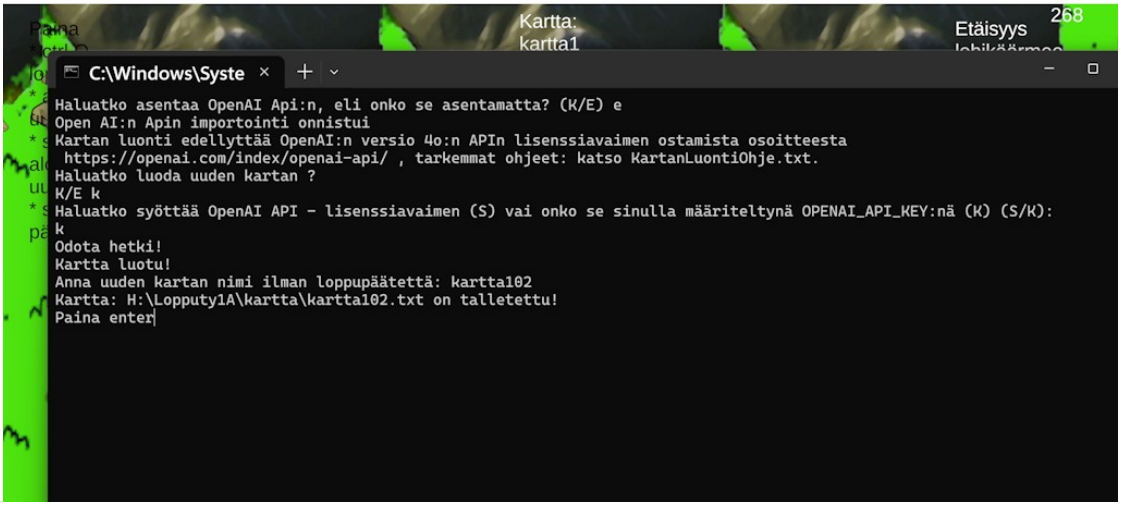
jonka pääohjelma käynnistää. Lopuksi ruutu sulkee itsensä ja jäljelle jää, mikäli jää, luotu kartta -tiedosta .txt muodossa. Kuvasta 4 on poistettu yli rivin mittainen avain. Huomattavaa on, että vaihtoehto avaimen syöttö ei aseta sitä pysyvästi ympäristömuuttujaksi vaan vain kyseisen kerran.

Pysyvästi ympäristömuuttuja täytyy asettaa **Windows PowerShellissä** pelin ulkopuolella. Tämä tapahtuu käskyllä setx OPENAI_API_KEY ”avain”. Tässä ”avain” heittomerkkien sisällä on siis saatu OpenAI:n sivulta.

Kuva 4: Kenttäeditorin käyttö 1.



Kuva 5: Kenttäeditorin käyttö 2.



Kartan luonnin käskyt ovat siis (käynnistyy Alt-M):

1. Haluatko asentaa OpenAI API:n (K/E)?
2. Haluatko luoda uuden kartan (K/E)?
3. Haluatko syöttää OpenAI API - lisenssiavaimen (S) vai onko se sinulla määriteltynä OPENAI_API_KEY:nä (K) (S/K):
4. [Syötä OpenAI-lisenssiavain:]
5. Odota hetki!
6. Kartta luotu
7. Anna uuden kartan nimi ilman loppupäätettä:
8. Kartta tallennettu

Ohjelmaa voidaan siis testata ilman maksuakin esimerkiksi niillä noin 60 kartalla, jotka on sisällytetty zip-pakettiin. Sinänsä OpenAI-palvelun maksut eivät ole kalliita, arvio on, että se pystyy tuottamaan noin 200 karttaa noin 10 €:n maksulla, tämäkin luku on varmasti alakanttiin. Toisaalta eivät nämä kartatkaan isoja ole, 40*40 merkkiä tekee noin 2 kilotavua per kartta. Siinäkin mielessä tämän ohjelman kyky pakata valmiita kenttiä pieneen tilaan on valtava.

4 Tekstitiedostosta pelikentäksi

Pelin ehkä isoin työvaihe oli maiseman perustana käytettyjen palojen teko. Ne tehtiin käsin, ei siis AI:lla, vaikka sekin olisi tietysti ollut mahdollista. Ongelmana olisi ollut collidereittein lisääminen spriteihin ja osamaisemien toimivuus, eli että esimerkiksi metsän lävitse voi kävellä. Olisi ollut mahdollista esimerkiksi tehdä metsää useita erilaisia paloja, joita olisi voinut vaihdella satunnaisesti. Tässä yritettiin säilyttää kenttäsuunnittelu kuitenkin mahdollisimman vähän satunnaisena ja kevyenä toteuttaa, joten käytettiin vain yhdenlaista metsäpalikkaa. Tulos oli kuitenkin ainakin omaa silmää miellyttävä.

Kun tehtiin OpenAI:n suunnittelemaa kenttää, tehtiin lopputulos siis kuten yleensä proseduaalisessa kenttien tuottamisessa eli käsityönä (Särkiniemi, 2024, s.4). Eli suuret linjat muodostettiin tekoälyllä, pienet palat itse suurimmalta osin valmiista grafiikkakirjastoista. Esimerkiksi puut tai talot tai lohikäärme olivat maksullisista tai ilmaisista grafiikkakirjastoista, jotka hankittiin Unityn omalta markkinapaikalta. Hintaa näille tuli noin 30 euroa, tosin jos haluaa grafiikkoihin voi kyllä käyttää helposti jopa tuhansia euroja.

Yksi poikkeus tästä oli vuoren sprite-kuvio. Siinä on pohjana OpenArt.ai:n avulla luotu kuva vuoresta. Sen muodostamisen jälkeen käytettiin Corel PaintShop -kuvankäsittelyohjelmaa. Kuvasta poistettiin tausta, josta tehtiin läpinäkyvän ja muokattiin se muutenkin **PNG-kuvaksi**, joka oli spriteksi sopivassa muodossa ja koossa. Sitten lisättiin siihen muita osia, kuten kiviä, jotta saatiin ”tiili” näyttämään siltä, ettei siitä pääse mistään kohtaa lävitse.

Kun pääohjelma on lukenut .txt -tiedoston, käy se läpi sen merkki merkiltä. 40 riviä ja 40 saraketta. Jokaisen kohdalla se katsoo, onko luettu merkki jokin joukosta M,H,D,F tai L. Ruoho, eli G tarkoittaa vain yksinkertaisesti, että mitään ei piirretä taustalla olevan kentän päälle, joka on tehty **tile**-vaihtoehdolla, niin että yksi pieni sprite on ladottu vierekkäin satoja kertoja.

Liitteessä 6 on maisemanluonnin pääluuppi C# -kielellä, selostus on huomautuksissa eli merkkien // jälkeen. Koodin osa on 95 riviä.

