

Oskari Kiviranta

# SUURTEN KIELIMALLIEN KÄYTTÖ PELEISSÄ

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittelyn koulutus

2024



**Kaakkois-Suomen  
ammattikorkeakoulu**

Tutkintonimike	Tradenomi (AMK)
Tekijä	Oskari Kiviranta
Työn nimi	Suurten kielimallien käyttö peleissä
Toimeksiantaja	Tutors Finland Oy, Kaakkois-Suomen ammatikorkeakoulun Pro-Rak- hanke
Vuosi	2024
Sivut	35 sivua
Työn ohjaaja	Jukka Selin

## TIIVISTELMÄ

Tekoäly on kehittynyt viime vuosina nopeaa tahtia, ja erityisesti suuret kielimallit ovat nousseet suosituksi puheenaiheeksi. Opinäytetyön tavoite on pohdita ja testata näiden kielimallien käyttökelpoisuutta pelien tarinankerronnassa. Esimerkeissä käytetään Unity-pelimoottoria ja kielimallien API-kutsut tehdään Pythonilla.

Python-koodia tai kooditiedostoja ei oletuksena pysty suorittamaan C#:lla, vaan sitä varten pitää hankkia Python.NET-lisäosa käyttäen NuGet-pakkaus-hallitsijaa. Lisäosaa ei pysty suoraan hankkimaan esim. Visual Studion kautta, koska Unity-projektin sisällä sitä ei tulla huomioimaan, sillä se on asennettuna muualle. Unityyn pitää asentaa yhteisön tekemä pakkaus "NuGetForUnity", joka toimii samalla tavalla kuin Visual Studion NuGet. Tätä pakkausta käyttäen asennetut lisäosat toimivat Unityn sisällä. Ennen API-kutsun tekoa pitää vielä miettiä sopiva *prompti* eli ne ohjeet ja avainsanat, jotka annetaan kielimallille ja jonka mukaan takaisin tuleva data muotoillaan. Haluttaessa kielimalli voi palauttaa dataa JSON-muodossa, mutta sille pitää tehdä olioluokat C#:lla siten, että olion muuttujien nimet noudattavat JSON-objektissa olevia vastaavia nimiä.

Tulokset näyttivät lupaavilta tulevaisuutta ajatellen, mutta nykyisin kielimallien käyttö ainakin API:n kautta tuntuu hitaalta, ja hitaus vaan kasvaa, mitä isompaa mallia käyttää. 5–10 sekunnin odotus jokaisella kutsulla helposti rikkoo pelin tarinaan uppoutumisen. Tällä hetkellä, jos haluaa käyttää kielimalleja peleissä, kannattaa pelit rakentaa siten, että kielimallien tekstin luominen tapahtuu ennen pelin käynnistymistä, ettei pelaajan tarvitse odottaa.

**Asiasanat:** tekoäly, kielimalli, Unity, Python, C#

Degree title	Bachelor of Business Administration
Author	Oskari Kiviranta
Thesis title	Using large language models in games
Commissioned by	Tutors Finland Oy, South-Eastern Finland University of Applied Sciences, ProRak project
Time	2024
Pages	35 pages
Supervisor	Jukka Selin

## ABSTRACT

Artificial intelligence has been progressing rapidly in the recent years, and especially large language models have risen to be a popular topic. The objective of the thesis was to consider the usability of these language models in games in storytelling. The examples used the Unity game engine, and the API calls were made with Python.

Python code or files cannot be run by default with C#, so an add-on called Python.NET had to be used using a package manager called NuGet. The add-on could not be installed directly through e.g. Visual Studio, because it was installed somewhere outside the Unity project folder, and therefore Unity does not recognize it. To get them to work inside Unity, it was necessary to get a community-made package "NuGetForUnity". This package works basically the same way as NuGet did in Visual Studio. Before doing API calls, it is still necessary to think of a good prompt, i.e. instructions and keywords, which are given to the language model and by which the outgoing data is formatted. Optionally the language model can also return data in JSON format, but that would require the use of C# classes in which the variable names are the same as in the JSON object.

The results appeared promising for the future, but at present using language models at least through API seemed slow and got slower the bigger the language model used was. Waiting 5–10 seconds with every API call can easily break immersion to the game. Currently, if language models are used in games, they should be built in a way where language model content generation happens before the game starts so that the player doesn't have to wait.

**Keywords:** artificial intelligence, language model, Unity, Python, C#

# SISÄLLYS

1	JOHDANTO.....	5
2	SUURET KIELIMALLIT.....	7
2.1	Mikä tekee malleista suuria? .....	7
2.2	Suurten kielimallien eroavaisuudet ja käyttötarkoitukset.....	8
2.3	Prompt engineeraus .....	11
2.4	Kielimallien API.....	13
3	KIELIMALLIN LIITTÄMINEN PELIMOOTTORIIN .....	13
3.1	Promptin rakentaminen.....	14
3.2	API.....	16
3.3	Kielimalleista palautuvan API-datan luku koodissa.....	19
4	KIELIMALLIEN KÄYTTÖ PELEISSÄ KÄYTÄNNÖSSÄ .....	23
4.1	Toimeksiantajan projekti .....	23
4.2	Oma esimerkki.....	24
5	PÄÄTÄNTÖ .....	31
	LÄHTEET.....	34
	KUVALUETTELO .....	36

## 1 JOHDANTO

Tekoälymallit ovat viime aikoina nousseet yleiseksi keskustelunaiheeksi, koska ne kehittyvät nopeasti. Oma kiinnostus tekoälyyn kasvoi myös näiden mallien mukana vuoden 2023 alussa. Kiinnostus keskittyi enimmäkseen diffusion-pohjaisiin tekoälymalleihin, kun taas kielimallit ovat transform-pohjaisia malleja. Mallit eroavat toisistaan monella tavalla, ja suurin ero on siinä, missä muodossa mallit palauttavat dataa. Diffusion-mallit palauttavat dataa kuva-muodossa, kun taas transform-mallit palauttavat datan tekstimuodossa (Acharya 2023; Merritt 2022).

Oma kiinnostus myös motivoi kokeilemaan muutamia avoimen lähdekoodin kielimalleja, kuten mm. "DialogPT" ja "GODEL", jotka olivat molemmat Microsoftin valmistamia. Malleja käyttäessä tuli harhaluulo, että mallit olisivat olleet uusinta uutuutta, mutta myöhemmin tuli ilmi, että mallit olivat jo melko vanhoja, ja erityisesti DialogPT oli jo useamman vuoden ikäinen sen oman GitHub-sivun mukaan. Kielimallien kokeilun aikana tuli selvästi esille, että mallit tarvitsivat vielä lisää kehitystä. Mallien antamat vastaukset jopa yksinkertaisinkin kysymyksiin tuntuivat olevan täysin satunnaista ja erityisesti edellisten keskusteluiden tallentaminen ja käyttäminen uusissa keskusteluissa aiheutti ongelmia.

Samoihin aikoihin alkoi myös tieto ChatGPT:stä levitä, jonka uusimman version oli silloin ChatGPT3. ChatGPT oli ihan erilainen verrattuna edellisiin testattuihin avoimen lähdekoodin kielimalleihin. Ensimmäinen oma keskustelu ChatGPT:n kanssa oli oikeastaan hieman epämukava, koska tekoäly tuntui niin aidolta, ja oli hieman vaikea uskoa sen olevan tekoäly. Oma kokemus ChatGPT:n käytöstä jäi vähäiseksi, mutta oli kiinnostavaa nähdä, mitä muut pystyivät tekemään sillä. Yksi näistä asioista tapahtui pelimoottorin sisällä, missä pelin NPC:den kaikki repliikit olivat ChatGPT:n luomia. Ideana tämä oli tosi mielenkiintoinen, koska kielimallien käyttäminen peleissä tekee niistä mahdollisesti lähes loputtomasti uudelleenpelattavia, koska täysin samojen repliikkien kuumemisella olisi lähes todennäköisyys kun lottovoitolla.

ChatGPT NPC:den näkemisen jälkeen omaksi haaveeksi tuli nähdä lähivuosina pelejä, joissa kielimalleja hyödynnetään tarinankerronnassa tai muussa

muodossa. Tämän opinäytetyön toimeksianto antoi täydellisen syyn tutkia asiaa enemmän, joka myös mahdollistaa omien päätelmien tekemisen siitä, onko kielimallien käyttö peleissä kannattavaa joko jo nykypäivänä tai ainakin lähitulevaisuudessa.