Kartalla merkki ”O” tarkoittaa ulkoreunaa. Se on merkitty vaaleanpunaisella tyylitellyllä nauhalla. Vaikka siihen on pyritty, niin kaikissa kentissä ei ole välttämättä mahdotonta karata tämän rajan ulkopuolelle, mutta siellä ei ole kuin ruohoa ja alkuperäiset elementit, joiden **instansseja** kentän eri elementit ovat. Esimerkkinä tässä kartta8.txt -tiedoston (ei siis testikenttä) sisältö kuvana, eli karttaesimerkki tietokoneen luomasta kentästä.

Kuva 6: Karttaesimerkki symbolisella kirjaingrafiikalla

```
OGGGGGGGGGGGGGGGGGGMGGGGGGGGGGGGGGGGOO
OGLLGGGGFGGFFFFFFFFMMGGLLLLGGGFFFGGGGGGMO
OGFFGGGGFGGFFFFFFFFMMGGLLLLGGGFFFGGGLLFFO
OGFFGGMMMGGGGGGLLGGFFGGLLLLGGGGGLLGGGGFFO
OGFFGGGMMGGMGGLLLLMGGGLMGGGFGGFFFFGMGGFFO
OGLLFFGGGGMGGGLLLLMGGGMMGGGFGGLLLLLMLGGFO
OGGGFFFGGGGMGGLLLLLGGGGMMGGGGLFFFLGGGLLFO
OGGGLLLLGGGGMGGLLLLGGGGGGMGGFFFFFFGGGGFFO
OFGGLGGGLGGFGGGGGGGGLLGGGGGGLGGGGLLGGFFMO
OGFFFGLGGLLLMGGGGGGGLLGGGMGGGFLGGGGGGFFFO
OGGFGGFFFGFGGGGGGGGGLLGGMMGGGGFGGGGGGFFFFO
OFGLFLLLLFFGGGGGGGGGLLLLMMGGFGGGGGLFFFGOO
OGGFLLLLLFFGGGGGGGGGLLGGMMGGGFGGLLLLLLFOO
OGLFLLLLLFFGGGLFGGGGFFGGMMGGGFLLGFGGGGFO
OGGFLLLLLFFFFGLFGGGGFFGGMMGGLFFGGGGGGGFO
OGGLLLLLLFFFLGGFGGGGLGGGMMMGLLFGGGGGFGLFO
OGGLLLLLLFFGGGLFGGGGGGGGMMMGGLLGGGFFFLFO
OGLLGGGGGFLGGGLFFFFFLLGGMBMGGGGFFGLFLFGFO
OGGGGLLLLGLGGGLFFFFFFFFLGMMMGGGGFLGFLGFLO
OGFFGLLLLGLGGGFMMMMMMGGGMBMGGGGFLGGGLFGOO
OGFGGLLLLLGLGFFFFFFFFFGLGMGGBBGGGFFLLGOO
OGFLLLLLFFGGLMMGMMMMMGGGGGLFGGGGGLGGFFGOO
OGFBMMMMMMGGMGGGGGGGGGGFFGGGGFGGGGGGFGFOO
OGGBMMMMMMFFGGGGGLGGGGGFGGGGGLGGGGFGFGFOO
OGGFLGGGLLFGGGGGGLGGGGLGGGGGGGGGGGFFFGOO
OGFLGGLLFFFGGGFFMMGGGGGLGGGGGGGLLLLFFGOO
OGGGFMMMMGFBFGGMMBMGGGMMMMGGGLLLLLLFFFOO
OGLGFMMMMLGGFMMMMMMGLLLLGGGFFLGLGFGFLFOO
OGLGFLMMLFBGFGMMMMGFGFGLGGGFLGLGLGGFFLOO
OGGGGFFGGLGFGLGBGGMFGLFFFGBGLGLGGGFGFFGOO
OGFFHGFLFGFFGLFGGLLFFFFFBGBGLGLFGGFFFGOO
OGFFFHGGGFGLGLFGLBMLLFGFLFGLGFFGGGLFFGOO
OGGFHHGGLFFFGLGGFFFLLFGGLFFGFFGGGLFGLGOO
OGFFFGGGGGGGGLFGLGGGBGGLGGGGFGLGMMMMGFOO
OGFFFGLGGGFGLFGLGGGLBGFFFFFGLGMMMMMMGFOO
OGGFFFFLLFFFFFGGGGFFFLFFFFFGLGMMMMMMGFOO
OGLLFFFLLLLFFFFFFFFFLFFFFFLGLGGFGGFMMGOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO#
```

Kuten huomataan, ei kartta ole tasasivuinen. Tämä korjataan oikealla laidalla ohjelmallisesti, siinä määrin kuin siihen ylipäänsä mennään. ”#”-merkki on tiedoston loppu. Tällaisia karttoja voi tehdä tekstieditorilla helposti itsekin käsin. Hyvän kartan tekeminen vaatii kuitenkin vaivaa. (Liikkuminen tapahtuu puolestaan kameran jatkuvasti seuratessa päähenkilöä, kuten 2D-peleissä usein.)

Mikäli käy niin että kartassa ei ole yhtään lohikäärmeen pesää, kuten tässä kartassa ei ollut, arvotaan sille sijainti kentän alaoikealle tosin ei aivan perille, niin että se osuu periaatteessa ruohoon tai metsään. Pesä tulee joka tapauksessa valitun tiilen päälle.

Kaikista ruohopaloista on yhteys toisiin ruoho- tai metsäpaloihin, eli hahmo ei voi jäädä vuorien saartamaksi. Tämän kaltaisia ongelmia jouduttiin miettimään muokatessa tekoälylle annettavaa käskyä. Silti ohjelma ei ole täysin varma, se voi tuottaa kentän, joka on mahdoton läpäistä. Yleensä käskyn lisämuokkaaminen on poistanut tämän ongelman, mutta erehtyminen on tekoälyllistä…

5 Vertailu: Ihmisten ja tekoälyn tekemät pelikentät, testi koehenkilöillä

5.1 Testiryhmä, ikä ja pelanneisuus sekä tutkimustyyppi

Tutkimusta tekemään aloittaessa olisi lähtökohtana voinut olla **Hallway Testing** -tyyppinen testi (esim. Dashly, 2020), jossa esimerkiksi kahvilassa olisi testattu eri ryhmiin kuuluvia henkilöitä pelin käyttökelpoisuudesta ilman mitään ennakkoinformaatiota.

Päädyttiin kuitenkin tekemään eräänlaista **Turingin testiä**, eli arvuuttelemaan ihmisillä mikä on ihmisen ja mikä tietokoneen tuote. Testistä on tuoretta kokemusta nimenomaan ChatGPT-4:n osalta keskustelusta; se läpäisi tämän testin (Pyyny, 2024).

Pelikenttiä koskevaan kokeeseen saatiin tosin vain neljä osanottajaa: yksi eläkeläinen, joka pelaa vain pasianssia ja vastaavia pelejä, yksi työiässä olevan nainen, joka pelaa myös pasianssia, mutta kumpikaan ei mitään toimintapelejä tms. ”videopelejä” ja sitten kaksi koululaista, jotka pelaavat runsaasti. Otos on sopiva, kun mitään laajaa otosta, jolla olisi voinut toteuttaa kvantitatiivisen tutkimuksen, ei ollut mahdollista järjestää.

Kaikkein kriittisimpiä pelin suhteen olivat täysi-ikäiset pelaajat, joilla ei ollut siis paljon videopelien käytön kokemusta. Kritiikki kohdistuikin esimerkiksi grafiikkaan, joka ei ollut itse tutkimuskohde.

Koska testiryhmään kuului lapsia/nuoria, tehtiin pelistä kaksi versiota, joista koeversiossa ei voi luoda lainkaan karttoja, niin etteivät nuoret koehenkilöt käyttäisi rahaa tähän (tarkoitus on kuitenkin jättää julkaistuksi tämän peli kokonaisena täysin toimivana versiona yleiseen jakeluun). Näin tein siitä huolimatta, että AI-avaimen hankkiminen on monimutkainen operaatio, joka vaatii luottokortin, joten ei ole myöskään todennäköistä, että tätä voisivat nuoret koehenkilöt ilman vanhempiensa suostumusta tehdäkään. Tietysti kysyin luvan nuorten vanhemmilta kyselyn tekemiseen sinänsä. Koeversiossa ei ollut siis kartta.py tiedostoa ja itse pääohjelmaa oli muutettu niin ettei kentän luonnin valitsemisesta

tapahtunut mitään. Molemmat versio ovat siis ladattavissa GitHubista (täysi versio) ja täältä (koeversio). Koska haastattelu tehtiin avustajan välityksellä, vain hän latasi pelistä version, jota hänen koneellaan sitten testattiin.

5.2 Testausasetelma

Koska testiotos oli niin pieni, neljä henkeä, ei kvantitatiivinen tutkimusote siis ollut mahdollinen. Lähinnä taustakysymyksiä lukuun ottamatta kysymykset olivat avoimia kvalitatiivisia kysymyksenasetteluja. Tarkat kysymykset löytyvät liitteestä 1.

Taustatiedoissa kysyin ikää, etunimeä (jota ei tässä julkaista), sekä pelanneisuutta (1 = päivittäin … 6 = ei koskaan). Kerrottiin, että pelissä on mahdollisuus luoda tekoälyllä kenttiä, mutta tätä ei tässä testattu, vaan kokeessa käytettiin kuutta ensimmäistä karttaa (n. 60:stä), jotka koehenkilön piti itse ladata pelin sisällä. Kaksi näistä kuudesta kartasta oli tehty käsin (kartta2 ja kartta4). Yksi pääkysymys olikin, että mitkä kartat oli luonut tekoäly. Olisi siis voitu salata tekoälyn käyttö kokonaan, mutta mielenkiintoisempaa oli asetelma, jossa koehenkilö pelatessaan joutuu arvuuttelemaan, onko kenttä ihmisen vai koneen suunnittelema.

Tehtiin myös selväksi, ettei tarkoitus ole testata itse grafiikkaa, joka oli siis ihmisten (pelin tekijän ja **Unity Asset Storessa** toimivien kauppiaiden myymien pelien ”rakennuspalikoiden” avulla luotua) tuotosta. Pelissä käytettiin siis myös yhden ”tiilen” grafiikan tekemiseen tekoälyä.

Kysymykset ja lupalomakkeet välitettiin sähköpostitse avustajalle. Hän peluutti peliä haastateltavilla noin puolen tunnin ajan, jonka jälkeen he vastasivat kysymyksiin. Pelaajia pyydettiin kokeilemaan jokaista kenttää vähintään kerran, mutta useimmat halusivat yrittää päästä niistä lävitse ja kokeilivat niitä moneen kertaan.

5.3 Pelaajapalaute, eroavaisuudet ja yhtäläisyydet ihmisen ja GPT-4o:n suunnittelemien kenttien välillä

Pelaajapalaute oli yleisesti positiivinen. Alaikäiset koehenkilöt halusivat pelistä itselleen version. Vanhemmat koehenkilöt olivat siis jopa kriittisempiä kuin nuoret, jotka olivatkin nähneet monenlaisia pelejä ja ymmärsivät että grafiikka ei kaikissa niissä ole pääasia.

Kokeesta oli myös hyvä puoli se, että siinä selvisi muutama koodivirhe palautteen mukana. Samoin saatiin palautetta itse koejärjestelystä, joka on hyödyksi, jos vastaavaa tutkimusta joskus teen. Esimerkiksi ohjeita pyydettiin yksinkertaistamaan.

Yleisesti kukaan koehenkilöistä ei osannut erottaa täysin mikä oli tekoälyn ja mikä ihmisen tekemä kenttä, joko he sanoivat, etteivät pystyneet erottamaan tai veikkasivat puolet väärin. Parhaaksi kentäksi valittiin kahdessa tapauksessa kenttä numero kolme, joka oli siis tekoälyn tekemä. Toisin sanoen peli läpäisi testin, eli ihminen ei osannut erottaa mikä oli tietokoneen suunnittelema ja mikä ihmisen.

Kentistä valitettiin, että ne olivat samankaltaisia. Tämä on ymmärrettävää, koska kentät koostuivat vain kuudesta tiilestä ja lohikäärme oli lähes samassa paikassa eri kentissä, edellä kerrotuista syistä. Samoin todettiin peleistä voisi hyvinkin tehdä tällä tavoin ”oikeita” pelejä, grafiikkaa vain pyydettiin kehittämään, mikä ei sinänsä liittynyt koekysymykseen.

Testi oli siis siinä mielessä onnistunut, etteivät osallistujat osanneet erottaa ihmisen ja OpenAI-4o:n tekemiä kenttiä toisistaan. Tietokoneen suunnittelema kenttä (kartta3) oli kaikille ylivoimaisesti vaikein. Vaikeutta se osoitti koska lohikäärmeen pesä oli sen karttaan sijoittama, ei siis sattumanvarainen. Tekoäly ymmärsi selvästi käskyjen perusteella, millaista karttaa käyttäjä haki, vaikka se ei tiennyt itse pelin juonta.