Tutors Finland Oy on suunnitellut hyödyntävänsä suurten kielimallien käyttöä luomalla peleihin pieniä tarinoita pelaajalla, joissa on eri toimintoja ja joita pelaaja voi valita. Jokainen toiminto tuo eri lopputuloksen, joka voi muuttaa pelin kulkua jollain tavalla. Kuvattu tarinan luonti ja lopputulosten muuttuminen ei itsessään ole kovin mullistavaa, mutta verrattuna ihmiseen, kielimallit voivat luoda periaatteessa loputtomasti näitä pieniä tarinoita. Suuria kielimalleja voisi ajatella kuin ihmisinä, mutta jonka skaalat ovat lähes äärettömiä. Suurien kielimallien käyttö peleissä voi kokonaan mullistaa nykyisen peliteollisuuden.

Kielimallien käytössä on myös negatiivisia asioita ja rajoitteita, kuten kielimallien eettisyys ja kielimallien käytön vaatimat tietokoneen tehot, jos sovellusta ajetaan paikallisesti. Esim. ChatGPT:n pyörittäminen paikallisesti vaatisi satoja gigabittejä videomuistia ja keskusmuistia. Kielimallien vaatimat tehot tosin vaihtelevat aika paljon, mutta mallien vaatimat tehot ovat suoraan verrannollisia siihen, mitä ne pystyvät tekemään/luomaan.

Ongelmana onkin, että tarinoiden tai muun sisällön luominen peleissä vie paljon aikaa ja resursseja. Kielimalleja käyttämällä pelituotanto voisi nopeutua, sillä tarinankertomiseen tarvitsisi sijoittaa vähemmän resursseja. Tämä ei kuitenkaan tarkoita, että kirjoittajat korvattaisiin kielimalleilla. Tarinoiden luonti promptien eli ns. kielimallille annettujen hakusanojen kautta vaatii silti luovuutta ja kokonaisuuden hahmoittamista, jotta tarinoista saa juuri haluttuja. Kirjoittajat voisivat sitten muokata kielimallien luomia tarinoita ja muokata niitä halutunlaisiksi.

Opinäytetyön aihe on kehittää tarinankerrontaa eteenpäin suurten kielimallien kautta käyttämällä joko niiden API:a koodissa, joka luo näitä tarinoita reaaliajassa annettujen promptien kautta, tai suoraan niiden omien käyttöliittymien kautta, jos sellainen on olemassa. API:n käyttäminen on kannattavampaa pienten tekstipätkien luomiseen, ja niiden rakenne ja muoto voivat vaihtua merkittävästi käyttäjän antamien promptien mukaan. Tällaisia ovat esimerkiksi

NPC:n repliikit. Käyttöliittymän kautta tekeminen on kannattavampaa, jos promptin rakenne on lähes samanlainen ja vain muutama avainsana muuttuu promptista toiseen.

Tavoitteena on luoda systeemi, jossa peli luo automaattisesti uusia tarinoita pelissä tapahtuvien asioiden mukaan käyttämällä suuria kielimalleja. Dall-E 3 ja ChatGPT:n yhdistyttyä kuvien luominen tarinoihin liittyen voisi olla myös mahdollista, mikä lisäisi uuden tason tarinankerrontaan. Tulokset tulevat olemaan hyödyllisiä toimeksiantajille, itselle ja todennäköisesti monille muille peliyrityksille, koska kielimallien käyttö saattaa säästää suuria määriä resursseja pelien valmistamisessa.

Luvussa 2 käydään läpi ”suuri kielimalli”-käsitettä, kuinka niitä käytetään, ja esitellään muutamia suosittuja kielimalleja. Luvussa 3 kuvataan prosessia kielimallin luoman datan liittämisestä pelimoottoriin ja kuinka tätä dataa voidaan muotoilla promptissa. Luvussa 4 esitellään, miten kielimallia käyttävän pelin voi tehdä käytännössä.

## **2 SUURET KIELIMALLIT**

Kielimallit ovat tekoälymalleja, jotka perustuvat sanojen tai sanojen järjestyksen käytön todennäköisyyteen. Mallit analysoivat edellisiä tai siihen liittyviä sanojen järjestyksiä ja niiden perusteella lauseen validisuutta, eli kielimallien tapauksessa, muistuttaako sanojen järjestys ihmisten tapaa kirjoittaa tekstiä. (Kapronczay 2022.)

Johdannossa mainittiin jo lyhyesti transform-käsitte, joka on periaatteessa lähes kaikkien kielimallien peruspalikka, jonka ympärille muu kielimalli on rakennettu/koulutettu. Transform-mallien arkkitehtuuri perustuu syvän oppimisen malleihin, jotka esiteltiin vuonna 2017 (What is a transformer model? s.a.).

### **2.1 Mikä tekee malleista suuria?**

Nimi ”suuri kielimalli” perustuu niiden kokoon tai parametrien määrään. Parametrien määrä suurissa kielimalleissa on usein monta sataa miljoonaa ja isoimmat on miljardeissa. Esim. ChatGPT-3 käyttää 175 miljardia parametria ja ChatGPT-4:n on arvioitu käyttävän n. 100 biljoonaa (Raghunath 2023).

Parametrit määrittävät periaatteessa kielimallin käyttäytymisen muuttamalla tekoälyn ymmärrystä kielestä muokkaamalla sitä, kuinka se käsittelee annettua syötettä ja muotoilee ulostuloa. Jokainen parametri on kuin palanen isossa palapelissä, jonka valmis kokonaisuus mahdollistaa ihmismäisen tekstin tuottamisen. (LLM Parameters s.a.)

## 2.2 Suurten kielimallien eroavaisuudet ja käyttötarkoitukset

BERT, ChatGPT, Gemini, Grok ja PaLM ovat kaikki tunnettuja kielimalleja ja samalla yrityksellä voi olla jopa useampi eri kielimalli. Esim. BERT, Gemini ja PaLM ovat Googlen valmistamia (Lutkevich 2020; Ritika 2023; Pichai & Hasabis 2023). Kielimalleja koulutetaan joskus vain yhtä aihealuetta varten, kuten BloombergGPT (BloombergGPT s.a.) ja BioGPT (Newton 2023), jotta tekoälyn tieto keskittyisi vain valittuun dataan. Tämä datan valikointi vähentää tekoälyn tekemiä ns. ”hallusinaatioita”, joissa annetaan virheellistä tietoa. Kielimallien antamat vastaukset perustuvat niiden koulutusdataan, joten jos datassa on ristiriitoja, se voi aiheuttaa ”hallusinaatioita”. (Roy 2023.)

Kielimalleja on tehty eri tarkoituksiin, jotta ne pystyvät tekemään tiettyä tehtävää mahdollisimman tehokkaasti. Koulutusvaiheessa kielimallille syötetty data myös määrittää mitä kielimalli voi tehdä. Kielimalleissa on 3 eri päätyyppiä: autoregressiivinen, autoencoding ja näiden kahden hybridi.

### *Autoregressiivinen*

Malli käyttää edellisiä merkkejä ja sanoja ennustaakseen, mitä tulisi seuravaksi. Malli olettaa, että tulevaisuus muistuttaa menneisyyttä. (Fernando 2022.)

### *Autoencoding*

Autoencoding-malleja on opetettu rakentamaan alkuperäinen lause puutteellisesta lauseesta, josta puuttuu esim. joku sana. Malleja voi käyttää myös tekstin luomiseen, mutta mallien päätarkoitus kielimalleissa on merkin tai lauseen tunnistus. (Boesch s.a.)

Kielimallien suurin ero saattaa olla niiden parametrien määrässä, eli mallin koulutusdatassa. Koulutusdata määrittää, mitä asioita kielimalli ”tietää”, ja se



on yleensä myös suoraan verrannollinen kielimallin tuottaman tekstin ”ihmismäisyyteen”. Asiaa voisi verrata ihmiseen oppimassa uutta kieltä. Mitä enemmän henkilö harjoittelee kieltä, sitä paremmin hän osaa käyttää sitä.

### *ChatGPT*

ChatGPT on OpenAI:n luoma kielimalli, joka on tällä hetkellä todennäköisesti isoin julkinen kielimalli parametrien suhteen. Uusien kielimallien kokoa ja laajuutta on nykyisin vaikea päätellä, sillä yritykset ovat pitäneet parametrien määrät salaisuutena, joten kielimallien oikeaa kokoa voi vaan arvailla. Ensimmäinen ChatGPT-malli julkaistiin marraskuussa 2022, ja siitä lähtien malli on kehittynyt merkittävästi. ChatGPT on koulutettu käyttämällä oppimisen vahvistamista ihmisen palautteen kautta, eli ihmiset ovat testauksessa antaneet tekoälylle joko positiivisen tai negatiivisen merkin kielimallin antamille ulostuloille. (Hetler 2023). Tekoäly on sitten näiden merkkien kautta oppinut joko välttämään huonoksi merkittyjä asioita tai erityisesti valitsemaan hyväksi merkittäviä asioita. ChatGPT pystyy luomaan tekstin lisäksi kuvia ja videoita käyttämällä Dall-E- ja Typeframes-sovelluksia saman käyttöliittymän sisällä (Hetler 2023).

### *PaLM 2*

Googlen valmistama PaLM 2 on paranneltu version vuoden 2022 alkuperäisestä PaLM-kielimallista, joka oli koulutettu 540 miljardilla parametrilla verrattuna PaLM 2:n 340 miljardiin parametriin. Uudessa versiossa on vähemmän parametrejä, koska Google sanoo kielimallin koulutuksen olevan tehokkaampaa ja optimoidumpaa. (Elias 2023.) Google on myös innokkaasti kertonut, kuinka PaLM 2:n käytön voi yhdistää verkon hakujen, sähköpostien, tekstinkäsittelyn ja taulukkojen käytön kanssa (Elias 2023).

### *Gemini*

Gemini on Googlen uusin julkinen kielimalli, joka Googlen mukaan on ensimmäinen kielimalli, joka päihittää ihmisasiantuntijat MMLU-testeissä (Massive Multitask Language Understanding). Geministä on kolme eri versiota, jotka ovat suuruusjärjestyksessä Ultra, Pro ja Nano. Näistä Nano on pienin. (Pichai & Hassabis 2023.)

Gemini on ”anything to anything”-malli, eli kielimallille voidaan antaa dataa tekstinä, videona/kuvina ja äänenä. Malli voi myös palauttaa dataa samoissa muodoissa (Welcome to Gemini era s.a.). Mallin esittelyvideossa näytettiin, kuinka kielimallille voidaan antaa live-videota esim. puhelimella, ja kielimallilta voidaan kysyä, mitä kamerassa näkyvillä materiaaleilla voidaan tehdä.

### Grok

Elon Muskin perustaman xAI:n luoma Grok eroaa muista isojen yritysten kielimalleista sen sensuroinnin määrällä, joka tässä kielimallissa on tosi vähäistä, tai jopa olematonta. XAI kertoo, kuinka Grokilla on ainutlaatuinen ja fundamentaalinen etu, sillä kielimallilla on reaaliaikainen pääsy tietoon maailman tapahtumista X-sovelluksen kautta. Yritys haluaa heidän mukaansa luoda tekoäly työkaluja, jotka ovat hyödyllisiä koko ihmiskunnalle riippumatta taustasta tai poliittisesta näkemyksestä. He haluavat Grokin tarjoavan relevanttia tietoa nopeasti kenelle tahansa. Grok xAI:n mukaan ”vastaa kysymyksiin nokkelasti ja kapinallisesti”. (Announcing Grok s.a.)

Grokin ensimmäinen versio, Grok-0, on koulutettu 33 miljardilla parametrilla ja sen kyvyt ovat verrattavissa LLaMA 2:een, joka on koulutettu 70 miljardilla parametrilla (Announcing Grok s.a.). Grokista on myöhemmin koulutettu uusi versio, Grok-1, joka pystyy kilpailemaan ChatGPT3.5:n ja muiden saman laskentaluokan kielimallien kanssa ja oikeastaan usein myös päihittää ne testien perusteella. Grok-1:n parametrien määrää ei ole julkaistu.

	Grok-0 (33B)	GPT-3.5	Grok-1	PaLM 2	GPT-4	Gemini Ultra
GSM8k	56.8%, 8-shot	57.1%, 8-shot	62.9%, 8-shot	80.7%, 8-shot	92.0%, 8-shot	94.4%, maj1@32
MMLU	65.7%, 5-shot	70.0%, 5-shot	73.0%, 5-shot	78.0%, 5-shot	86.4%, 5-shot	83.7%, 5-shot
HumanEval	39.7%, 0-shot	48.1%, 0-shot	63.2%, 0-shot	-	67.0%, 0-shot	74.4%, 0-shot(IT)
MATH	15.7%, 4-shot	23.5%, 4-shot	23.9%, 4-shot	34.6%, 4-shot	42.5%, 4-shot	53.2%, 4-shot
GSM8k	=	Ylä-asteen matematiikan sanatehtäviä				
MMLU	=	Monitieteisiä monivalintakysymyksiä				
HumanEval	=	Python-koodin viimeistelytehtävä				
MATH	=	Ylä-asteen ja lukion matemaattisia ongelmia kirjoitettuna LaTeX:illa				

Kuva 1. Kielimallien tulokset eri testeissä

Kuvassa 1 voidaan nähdä eri kielimallien pärjäämistä erityyppisissä testeissä. Gemini Ultran testitulokset ovat Googlen antamia ja kaikkien testien testatavastavat eivät ole täysin samat, joten todelliset arvot saattavat vaihdella. Tilastojen alapuolella testien luonteet ovat selitettynä.

### *MoE*

Avoimen lähdekoodin kielimallit ovat yleensä olleet kehityksessä jäljessä verrattuna isojen yritysten kielimalleihin, mutta nyt on keksitty tapa saada jopa avoimen lähdekoodin kielimallit kilpailemaan isojen yritysten mallien kanssa. Tekniikan nimi on *Mixture of Experts* tai *MoE*, jossa käytetään montaa pienempää kielimallia, jotka ovat koulutettu jotain tiettyä aihealuetta varten. Kaikkia malleja ei käytetä yhtä aikaa, vaan kielimalli on koulutettu valitsemaan tietyt kielimallit, jotka pystyvät vastaamaan tai toteuttamaan käyttäjän pyynnön parhaiten. Pyyntö ovat tätä tekniikkaa käyttäessä nopeampia verrattuna saman parametrien määrän sisältämään normaaliin kielimalliin.

Kielimallien pyörittäminen paikallisesti voi viedä paljonkin koneen tehoista ja pelimoottorin pyörittäminen taustalla todennäköisesti veisi loput. Tästä johtuen mielipide kielimallin käytöstä myöhemmissä esimerkeissä nojautui enemmän pilvipalveluna ajettavan ChatGPT:n käyttöön. ChatGPT tarjoaa myös jotain, mitä moni muu kielimalli ei voi tarjota, eli kuvien luomisen. Kuvien luominen API:n kautta ei ole itsessään kovin vaikeaa, mutta se myös tarvitsee aika paljon tietokoneen tehoista, joten jos yhdistää kielimallin ja kuvan luomisen paikallisesti, resurssit loppuvat nopeasti kesken.

### **2.3 Prompt engineeruus**

Suurien kielimallien käytössä prompt engineeruus on keskeisin käsite, sillä suurin osa käyttäjän ajasta tulee kulumaan promptien valitsemisessa. Promptit ovat periaatteessa kuin hakusanoja ja avainsanoja, joita annetaan kielimallille. Kielimalli käyttää näitä hakusanoja datan luomiseen ja muotoiluun. Hakusanat voivat muodostaa myös kokonaisia lauseita. Yksittäisillä prompteilla voi olla suurikin vaikutus lopputulokseen, joten ne kannattaa valita harkiten.

Prompt engineerukseen kuuluu myös muutama käsite promptien rakentamiseen liittyen, jotta tekoäly antaisi juuri halutun asian ja oikeassa muodossa. Tapoihin kuuluu jusk ask, one-/few-shot, output strukturointi ja prompt-persona. Käsitteet perustuvat Sinan Ozdemirin kirjassa esiintyviin vinkkeihin (Ozdemir 2023, 98–106).

### *Just ask*

Nimensä mukaan tässä kysytään haluttu asia suoraan ja mahdollisimman selkeästi (kuva 2). Mitä yksinkertaisempi kysytty asia on, sitä tärkeämpää on antaa selkeät ohjeet. Jos haluaa tekoälyn olevan varmempi siitä, mitä se tekee, niin kannattaa käyttää merkkejä apuna merkkaamalla esimerkiksi, mitä lausetta halutaan kääntää tai mistä tarkistaa kirjoitusvirheet. (Ozdemir 2023, 98.)

```
Translate from English to Finnish.  
  
English: Hey, what day is it today?  
Finnish:
```

Kuva 2. Just ask

### *One-shot/few-shot*

Edellisessä osiossa kerrottiin vähän jo esimerkkiä tästä tavasta, jossa periaatteessa annetaan tekoälylle esimerkkejä, minkälainen sen outputin tulisi olla. Tekoälylle periaatteessa opetetaan jotain, joka tekee muista saman keskustelun ulostuloista parempia tekemään opetettua asiaa. Methodin nimi one-shot ja few-shot viittaa nimensä mukaan annettujen esimerkkien määrään. Käytettyjen esimerkkien määrä vaikuttaa siihen, kuinka hyvin tekoäly oppii/ymmärtää käsitteen.