5.4 Tekoäly ja kenttäsuunnittelu, ihminen ja AI vuorovaikutuksessa kenttien suunnittelussa

Isoin ongelma oli kuten todettu lohikäärmeen sijoittaminen, joka onnistui tekoälyltä oikein vain harvoin. Tässä korjaajana käytettiin aika yksinkertaista algoritmia, joka sijoitti lohikäärmeen kentän oikealle alaneljännekselle. Koehenkilöt myös huomasivat tämän. Jos tällaista projektia tekee, niin tähän pitää kiinnittää enemmän huomiota.

Tekoäly ei siis tiennyt pelin juonta, se vain loi kenttiä. Siihen nähden se osoitti kuitenkin usein miten riittävän selvästi ymmärtävänsä, millaista kenttää haettiin. Ja kentät vaihtelivat toisistaan, joissakin ei ollut vuoria, toisissa ei kyliä, joissakin paljon järviä jne. Näin siis siitä huolimatta, että käskyssä oli annettu selkeät määrät, kuinka paljon kutakin elementtiä pitäisi olla. Eli kone ei vain mekaanisesti totellut käskyjä, vaan osoitti eräänlaista luovuutta. Tässä on iso ero perinteisiin algoritmeihin, joista tietää periaatteessa aina mikä on

lopputulos. Vielä suuremmalla vaivannäöllä ja testaamisella on mahdollista tehdä aivan kaupallisen tason kenttiä.

Täysin automaattista kentänluontia tämä peli ei pystynyt tarjoamaan. Kuitenkin vain vähemmistö sen luomista kentistä jouduttiin hylkäämään liian helppoina tai mahdottomina. Joukossa on todellisia helmiäkin.

Ihmistä tarvitaan kolmessa vaiheessa. 1) Itse käskyn luomisessa, joka on aikaa vievä vaihe, mutta ei yhtä paljon kuin jos kentät olisi jouduttu tekemään kokonaan käsin. 2) Sitten tietysti itse ohjelmat täytyy ihmisen itse tehdä, toinen luomaan ja toinen tulkitsemaan karttoja. Ja 3) karttojen laadun tarkastaminen on myös ihmisen suorittama.

5.5 Ohjelmoitaessa huomioonotettavat asiat

Pelin teko oli suhteellisen työläs operaatio. Jos tämän tyyppistä peliä tekee, kannattaa etukäteen opetella C#, Unity ja Python. Tässä pelissä käytettiin myös muiden tekemien koodien ideoita, mutta ei plagioitu. Pahimmillaanhan tekoäly on **plagiaatti-software** (“basically high-tech plagiarism”, [”pohjimmiltaan huipputeknistä plagiaattia”]), kuten kielitieteilijä Noam Chomsky (Marshall, 2023) asian ilmaisi. Tällä tavoin käytettynä AI estäisi oppimisen, eli koko opiskelun perimmäisen syyn.

Kannattaa pitää itse ohjelmista runsaasti varmuuskopioita eri versioista. Samoin jos avainkoodi ei ole asetettu ympäristömuuttujaksi, kannattaa se pitää helposti saatavilla esim. .txt-tiedostossa. Ja testausta pitää olla runsaasti, myös silloin kun ohjelma tuntuu toimivan. Varsinkin tekoälylle annettavaa englanninkielistä käskyä joutui muuttamaan paljon.

6 Yhteenveto

6.1 Opinnäytetyön onnistumiset/epäonnistumiset

Suurin epäonnistuminen testissä oli testiryhmän pienuus. Alun perin oli tarkoitus tehdä useampia tutkimusistuntoja, mutta jo neljän hengen testaaminen osoittautui sen verran sekä vaikeaksi järjestää että testattavilta aikaa ja vaivaa vaativiksi, että luovuttiin ajatuksesta suuresta satunnaisotoksesta. Tämä tietenkin merkitsee, etteivät tulokset ole tilastollisesti päteviä. Niiden tarkoitus on vain osoittaa itse ohjelman teossa olleita ongelmia

ja ennen kaikkea mitä tämän tyyppisessä ohjelmassa pitää ottaa huomioon ja ylipäätänsä onko LLM:sta lainkaan symbolisen grafiikan tekoon (eli on).

Peli itsessään toimi jopa pelinä, vaikka tämä oli vain sivuasia kenttien testaamisen tekemiseksi edes jossakin määrin järkeväksi. Jos haluaa tehdä esimerkiksi roolipelin tällä tekniikalla, on se mahdollista, mutta vaatii huomattavasti enemmän vaivaa jo kartta.py -ohjelman englanninkielisten käskyjen muodostamiseen. Nythän valittavia ”tiiliä” oli vain kuutta erilaista.

6.2 Tuloksien laajempi merkitys

Aivan ensimmäiseksi on sanottava, ettei tekoäly ole viemässä ohjelmoijien työpaikkoja ainakaan ihan heti. Ensinnäkin AI vaatii aivan yhtä lailla tarkkoja ohjeita mitä sen pitää tehdä, jotta se voi tietää mitä siltä odotetaan tulokseksi. Periaatteessahan tähän voisi kyllä yhdistää internetin loputtoman esimerkkiaineiston, mutta jos halutaan luoda jotain uutta, siihen tarvitaan ihmisen kädenjälkeä.

Tekoälyllä on luotu valmiita grafiikoita ja toisaalta generoitu tiilistä kenttiä ennenkin. Tämän tutkimuksen tulos oli, ettei tämän tarvitse olla sattumanvaraista tai vaivanloista. Ennen kaikkea merkkigrafiikka mahdollisti pieneen kokoon pakattuna suuren määrän kenttiä. Näissä kentissä oli usein miten selkeä suunnitelmallisuus, jopa ihmisälyyn verrattava. Kunhan käytännön kysymykset esim. maksullisuudesta ratkaistaan, on mahdollista tehdä aivan kaupallisen tason pelejä tällä tekniikalla. Eikä mikään estä käyttämästä vastaavaa tekniikkaa kolmiulotteisissa kentissä. Sitä varten täytyy vain kehittää omat algoritminsa.

Sinänsähän on kysymys vain selkokielisestä ohjelmoinnista, jolla on omat rajoitteensa. Nimittäin jos halutaan täsmällisesti saada jokin tietty tulos, perinteiset ohjelmointikielet ovat selkeämpiä ja tarkempia algoritmin tekemiseen. Mutta jos halutaan esimerkiksi luoda suuri määrä karttoja, jotka voitaisiin tehdä muutaman kymmenen rivin käskyillä, on kiusaus varmasti suuri ja miksi ei olisi?

Lähteet

BlackHellHound1. (4.7.2016*).* Make IL-2 look stunning with improved ground detail and visibility! *How-to and discussion!*, *Il2-Sturmovik forums*. https://forum.il2sturmovik.com/topic/23699-make-il-2-look-stunning-with-improved-ground-detail-and-visibility-how-to-and-discussion/

Dashly. (1.12.2020*).* A Detailed Guide on Hallway Usability Testing. *Dashly Blog.* https://www.dashly.io/blog/detailed-guide-on-hallway-testing/

Deeeds. (2018). Difference between 2d and 3d projects. *Discussion Unity.* https://discussions.unity.com/t/difference-between-2d-and-3d-projects/714593

gbiz123. (22.3.2024). How to set persistent environment variable using python*.* *Stack overflow*, https://stackoverflow.com/questions/78200609/how-to-set-persistent-environment-variable-using-python

Heiskanen, E. (2021). *Python-kieli ensikosketuksena koneoppimiseen* [Opinnäytetyö, Metropolia Ammattikorkeakoulu].

Joensuu, J. (2016). *3D-ALAN SANASTO – 3D-grafiikan termit suomeksi* [Opinnäytetyö, Kajaanin Ammattikorkeakoulu].

Marshall, C. (10.2.2023). Noam Chomsky on ChatGPT: It's "Basically High-Tech Plagiarism" and "a Way of Avoiding Learning*".* *Open Culture*. https://www.openculture.com/2023/02/noam-chomsky-on-chatgpt.html

Neto_Kokku. (2022). System.Random with same seed not 100% deterministic? *Discussion Unity.* https://discussions.unity.com/t/system-random-with-same-seed-not-100-deterministic/876188

OpenAI. (2015–2025). *OpenAI Developer Platform.* https://platform.openai.com/docs/overview

Pyyny, P. (15.6.2024). Tekoäly läpäisi Turingin testin*.* *afterDawn*. https://dawn.fi/uutiset/2024/06/15/tekoaly-lapaisi-turingin-testin

Qui, Z., Liu W., Feng, H., Liu, Z., Xiao, T., Collins, K., Tenenbaum, J., Weller, A., Black, M. Schölkopf, B. (15.8.2024). *Can Large Language Models Understand Symbolic Graphics Programs?* Max Plank Institute for Intelligent Systems, University of Cambridge, MIT. https://www.arxiv.org/abs/2408.08313

Selin, J. (2024). *Unity, osat 1 ja 2* [xamk-ammattikorkeakoulu].

Särkiniemi, E. (2024). *Proseduaalinen generointi modernissa pelinkehityksessä* [Opinnäytetyö, LAB-ammattikorkeakoulu].

Unity Technologies. (2024). *Unity 2D suite for game creators.* https://unity.com/solutions/2d

Liite 1. Kyselyn kysymykset

Tutkimukseen sisältyneen empiirisen kyselyn kysymykset

- Etunimi (ei julkaista), ikä
- Kuinka paljon pelaat tietokone-/kännykkä-/pelikonsolipelejä? (1= lähes päivittäin tai päivittäin, 2= joka viikko, 3 = joka kuukausi, 4 = harvemmin, 5 = olen kokeillut pelaamista, 6 =en ole edes kokeillut.)
- Toimiko peli mielestäsi oikein, oliko siinä jokin selvä vika (jos oli, niin voidaan yrittää korjata)?
- Tehtävänäsi on pelata kuusi (6) ensimmäistä kenttää joko lävitse tai kunnes aika loppuu. Kentän numeron näkee ylälaidassa (kartta1 jne.). Kentän voi valita painamalla Shift-R ja syöttämällä valkeapohjaiseen kenttään ”kartta1”, ”kartta2”, ”kartta3” jne. Kenttiä voi myös luoda itse painamalla Alt-M, mutta tätä ei tässä testata ja se vaatii rekisteröitymisen ja maksun (n.10 €) ulkopuoliselle taholle. Valitse vain siis kuusi ensimmäistä kenttää 60:stä. Pelattuasi ne kaikki 6 ensimmäistä kenttää lävitse arvioi mikä niistä oli mielestäsi paras.
- Pelattuasi 6 kenttää joko lävitse tai ajan loputtua (voit toki halutessasi pelata niitä useaan kertaan), arvioi mitkä niistä olivat ihmisen ja mitkä tekoälyn suunnittelemia. Suunnittelulla tarkoitetaan kentän asettelua, ei varsinaista graafista ulkomuotoa.
- Olivatko kentät yleisesti mielestäsi sellaisia, että niitä voisi käyttää muutenkin kuin koemielessä peleissä?
- Mitä mieltä olet tämän kokemuksen jälkeen ylipäänsä tekoälyn käytöstä pelikenttien tekemiseen?

Liite 2. Ohjelman tekniset vaatimukset

Ohjelma on testattu toimivaksi kokoonpanolla:

- AMD Ryzen 5 4600G
- 16 Gt RAM
- NVIDIA GeForce GTX 1660S (6Gt).

Ohjelma voi toimia heikommallakin kokoonpanolla, mutta ei esimerkiksi 2Gt:n näytönohjaimella ainakaan niin kuin sen pitäisi toimia. Yleisesti Unity 6 vaatii DX12- ja **Vulkan**-kelpoisen näytönohjaimen ja 64-bittisen prosessorin, sekä vähintään Windows 10-käyttöjärjestelmän.

Liite 3. Kenttägeneroijan käytön ohje

(Lohikäärmeen tappaja -pelin kenttägeneroijan käyttö.)

Tässä pelissä on valmiina n. 60 kenttää Kartta kansiossa. Ne voi vaihtaa pelissä toisiinsa painamalla shift-R ja syöttämällä kentän nimen ilman .txt loppupäätettä. Uusia kenttiä pystyy luomaan aukaisemalla kartta.py -ohjelman pelin sisällä painamalla alt-M, joka avaa uuden ikkunan, olettaen että on asennettu pythonin suht tuore versio. Editorin käyttö edellyttää OpenAI-4o API:n käyttölisenssiä, joka on maksullinen. Se maksaa vähimmillään 10 euroa ja kuluu käytön mukaan. Sen voi hankkia osoitteesta https://openai.com/index/openai-api/ ja kohdasta sign up rekisteröitymällä, antamalla maksutiedot jne. Myöhemmin API key:hin pääsee käsiksi Products valikosta valitsemalla API login. Sisäänkirjautumisen jälkeen valitaan API reference yläoikealta ja Authentication kohdasta User keys API keys ja näkyy painike +Create new secret key. Valitaan permission all ja default project. Create jne. SECRET KEY näkyy API keys -kohdassa kokonaan vain kerran, kopioi se turvaan. Avain on pitkä. Muussa tapauksessa voit joutua luomaan uuden avaimen. Avainta ei saa levittää, muuten se voidaan sulkea, siksi tässä pelissä ei ole sitä valmiina.

API key kannattaa asettaa ympäristömuuttujaksi **powershellissä** setx OPENAI_API_KEY ”[avain]” -käskyllä, jolloin sitä voi käyttää suoraan ohjelmassa valitsemalla kohdassa, jossa kysytään, halutaanko syöttää avain ”S”, vai käyttää valmiiksi asennettua avainta ”K”, niin kartta.py ohjelma osaa hakea sen automaattisesti.