### *Output strukturointi*

Prompti muokataan siten, että tekoäly antaa sen oikeassa muodossa, kuten esim. JSON:ina. Koodaajien on helpompi hakea dataa tekoälyn antamasta outputista, kun se on helposti käsiteltävässä muodossa. Muodon/rakenteen perustaminen myös vähentää turhan tiedon määrää, kuten esimerkiksi tekoälyn kommentteja.

### *Prompt persoona*

Tekoälylle voidaan antaa just ask -tapaa käyttäen melkein mikä tahansa käyttäjän määrittelemä persoona, jota tekoäly yrittää matkia. ChatGPT4:ssa persoonan voi antaa sen omaa käyttöliittymää käyttäen myös liitteenä tekstitiedostossa.

## 2.4 Kielimallien API

API mahdollistaa kielimallien käytön/kutsun koodin kautta. Tämä mahdollistaa kielimallien käytön niiden omien käyttöliittymien ulkopuolella. API-kutsut tapahtuvat esimerkiksi Pythonin tai Node.js:n kautta, mutta ei ole rajoitettu vain niihin.

OpenAI:lla on virallinen asennettava Python- ja Node.js-lisäosa. Heidän sivuillaan he myös antavat linkkejä yhteisön tekemiin lisäosiin, joita on melkein jokaiselle kielelle vähintään yksi. Myös sovelluksille, kuten Unity ja Unreal Engine 5, on tehty lisäosat. ChatGPT API:n käyttäminen on maksullista ja se tarvitsee aktiivisen OpenAI API-avaimen. (API Reference – OpenAI API s.a.) API:a kutsuessa voi vaihtoehtoisesti määrittää yrityksen, jonka kautta käyttää ChatGPT:tä, jos käyttäjällä on useampi eri tilaus liitettynä omaan käyttäjätiliin (API Reference – OpenAI API s.a.).

API-kutsussa voidaan valinnaisesti määrittää, palautetaanko saapuva teksti JSON muodossa. Kutsussa myös määritetään, mitä versiota ChatGPT:stä käytetään, ja mikä on saapuvien viestien ”lämpötila” eli kuinka luova vastaus on. (API Reference – OpenAI API s.a.) Onnistuneen kutsun jälkeen palautuu JSON-objekti, jossa on tekoälyn vastauksen lisäksi mm. käytetty merkkimäärä (API Reference – OpenAI API s.a.).

API-kutsua käsitellessään ChatGPT ei oletuksena huomioi edellisiä viestejä ollenkaan, tai olisi parempi sanoa, että kielimalli ei muista niitä. Kielimallien muisti ei perustu siihen, että kielimallit pystyisivät itse tallentamaan dataa muistiin, vaan se perustuu kutsun tekemiseen siten, että kutsun yhteydessä annetaan myös lista edellisistä keskusteluista.

## 3 KIELIMALLIN LIITTÄMINEN PELIMOOTTORIIN

Tässä osiossa kuvaillaan prosessia, miten sisällön luominen kielimallien kautta päätyy lopulta pelin käyttöön. Ilman API:n käyttöä prosessi olisi merkittävästi helpompi, sillä koodikielten välistä keskustelua ei tarvitsisi huomioida, koska kaikki tapahtuu suoraan kielimallin omassa käyttöliittymässä.

### 3.1 Promptin rakentaminen

Promptaaminen on todennäköisesti tärkein osa missä tahansa kielimallille tehdyssä pyynnössä, siksi on tärkeää miettiä tarkkaan ja testata prompteja ennen niiden oikeaa käyttöä. Tämä koskee varsinkin API:n käyttöä, sillä API-pyyntöjen tekeminen on maksullista, vaikka käyttäjällä olisikin jo Plus- tai Enterprise-tilaus.

Promptin rakentaminen alkaa kirjoittamalla ensin yksinkertainen kuvaus halutusta ulostulosta, eli tällä kertaa pyydettiin kielimallia luomaan tarina, jossa on kaksi eri vaihtoehtoa, josta toinen on turvallisempi (mutta pienempi vaikutus) ja toinen vaarallisempi (ja isompi vaikutus). Molemmilla vaihtoehdoilla on kaksi eri asiaa, jotka voivat tapahtua. Näistä toinen on pelaajan näkökulmasta hyvä ja toinen on huono. "Consequences"-kohdassa on lueteltuna kaikki mahdolliset vaihtoehdot, joten kielimallin pitäisi valita niiden joukosta vähintään yhden. Kielimallille kerrottiin vielä, että tarinan tulisi muuttua tiettyjen avainsanojen mukaan ja lopuksi ulostulon pitäisi vielä olla JSON-muodossa (kuva 3).

```
Create a story event with 2 possible choices for a medieval game. The first choice is safer and the other is riskier but it also has a bigger effect. Also create a good and a bad outcome for both the safe and the riskier choices. The story is based on playerUnitType, enemyUnitType, territoryType and territoryOwnership. Give the final story event as a valid JSON.

Story_event1:

{
  "parameters": {
    "playerUnitType": (player unit type),
    "enemyUnitType": (enemy unity type),
    "territoryType": (territory type),
    "territoryOwnership": (who owns the territory)
  },
  "introduction": (a short scenario describing the situation),
  "choices": {
    "safeChoice": {
      "description": (description of the safer option),
      "outcomes": {
        "goodOutcome": {
          "description": (good outcome description),
          "consequences": (consequences of the safe choice, could be 1 or more of the following: player_unit_dead, enemy_unit_dead, player_unit_wounded, enemy_unit_wounded, territory_ownership_change or none)
        },
        "badOutcome": {
          "description": (bad outcome description),
          "consequences": (consequences of the safe choice, could be 1 or more of the following: player_unit_dead, enemy_unit_dead, player_unit_wounded, enemy_unit_wounded, territory_ownership_change or none)
        }
      }
    }
  }
},
```

Kuva 3. Perus prompti ja rakenteen esimerkki

Kielimallille annettiin esimerkkipohja, jossa eri kohdat ja avainsanat ovat selitetty lyhyesti. Esim. promptissa on avainsana "playerUnitType", josta ihminen ainakin voisi jo päätellä nimen luonteen, mutta koska tekoäly saattaa käsittää väärin, sen on selitetty olevan pelaajan pelinappulan tyyppi.

Avainsanojen määrittelyyn voi tunnistaa olevan tehty oikein, jos lopputulos ja lopputuloksen kuvaus sopivat keskenään. Esimerkiksi jos lopputuloksen kuvauksessa kerrotaan pelaajan pelinappulan tuhoutuneen, lopputulos pitäisi myös olla "player\_unit\_dead". Eri kohtien selittämisen jälkeen prompttiin lisättiin vielä käsin kirjoitettu esimerkkitapahtuma, joka periaatteessa kertoo kieli-mallille, miltä valmis tarina voisi näyttää (kuva 4).

```
example Story_event2(Slinger, Legionaire, Mountain, Enemy):
{
  "parameters": {
    "playerUnitType": "Slingers",
    "enemyUnitType": "Legionaries",
    "territoryType": "Mountain",
    "territoryOwnership": "Enemy"
  },
  "introduction": "Your rebel slingers are going over a mountain pass. They suddenly spot a band of roman legionaries in the valley down below. You decide to...",
  "choices": {
    "safeChoice": {
      "description": "...Let some rocks loose and try to crush the enemy without being seen.",
      "outcomes": {
        "goodOutcome": {
          "description": "The roman legionaries cannot escape the landslide you caused. Most of them are now injured.",
          "consequences": "enemy_unit_wounded"
        },
        "badOutcome": {
          "description": "Some of your slingers are not in good shape, so they loose grip and fall down the mountain together with the rocks.",
          "consequences": "player_unit_wounded"
        }
      }
    }
  }
},
```

Kuva 4, Käsintehty esimerkki

Viimeisenä kohtana promptissa on kuin eräänlainen funktio, jolle on annettu avainsanoja, joita käyttämällä tarina pitäisi kertoa (kuva 5). Funktiot ei ole uusi asia käytetyssä promptissa ja niiden käyttö näkyy edellisissä kuvissa 3 ja 4. Avainsanoja on neljä, ja niiden tyypit ovat kuvattuna rakenne-esimerkin kuvan 3 "parameters"-kohdassa.

```
Story_event3(Gladiator, Archers, Plain, Gladiator Rebels):
```

Kuva 5, Funktio avainsanojen kanssa

Kuvassa 6 on ChatGPT:n luoma tarina käyttäen edellä esitettyä promptia ja sen sisältämiä annettuja avainsanoja. Kielimalli oli ymmärtänyt avainsanojen tarkoituksen ja niiden kautta oli rakentanut loogiselta kuulostavan tarinan. Eri-tyisesti kannattaa huomioida kuvaustekstin ja itse lopputuloksen olevan kes-kenään sopivia.

```

{
  "story_events":
  [
    {
      "parameters": {
        "playerUnitType": "Gladiators",
        "enemyUnitType": "Archers",
        "territoryType": "Plain",
        "territoryOwnership": "Gladiator Rebels"
      },
      "introduction": "Your gladiator rebels are marching through a plain when they spot a group of enemy archers preparing an ambush. You decide to...",
      "choices": {
        "safeChoice": {
          "description": "...quickly retreat to a nearby forest for cover and avoid the confrontation.",
          "outcomes": {
            "goodOutcome": {
              "description": "Your gladiators successfully evade the archers, avoiding any casualties and maintaining their strength.",
              "consequences": "none"
            },
            "badOutcome": {
              "description": "During the retreat, some of your gladiators are hit by arrows, leading to several injuries.",
              "consequences": "player_unit_wounded"
            }
          }
        }
      }
    }
  ],
}

```

Kuva 6, ChatGPT:n luoma tarina

Promptia kannattaa testata ensin sen omaa käyttöliittymää käyttäen, jos sellainen on olemassa ennen sen käyttämistä API-kutsujen kautta. Kielimallin ulostulosta pystyy helposti huomaamaan, jos promptissa on jotain virheitä tai jotain kohtaa ei ole selitetty tarpeeksi. Kielimallit yleensä pystyvät tunnistamaan, jos jokin teksinpätkä on koodia, milloin kielimalli tulee merkkamaan sen erityisellä tavalla. Tämän merkkauksen puute yleensä merkkaa, että esim. JSON-rakenteessa on jotain viallista.

### 3.2 API

API:n käyttäminen alkaa siitä, että päättää mitä koodikieltä käyttää itse API-kutsujen tekemiseen. Kutsun voi tehdä melkein millä tahansa kielellä, mutta vain Pythoniin ja Node.js:ään on viralliset lisäosat. Muihin koodikieliin on yhteisön tekemiä lisäosia. Tämän opinäytetyön esimerkissä käytetään virallista Python-lisäosaa.

Aiemmin tuli jo esille, että kielimallit eivät itsessään osaa/pysty tallentamaan tai huomioimaan käyttäjän tai kielimallin edellisiä viestejä, vaan edelliset viestit pitää aina lähettää samalla uuden pyynnön kanssa. ChatGPT:n omassa käyttöliittymässä tämä on piilotettu, joten käyttäjä saattaa saada väärän kuvan siitä, miten muisti edellisistä viesteistä oikeasti toimii.

API-kutsun teko Pythonilla on melko yksinkertainen prosessi. Ensin tuodaan (import) OpenAI:n lisäosa "openai", minkä jälkeen sille pitää antaa API-avain.



Avain luodaan OpenAI:n nettisivuilla ja itse avaimen luonti ei ole maksullista, mutta sen käyttö on. Kutsua tehdessä API-avain tarkastetaan, ja jos avaimen tehneellä profiililla ei ole luottotasapainoa (credit), kutsu tuottaa virheen. Virheilmoituksessa näkyy myös luottotasapainon puuttuminen.

Kutsua tehdessä (kuva 7) kielimallille annetaan JSON-muodossa oleva pyyntö, jossa eri prompteille voidaan antaa kaksi eri roolia. Yleisempi on "user", joka nimensä mukaan viittaa käyttäjään, eli käyttäjän tekemät viestit merkataan sillä. Toinen rooli voi vapaaehtoisesti olla "system"-konteksti, joka tullaan huomioimaan haluttaessa jokaisessa tulevaisuuden promptissa, säästäten sen kirjoittamiseen menevää aikaa. Haluttaessa kielimallin edelliset viestit voidaan yhdistää API-kutsuun merkkamalla viestit "assistant"-roolilla.

```
def DoAPIRequest(difficulty, theme):
    openai.api_key = API_KEY

    context_prompt = "Pretend you are hosting a gameshow where the player tries to answer the questions. Write a question that is based on the given theme and

    prompt_json1 = open('Assets\prompt_part1.json', 'r')
    prompt_json2 = open('Assets\prompt_part2.json', 'r')

    messages = [ {"role": "system", "content": context_prompt + " question1:" + prompt_json1.read() + " example question2(0, Food): " + prompt_json2.read()} ]

    messages.append(
        {"role": "user", "content": f"question3({difficulty}, {theme}):"},
    )

    chat = openai.chat.completions.create(
        model="gpt-4-0125-preview", messages=messages
    )

    reply = chat.choices[0].message.content
    return reply
```

Kuva 7. ChatGPT API-kutsu Pythonilla

Konteksti ja käyttäjän antama prompti yhdistetään samaan listaan, joka lähetetään lopuksi kielimallille. Kielimalli palauttaa JSON-objektin, jossa on erilaista dataa, kuten käytettyjen merkkien määrä. Objektista saa tärkeimmän datan, eli itse kielimallin luoman datan viittaamalla "chat.choices [0].message.content". Kielimallin mukaan takaisin tulevan datan vasteaika vaihtelee. GPT3.5-turbolla pyynnön odottaminen kesti n. 5 sekuntia, ja GPT4-turbolla n. 10 sekuntia.

Kuvien luonti ChatGPT:n API:n kautta toimii hyvin samalla tavalla ja siinä käytetään samankaltaista funktiota kutsun tekemiseen. Kutsuvassa funktiossa määritetään perusasiat, kuten käytetty malli, kuvan prompti, koko ja laatu

(kuva 8). Kuvien koot voivat olla vain tiettyjä kokoja, joten ne kannattaa tarkistaa dokumentaatiosta ennen kutsujen tekoa. Kuvien määrä on myös yksi muuttujista, mutta vain Dall-E 2 voi luoda enemmän kuin yhden kuvan kerrallaan. Dall-E 3-mallia käyttäen voi myös vapaaehtoisesti nostaa kuvan laatua asettamalla *quality*-muuttujan "hq":ksi.

```
def GenerateImage(prompt):  
    openai.api_key = API_KEY  
    response = openai.images.generate(  
        model="dall-e-3",  
        prompt=prompt,  
        size="1024x1024",  
        quality="standard",  
        n=1,  
    )  
    image_url = response.data[0].url  
    res = requests.get(image_url)  
    img = Image.open(BytesIO(response.content))  
    return img
```

Kuva 8. Kuvien luonnin API

Kutsu palauttaa oletuksena kuvien URL-osoitteen, mutta voi myös olla b64\_json-muodossa. Kuvan URL palautetaan sitä kutsuvaan C#-funktioon, jossa se tallennetaan ja asetetaan *Image*-komponenttiin. Kuvassa 9 on luotu kuva käyttäen Dall-E 3, jossa promptina oli "spongebob driving a racing car with shades on".



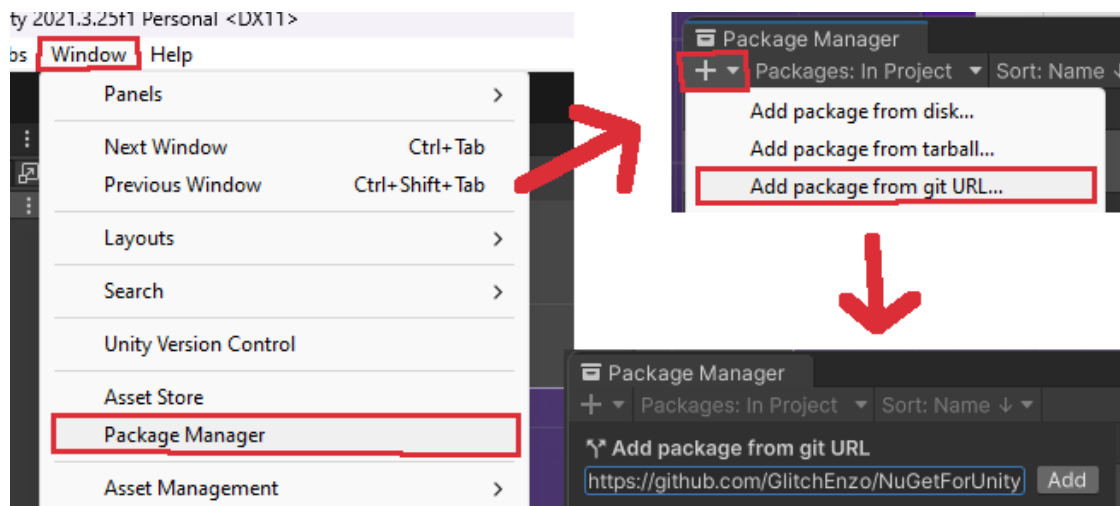
Kuva 9. Valmis kuva

Yhden 1024x1024 kuvan luonti käyttäen Dall-E 3 kesti kauemmin kuin normaali pyyntö käyttäen GPT-4, eli n. 12 sekuntia. Odotus on melko pitkä, joten jos kuvien luontia aikoo käyttää pelissä, ne kannattaa tehdä muun logiikan taustalla