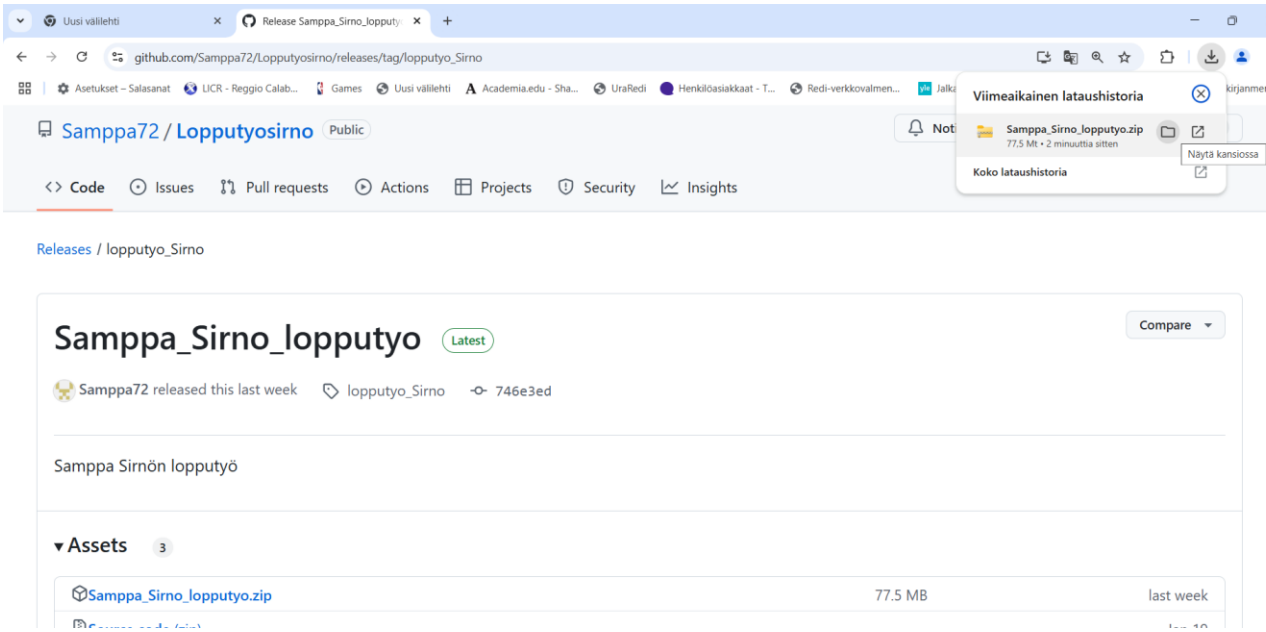
Ja mikäli ei haluta ostaa API keytä, voi mennä kartta.py ohjelmaan ja kopioida sieltä parikymmentä riviä englanninkielistä käskyä (m = "XXXX", jossa X=komennot) ja syöttää sen ChatGPT-syötteeseen ja tuloksena on aivan toimiva kartta, jonka voi tallettaa .txt-tiedostona kartta -kansioon. Karttoja voi tehdä myös itse käsin.

Liite 4. Pelin lataaminen

Mene osoitteeseen: https://github.com/Samppa72/Lopputyosirno/releases/tag/lopputyo_Sirno . Lataa peli kohdasta Assets. Peli on nimeltään Samppa_Sirno_lopputyo.zip .

Paina hiiren vasemmalla näppäimellä tuota nimeä ja talleta peli haluamaasi paikkaan. Mene kansioon, johon olet pelin tallettanut (esimerkiksi selaimen oikeassa yläkulmassa lataussymboli ja siitä kansio ikoni). (Kuva 7).

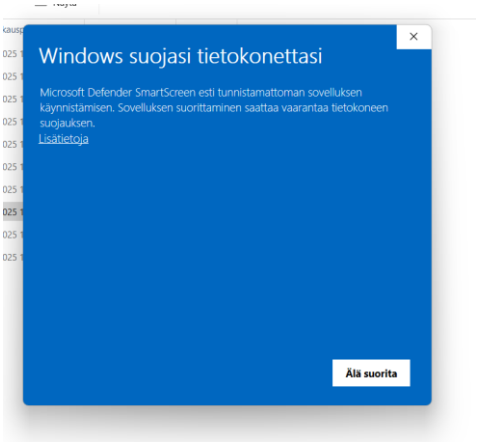
Kuva 7: Pelin lataaminen.



Paina tiedoston Samppa_Sirno_lopputyo.zip kohdalla oikeaa nappia ja avautuvasta valikosta pura kaikki. Avautuvassa kansiossa peli aukeaa painamalla Lopput.exe kohdasta

Jos ruudulle ilmestyy seuraava kuva

Kuva 8: Varoitusruutu.



Kaksoisklikkaa lisätietoja ja Suorita joka tapauksessa, niin peli alkaa.

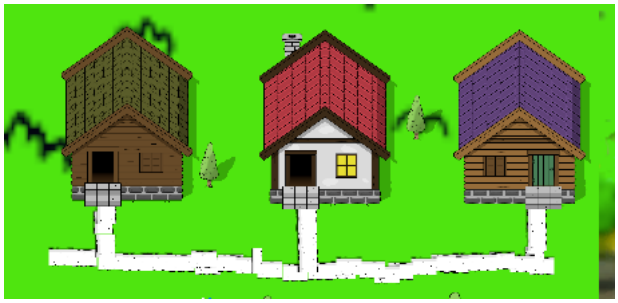
Liite 5. Ohjeet pelikenttiin ja pelin pelaamiseen ylipäänsä