### 3.3 Kielimalleista palautuvan API-datan luku koodissa

Ennen datan lukua koodissa pitää Pythonin koodi yhdistää käytetyn pelimoottorin pääkoodikielen kanssa, eli tässä tapauksessa C#-kielen kanssa, jotta palautuvaa dataa pystyy lukemaan ja käsittelemään Unityn alaisuudessa C#:ssa. Visual Studioon kautta asennetut NuGet-lisäosat eivät tule toimimaan, koska asennetut lisäosat ovat Unity-projektin ulkopuolella.

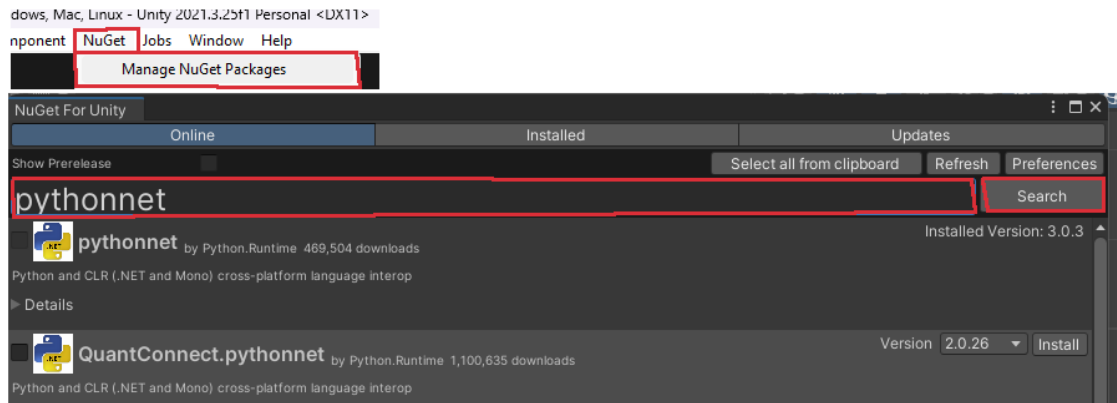
Unityyn pitää ensin asentaa NuGetForUnity-lisäosa Unityn oman lisäosien hallintatyökalun (Package Manager) kautta. Package Managerin saa auki Unityn Window-valikosta valitsemalla *Package Manager*. Package Managerin yläkulmassa olevaa plussaa painamalla valitaan *Add package from git URL*. Avautuvaan laatikkoon annetaan linkki lisäosan GitHub-sivulle. Lopuksi painetaan *Add*, joka asentaa lisäosan (kuva 10).



Kuva 10. NuGetForUnity asennus

Asentamisen jälkeen Unity pitää käynnistää uudelleen, minkä jälkeen Unityn ylävalikossa on uusi valikko *NuGet*. NuGet mahdollistaa koodipuolen lisäosien asentamisen. NuGetia tarvitaan Python.NET-lisäosan asentamiseen, joka mahdollistaa Python-koodin kirjoittamisen ja myös Python-tiedostoissa olevien koodien suorittamisen C#-kooditiedostojen sisällä.

Asennus alkaa valitsemalla NuGet-valikosta *Manage NuGet Packages*. Tämä avaa uuden ikkunan, josta voi etsiä lisäosia, tarkistaa asennetut lisäosat, ja päivittää asennettuja lisäosia. Ikkunassa on hakukenttä, johon kirjoitetaan ”pythonnet”. Hakunapin painamisen jälkeen asennetaan ylimmäinen osuma. Lisäosa oli jo asennettuna, joten asennusnappi puuttui, mutta se näyttää täysin samalta, kuin sen alapuolella olevassa lisäosassa (kuva 11).



Kuva 11. Python.NET asennus

Python.NET ollessa asennettuna, sitä voidaan käyttää koodissa. Lisäosaa kutsutaan *Python.Runtime*-komennolla ja sitä käyttäessä pitää näyttää, missä *PythonXYZ.dll* on paikallisesti asennettuna tietokoneella. Se sijaitsee normaalisti samassa paikassa, missä muukin Python on asennettuna. Lisäosa ”käynnistetään” käyttämällä *PythonEngine.Initialize()*-komentoa ja sitä pitää kutsua ennen muiden toimintojen käyttöä. *PythonEngine*:n kutsuminen toisen kerran aiheuttaa virheen, joten se kannattaa laittaa *Start*-funktion sisään, koska sen koodi suoritetaan vain kerran heti pelin käynnistyessä. Seuraavaksi määritellään *using (Py.GIL()) { }*, jonka sisälle voidaan kirjoittaa koodia Pythonilla tai kutsua muita Python-tiedostoja ja niiden funktioita (kuva 12). Jos haluaa kutsua muita Python-tiedostoja, ne pitää olla ”C:\Users\Username\AppData\Local\Programs\Python\PythonXYZ\Lib”-kansiossa (tai missä tahansa Python on asennettuna), jotta *Py.Import* pystyy löytämään ne.

```

using Python.Runtime;
| reference
public void FetchDataFromAI()
{
    Runtime.PythonDLL = @"C:\Users\████████\AppData\Local\Programs\Python\Python310\Python310.dll";
    PythonEngine.Initialize();

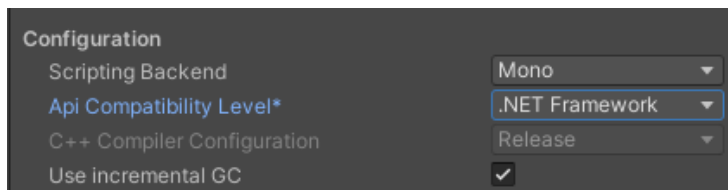
    using (Py.GIL())
    {
        dynamic pythonModule = Py.Import("gptapi");
        var output = pythonModule.DoAPIRequest();

        Debug.Log(output);
    }
}

```

Kuva 12. Python.NET käyttö

Koodi käyttää *dynamic*-tyyppistä muuttujaa, joka oletuksena tulee aiheuttamaan virheen. Käyttäjän pitää vaihtaa projektin asetuksista *pelaaja*-kohdan *Api Compability Level*-määrittely joko .NET 4.x tai .NET Frameworkiksi. Molemmat ovat periaatteessa sama asia, mutta uudemmissa Unityn versioissa se on nimetty .NET Frameworkiksi (kuva 13).



Kuva 13. Api Compability Level

Ennen seuraavaa vaihetta JSON:lle pitää tehdä rakenne, jota se noudattaa käyttäen luokkia. Luokkien muuttujien nimet pitää olla täysin samat, kuin mitkä niiden nimet ovat tehdyssä promptissa. Luokkien parent, eli kaikkien luokkien vanhempi, pitää olla taulukkona (arrayna) tai listana, jotta rakenne toimii (kuva 14).

```

[System.Serializable]
2 references
public class APIData
{
    public APIDataObject[] data;
}

[System.Serializable]
1 reference
public class APIDataObject
{
    public string difficulty;
    public string theme;
    public string question;
    public ObjectAnswers answers;
}

[System.Serializable]
1 reference
public class ObjectAnswers
{
    public string answer1;
    public string answer2;
    public string answer3;
    public string answer4;
}

```

Kuva 14. APIData-luokan JSON rakenne

Pythonin puolella kielimalli palauttaa lopulta JSON-objektin, ja sieltä saa haettua tarvittavan datan aiemmin API-luvussa kuvatulla tavalla. Kutsuttu Python-funktio palauttaa dataa, joka kuvan 15 esittämällä tavalla annetaan "incJson"-muuttujalle. Muuttujan data muutetaan seuraavaksi JsonUtilityä käyttäen "APIData"-rakenteen mukaiseksi JSON:ksi, mistä voidaan helposti hakea tarvittavat tiedot (kuva 15).

```

using (Py.GIL())
{
    dynamic pythonModule = Py.Import("gptapi");
    var output = pythonModule.DoAPIRequest(_diff, _theme);
    incJson = output;
}

APIData apiData = JsonUtility.FromJson<APIData>(incJson);

List<string> answerList = new List<string>
{
    apiData.data[2].question,
    apiData.data[3].answers.answer1, // correct
    apiData.data[3].answers.answer2,
    apiData.data[3].answers.answer3,
    apiData.data[3].answers.answer4
};

```

Kuva 15. API:sta palautuvan datan luku

Pythonista palautuvan apidatan lukeminen koodissa on melko monimutkainen prosessi, mutta onneksi se tarvitsee tehdä vain kerran. Pythonin yhdistäminen koodiin mahdollistaa myös muiden voimakkaiden tekoälysovellusten käytön, kuten TTS ja puheen tunnistus.

## 4 KIELIMALLIEN KÄYTTÖ PELEISSÄ KÄYTÄNNÖSSÄ

Tässä luvussa esitellään, miltä kielimallien käyttö pelien tekemisessä käytännössä näyttää. Ensin käydään läpi päätoimeksiantajan projekti, jossa käytetään ChatGPT:n omaa käyttöliittymää tarinoiden luomiseen. Niissä on eri vaihtoehtoja ja lopputuloksia. Lopuksi esitellään vielä oma esimerkki, jossa käytetään kielimallin API:a peliprojektista suoraan reaaliajassa

### 4.1 Toimeksiantajan projekti

Päätoimeksiantajan peliprojektissa päätettiin käyttää ChatGPT:tä tarinoiden luomiseen, mutta API-kutsujen ei nähty olevan tarpeellisia, erityisesti koska Enterprise-tilauksen ulkopuolella API-kutsuilla on raja paljonko kutsuja voi tehdä kolmen tunnin välein. Jos halutaan luoda paljon tarinoita lyhyessä ajassa, niin se on kannattavampaa ChatGPT:n oman käyttöliittymän kautta. Tarinat luotiin JSON-muodossa ja ne yhdistettiin samaan tiedostoon, jotta niiden lukeminen on helpompaa koodissa.

JSON-tiedostojen luku toimii C#:n sisäänrakennetulla JsonUtility:lla sen tarjoaminen valmiiden luokkien kautta, missä jokaiselle JSON objektille pitää olla oma luokka (kuva 16). Tässä tapauksessa luokkia tulee olemaan kaiken kaikkiaan seitsemän. Luokkien käyttämisen voi jättää mahdollisesti väliin, jos käyttää ulkopuolisia lisäosia JSON-tiedostojen lukemiseen.

```
[System.Serializable]
public class StoryEvents {
    public StoryEvent[] story_events;
}

[System.Serializable]
public class StoryEvent {
    public Parameters parameters;
    public string introduction;
    public Choices choices;
}

[System.Serializable]
public class Parameters {
    public string playerUnitType;
    public string enemyUnitType;
    public string territoryType;
    public string territoryOwnership;
}
```

Kuva 16. JSON-tiedoston luku C#:lla

Kun luokat ovat valmiita, tapahtumia voidaan lukea yksinkertaisella koodilla, jossa ensin muutetaan tapahtumien sisältämä tekstitiedosto oikeaan muotoon käyttäen JsonUtilitya. Kaikki tapahtumat käydään läpi ja jos ne täsmäävät annettuja avainsanoja, ne lisätään väliaikaiseen listaan (kuva 17).

```
public void FetchSpecificEvent(string playerUnit, string enemyUnit, string territoryType, string territoryOwner) {
    StoryEvents storyEventsInJson = JsonUtility.FromJson<StoryEvents>(jsonFile.text);

    foreach (StoryEvent storyEvent in storyEventsInJson.story_events) {
        if (storyEvent.parameters.playerUnitType == playerUnit
            && storyEvent.parameters.enemyUnitType == enemyUnit
            && storyEvent.parameters.territoryType == territoryType
            && storyEvent.parameters.territoryOwnership == territoryOwner) {

            fetchedEvents.Add(storyEvent);
        }
    }
}
```

Kuva 17. Tapahtumien haku JSON-tiedostosta

Väliaikaisesta listasta voisi esim. tarkistaa, onko tapahtuma jo ennen käyty läpi ja oikeastaan se olisi tämän projektin looginen seuraava vaihe. Projekti on vielä keskeneräinen, joten tässä vaiheessa listasta vain valitaan satunnaisesti yksi tapahtuma, ja se kopioidaan currentEvent-muuttujaan (kuva 18).

```
if (fetchedEvents.Count > 0) {
    int randomInt = Random.Range(0, fetchedEvents.Count);
    currentEvent = fetchedEvents[randomInt];
}
else { Debug.Log("No events found"); }
```

Kuva 18. Väliaikaisen tapahtumalistan käsittely

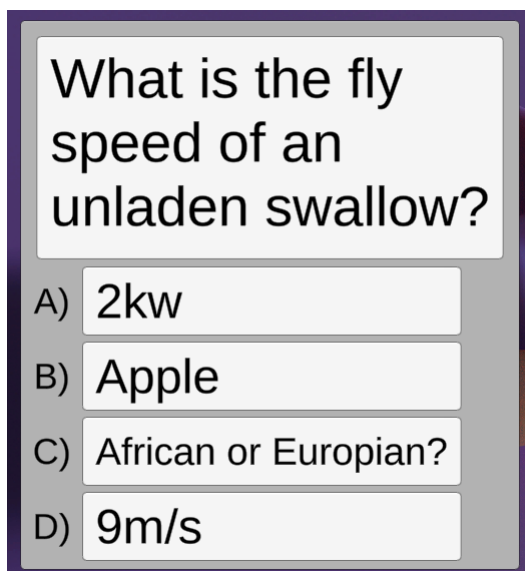
Tämän jälkeen kielimalliosuus on periaatteessa valmis, koska kielimalli täytti kaikki asetetut vaatimukset, eli se loi tarinoita koodilla luettavassa JSON-muodossa. Projekti oli prototyyppi, joten tarinat luotiin yksitellen ja lopuksi yhdistettiin samaan tiedostoon, mutta varsinaisessa projektissa kielimalli voisi luoda monta tarinaa kerralla, mikä nopeuttaisi prosessia myös huomattavasti.

## 4.2 Oma esimerkki

Ensimmäisenä loogisena vaiheena pelin tekemisessä on miettiä pelin päämekaniikoita, eli millainen peli halutaan luoda. Tässä tapauksessa halutaan luoda peli, jossa kielimallin käyttö on mahdollisimman iso osa pelin omaa tarinankerontaa. Tietovisapelit ovat suosittuja ja "Haluatko miljonääriksi?"-peliohjelmia



on jopa näytetty televisiossa. Tästä peliohjelmasta löytyi idea tehdä samankaltainen peli, mutta ilman apureita kuten 50–50 tai kysy kaverilta. Ennen pelin alkua pelaajalta kysytään teema ja vaikeustaso, joiden perusteella kielimalli luo kysymyksen ja neljä vastausta, joista yksi on oikein. Idean saatua peli alettiin rakentamaan Unityssä.



Kuva 19. Esimerkki tietovisapeli

Peli on tässä vaiheessa Unityn ja koodin osalta muuten valmis, mutta se tarvitsee vielä kielimallin luomaa dataa. Data pitää palauttaa siten, että se noudattaa koodissa olevaa muotoa. Datan muoto on lista, jossa kysymys ja vastaukset ovat yhdessä ja kysymys on listan ensimmäinen, eli [0] objekti. Oikea vastaus on kysymyksestä heti seuraava (kuva 20).

```
List<string> testanswerList = new List<string>
{
    "What is the fly speed of an unladen swallow?",
    "African or European?", // correct
    "2kw",
    "Apple",
    "9m/s"
};
```

Kuva 20. Kysymyksen ja vastauksien lista

Listan objektien järjestys on tärkeä, koska koodin muuttujien arvot haetaan aina samoista kohdista, joten vastausta olisi vaikeampi löytää, jos se on sattunnaisesti lisätty listaan. Tällä tavalla koodia ei tarvitse muuttaa ollenkaan, ja koodi samalla pysyy yksinkertaisena (kuva 21).

```
question.text = testanswerList[0];
correctAnswer = testanswerList[1];
```