Lohikäärmeen tappaja pelin idea ja pelaamisohjeet

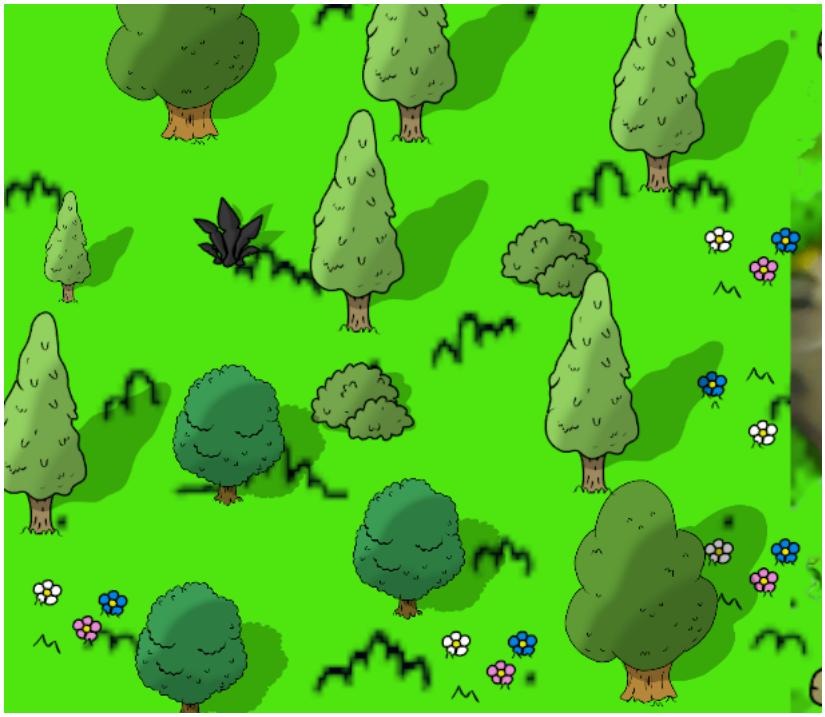
- Lohikäärmeen tappaja -pelin idea on löytää lohikäärmeen luola viidessä minuutissa kentässä, joka koostuu kuuden erilaisen elementin vaihtelusta (ruoho, metsä, järvi, vuori, kylä, lohikäärmeen pesä).
- Päähenkilö (kuva 9) voi liikkua kahdeksaan suuntaan WASD- tai nuolinäppäimillä.
- Kylistä (kuva 10) saa sata sekuntia lisäaikaa käymällä niiden luona, mikäli niitä kentässä on.
- Metsässä puiden rungot ovat läpikäymättömiä, mutta latvan takaa voi kävellä (Kuva 11).
- Järvet ovat läpikäymättömiä, mutta eivät peitä kuin osan elementin alueesta. Järvien luona sataa, mikä ei vaikuta peliin millään lailla. (Kuva 12).
- Vuori on kokonaisuudessaan läpikäymätön. Mikäli kahden vinottain liittyneitten vuorielementtien yhtymäkohdasta voi mennä lävitse, kyse on joko bugista tai siitä että tietokoneen suorituskyky on niin heikko, ettei se reagoi jokaiseen liikuttuun kohtaan. (Kuva 13).
- Ruoho on täysin läpikäveltävä elementti.
- Lohikäärmeen pesä on täysin läpikäveltävä. Siihen kävely päättää pelin voittoon ja lohikäärmeen kuolemaan. Mitään varsinaista lopputaistelua ei käydä, riittää kun kävelee luolaan, joku koostuu vuoresta, aarteesta ja lohikäärmeestä. (Kuva 14)
- Pelissä on tekstikenttiä. Vasemmalla ohjeteksti, jonka saa pois päältä ja päälle painamalla Ctrl-T. Ylhäällä on teksti, joka kertoo käytössä olevan kartan nimen ilman .txt loppua. Oikeassa yläkulmassa on jäljellä oleva aika. Siitä alavasemmalle puolestaan löytyy Pythagoraan lauseella laskettu etäisyys lohikäärmeen ja päähahmon välillä. Lisäksi painamalla Shift-R ilmestyy teksti, jossa kysytään mihinkään kenttään haluat siirtyä, sekä sen alle valkopohjainen syötekenttä, johon voi syöttää halutun kentän nimen ilman päätettä ja painamalla Return-näppäintä peli lataa tämän kentän. Alt-M:stä ilmaantuu python-kielellä ohjelmoitu kenttäeditorin käyttäjäliittymä, jota on käsitelty liitteessä 1, Pelin alussa on ohjeet ja pelin loputtua tieto voitosta tai tappiosta keskellä ruutua. Painamalla Ctrl-Q, peli päättyy lopputekstien ja Returnin painamisen jälkeen. (Kuvat 15, 16, 17, 18, 19, 20).
- Kenttää reunustaa vaaleanpunainen katkoreuna (kuva 21).

Kuva 9: Lohikäärmeen tappaja -pelin päähahmo.

Kuva 10: Kylän kuva.



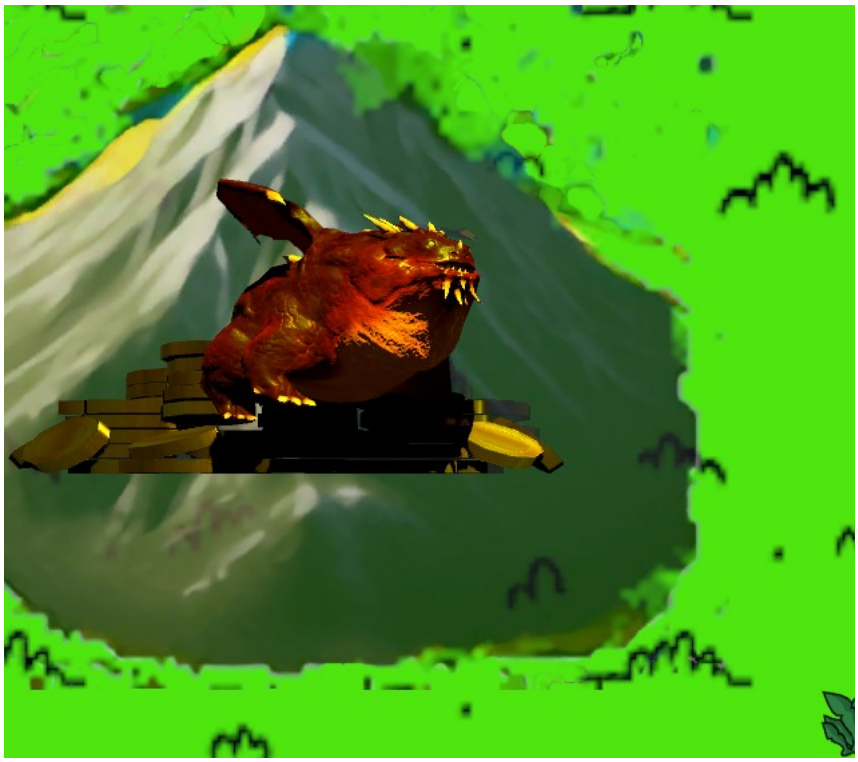
Kuva 11: Metsä-kuvatiili.



Kuva 12: Kuvassa järvi ja päähahmo. Sataa.

Kuva 13: Vuoren kuva.

Kuva 14: Etsinnän kohde, eli lohikäärme ja aarre.



Kuva 15: Pelin päänäkymä, tekstit esillä.

Kuva 16: Pelin aloitusruutu.

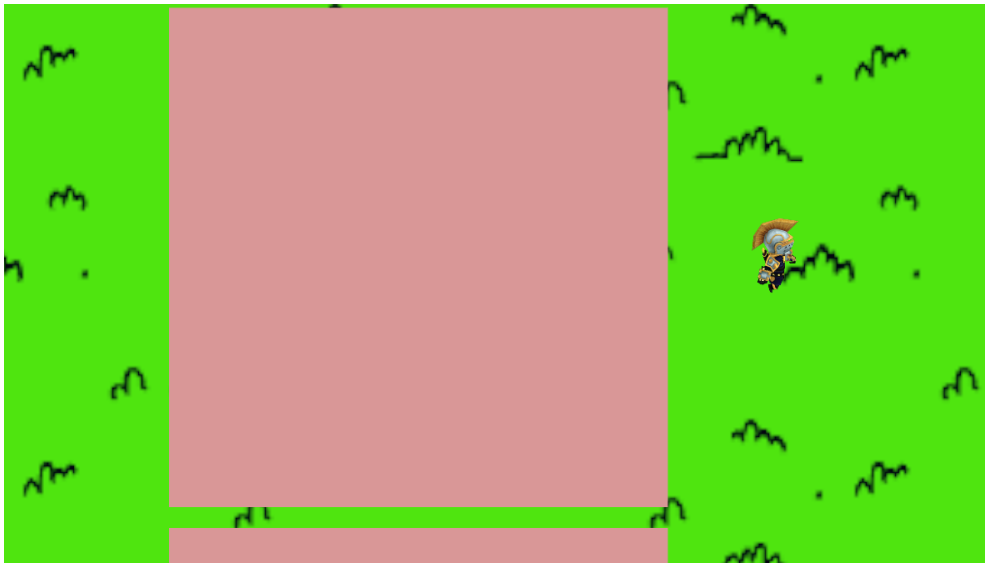
Kuva 17: Häviöruutu ja odotus.

Kuva 18: Voitto! Lopetus ja pisteet kuvassa.

Kuva 19: Halutaanko toinen kenttä? Kuvassa valikko.

Kuva 20: Lopputekstit.

Kuva 21: Kentän reunaa.



Liite 6. Maisemanluonnin pääluuppi C#-koodilla

C#-kielellä ohjelmoidun maisemanluonnin pääluuppi, eli katkelma koodista:

Kuvat 22-25: Maisemanluonnin pääluuppi (4 kuvaa)

```
bool lohikaarmeLoytyy = false;
//looppi x ja y sarake ja rivi
for (int x = 0; x < 41; x++)
{
    for (int y = 0; y < 40; y++)
    {
        char merkki = (kartta[x, y]);
        // sijainti pelin näytöllä
        sijainti = new Vector3(x * 20f, y * 20f - 20f, 0);
        // järvi
        if (merkki == 'L')
        {
            // y lisätään 60 pixeliä  ja tiili tiedostoon vesi instanssi
            sijainti.y = sijainti.y + 60f;
            tiili[x * 40 + y] = Instantiate(vesi);
            tiili[x * 40 + y].transform.position = tiili[x * 40 + y].transform.position + sijainti;
        }
        //vuori
        else if (merkki == 'M')
        {
            sijainti.y = sijainti.y + 100f;
            tiili[x * 40 + y] = Instantiate(vuori);
            tiili[x * 40 + y].transform.position = tiili[x * 40 + y].transform.position + sijainti;
        }
        //metsä
        else if (merkki == 'F')
        {
            sijainti.y = sijainti.y + 80f;
            tiili[x * 40 + y] = Instantiate(puut);
```

```
            tiili[x * 40 + y].transform.position = tiili[x * 40 + y].transform.position + sijainti;
        }
        //lohikäärmeen luola
        else if (merkki == 'D')
        {
            sijainti.y = sijainti.y + 40f;
            lohikaarmeLoytyy = true;
            tiili[x * 40 + y] = Instantiate(Lohik);
            tiili[x * 40 + y].transform.position = tiili[x * 40 + y].transform.position + sijainti;
            kaarmex = tiili[x * 40 + y].transform.position.x;
            kaarmey = tiili[x * 40 + y].transform.position.y;
        }
        //kylä
        else if (merkki == 'H')
        {
            sijainti.y = sijainti.y + 20f;
            tiili[x * 40 + y] = Instantiate(kyla);
            tiili[x * 40 + y].transform.position = tiili[x * 40 + y].transform.position + sijainti;
        }
        //ulkomuuri uudelleen luotuna
        if (x == 0 || x == 39 || y == 0 || y == 40)
        {
            sijainti.y = sijainti.y + 20f;
            tiili[x * 40 + y] = Instantiate(muuri);
            tiili[x * 40 + y].transform.position = tiili[x * 40 + y].transform.position + sijainti;
        }
    }
}
```

No issues found

```
58      //ei löydy lohikäärmettä, laitetaan sattumanvaraisesi yksi
59      if (!lohikaarmeLoytyy)
60      {
61          int dx;
62          int dy;
63          GameObject lo = Instantiate(Lohik);
64          Vector3 tempd = lo.transform.position;
65          do
66          {
67              int rx = rnd.NextInt(20) + 20;
68              int ry = rnd.NextInt(20) + 19;
69              dx = math.abs((int)(rx * 20f / 20));
70              dy = math.abs((int)((ry * 20f + 40f) / 20));
71              //arvotaan kunnes lohikäärme 20<=x<=40, 20<=y ja jos kartta(x,y) ei ole ruoho tai metsä
72          } while (dx > 39 || (dx <= 20) || (dy >= 39) || dy <= 20 && kartta[dx, dy] != 'G' && kartta[dx, dy] != 'F');
73          print(dx + "," + dy);
74          tiili[dx * 40 + dy] = Instantiate(Lohik);
75          lo.transform.position = new Vector3(60f - dx * 20f, 150f - dy * 20f + 60f, -50f);
76          tiili[dx * 40 + dy] = lo;
77          kaarmex = tiili[dx * 40 + dy].transform.position.x;
78          kaarmey = tiili[dx * 40 + dy].transform.position.y;
79          kartta[dx, dy] = 'D';
80      }
81      //päähahmon satunnainen alkusijaintii
82      int px;
83      int py;
84      Vector3 tempV = hahmo.transform.position;
85      do
86      {
```

```
85      do
86      {
87          hahmo.transform.position = tempV;
88          px = rnd.NextInt(10);
89          py = rnd.NextInt(10);
90          //siirretään päähahmo paikalle
91          hahmo.transform.position = hahmo.transform.position + new Vector3((px * 20f + 60f), (-py * 20f - 50f), 0);
92          px = (int)((hahmo.transform.position.x + 720) / 20);
93          py = 7 - (int)((hahmo.transform.position.y) / 20);
94      } while (px < 1 || px > 10 || py < 0 || py > 10 || (kartta[px, py] != 'G' && kartta[px, py] != 'F'));
95      //toistetaan kunnes päähahmo 1<=x<=10 ja 0<y<10 ja ruoho tai metsä
96
```