Kuva 21. Muuttujia

Objektien järjestys luodaan promptissa, mikä on seuraava vaihe. Promptissa pitää mainita kaikki tarpeelliset kohdat, ja ne on hyvä myös selittää, jotta kieli-malli olisi varmempi eri kohtien luomisessa. Pelissä tullaan käyttämään vai-keustasoa välillä 0–10 sekä teemaa, johon kysymys liittyy, joten ne pitävät olla myös promptissa. Toimeksiantajan esimerkin kaltaisesti prompti on funk-tiomuodossa, jossa "question1" toimii kuin funktiona (kuva 22).

```
question1:
{
  "data": [
    { "difficulty": "(difficulty of the question)" },
    { "theme": "(the theme of the question and answers)" },
    { "question": "(a question that is related to the theme and is based on the difficulty)" },
    {
      "answers": {
        "answer1": "(the correct answer)",
        "answer2": "(a wrong answer)",
        "answer3": "(a wrong answer)",
        "answer4": "(a wrong answer)"
      }
    }
  ]
}
```

Kuva 22. Tietovisapeli promptin selitys

Kielimallille on hyvä antaa myös one-take/few-take-esimerkki, eli miltä valmis ulostulo voisi näyttää. Mitä enemmän esimerkkejä antaa, sitä varmempi kieli-mallin pitäisi olla. Tämän projektin prompti on niin yksinkertainen, että yksi esi-merkki riittää. Prompti-esimerkissä käytetään myös funktiota, joka on tässä ta-pauksessa "question2" ja sille on annettu kaksi muuttujaa, joista ensimmäinen on vaikeustaso ja toinen kysymyksen aihe alue tai teema (kuva 23).

```
example question2(0, Food):
{
  "data": [
    { "difficulty": "0" },
    { "theme": "Food" },
    { "question": "What color are bananas?" },
    {
      "answers": {
        "answer1": "Yellow",
        "answer2": "Purple",
        "answer3": "White",
        "answer4": "Red"
      }
    }
  ]
}
```

Kuva 23. Tietovisapeli promptin esimerkki

Promptin lopussa annetaan vielä funktio, joka kielimallin pitäisi tehdä loppuun käyttäen annettuja muuttujia, eli kielimallin halutaan luovan kysymyksen vaikeustasolla 0, ja jonka teemana on matematiikka (kuva 24).

```
question3(0, Math):
```

Kuva 24. Promptin viimeinen osa

Kuvassa 25 käytetään luotua promptia testaamaan, kuinka se toimii käytännössä käyttäen GPT-3.5-kielimallia. Kielimalli osasi yhdistää funktion antamat muuttujat kysymyksen luomiseen ja oikea vastaus on myös heti ensimmäisenä.

```
"data": [
  { "difficulty": "0" },
  { "theme": "Math" },
  { "question": "How many fingers do most people have on one hand"
  {
    "answers": {
      "answer1": "5",
      "answer2": "8",
      "answer3": "10",
      "answer4": "3"
    }
  }
}
```

Kuva 25. Promptin testaus ChatGPT3.5 vaikeustasolla 0

Kuvassa 26 kokeiltiin samaa promptia käyttäen Googlen Bardia, joka tässä tapauksessa käytti PaLM2 kielimallia vastaamiseen. Ulostulon teema oli oikein, mutta vaikeustaso näyttää hieman oudolta. Alkuluvut tuntuvat olevan aiheena liian vaikeita vaikeustasolle 0, koska ainakaan Suomessa niitä ei opeteta ollenkaan peruskoulussa.

```
"data": [
  { "difficulty": "0" },
  { "theme": "Math" },
  { "question": "What is the first prime number?" },
  {
    "answers": {
      "answer1": "2",
      "answer2": "3",
      "answer3": "5",
      "answer4": "7"
    }
  }
}
```

Kuva 26. Promptin testaus PaLM2 vaikeustasolla 0

PaLM2-kielimallia käyttäen tehtiin vielä toinenkin testi, mutta vaikeustasolla 10. Kysymys oli piin likiarvon pyöristys sadasosiin, mikä ei todellakaan ole eksperttien tasolla, joten PaLM2-kielimallilla näyttää olevan ongelmia vaikeustason ymmärtämisessä (kuva 27).

```
{ "difficulty": "10" },
{ "theme": "Math" },
{ "question": "What is the value of pi (π) to the nearest hundredth?" },
{
  "answers": {
    "answer1": "3.141",
    "answer2": "3.142",
    "answer3": "3.140",
    "answer4": "3.143"
```

Kuva 27. Promptin testaus PaLM2 vaikeustasolla 10

Sama testi tehtiin myös ChatGPT3.5-kielimallin kanssa, ja vaikka kysymys ei ole täysin eksperttien tasolla, se on kuitenkin vaikeampi kuin piin likiarvon pyöristys. Saattaa olla, että vaikeustasoa ei ollut selitetty tarpeeksi tarkkaan, mikä aiheuttaa epätarkkuuksia kysymyksien vaikeudessa (kuva 28).

```
"data": [
  { "difficulty": "10" },
  { "theme": "Math" },
  { "question": "What is the value of the mathematical constant
  {
    "answers": {
      "answer1": "2.7182818284",
      "answer2": "3.1415926535",
      "answer3": "1.6180339887",
      "answer4": "0.5772156649"
```

Kuva 28. Promptin testaus GPT3.5 vaikeustasolla 10

Testi tehtiin vielä eri teemoilla. Ongelma saattaa olla pelkästään matematiikan kanssa, koska tiede-aiheisessa kysymyksessä molemmat kielimallit antoivat vaikeustasoon nähden sopivan kysymyksen, joista GPT3.5 näytti antavan helpomman (kuva 29).

The image shows two JSON snippets side-by-side. The top snippet is for PaLM2 and the bottom for GPT3.5. Both snippets represent a question with a difficulty level of 0. The PaLM2 snippet shows four possible answers: Gravity, Electromagnetism, Strong Nuclear Force, and Weak Nuclear Force. The GPT3.5 snippet shows four possible answers: Sun, Moon, Mars, and Venus.

```

"data": [
  { "difficulty": "0" },
  { "theme": "Science" },
  { "question": "What is the name of the force that keeps us on the ground?" },
  {
    "answers": {
      "answer1": "Gravity",
      "answer2": "Electromagnetism",
      "answer3": "Strong Nuclear Force",
      "answer4": "Weak Nuclear Force"
    }
  }
]

```

PaLM2

```

"data": [
  { "difficulty": "0" },
  { "theme": "Science" },
  { "question": "What is the name of the closest star to Earth?" },
  {
    "answers": {
      "answer1": "Sun",
      "answer2": "Moon",
      "answer3": "Mars",
      "answer4": "Venus"
    }
  }
]

```

GPT3.5

Kuva 29. Kielimallien vertailu vaikeustasolla 0

Samalla aiheella vaikeustasolla 10 molemmat kielimallit antoivat vaikeammat kysymykset, mutta PaLM2-kielimallin kysymys on selvästi helpompi ja ei todellakaan ole ekspertin tasoa ja muistuttaa enemmän lukion tasoa. GPT3.5-kielimallin kysymyksen vaikeustasoa on vaikeampaa arvioida, sillä tieto siitä ei ole yhtä yleistä (kuva 30).

The image shows two JSON snippets side-by-side. The top snippet is for PaLM2 and the bottom for GPT3.5. Both snippets represent a question with a difficulty level of 10. The PaLM2 snippet shows four possible answers: Atom, Proton, Neutron, and Quark. The GPT3.5 snippet shows four possible answers: 6.62607015 x 10^-34 m^2 kg / s, 3.1415926535, 9.81 m/s^2, and 299,792,458 m/s.

```

"question": "What is the smallest known particle?"
"answers": {
  "answer1": "Atom",
  "answer2": "Proton",
  "answer3": "Neutron",
  "answer4": "Quark"
}

```

PaLM2

```

"question": "What is the precise value of Planck's constant in SI units?"
"answers": {
  "answer1": "6.62607015 x 10^-34 m^2 kg / s",
  "answer2": "3.1415926535",
  "answer3": "9.81 m/s^2",
  "answer4": "299,792,458 m/s"
}

```

GPT3.5

Kuva 30. Kielimallien vertailu vaikeustasolla 10

Kuvan 31 Python-kutsussa kielimallille on ensin annettu konteksti-promptti, jossa selitetään viestien asetus, eli tässä tapauksessa kielimalli toimii kuin tietovisaohjelman juontajana, jonka tarkoitus on antaa käyttäjälle aiheeseen ja vaikeustasoon liittyen kysymys, ja neljä vastausta. Kontekstissa on myös mainittu vaikeustason olevan 0–10, josta vaikeustaso 0 on jotain, mitä pieni lapsikin voisi tietää. Vaikeustaso 10 on taas ekspertin tasoa. Kutsussa käytetään

myös kahta eri JSON-tiedostoa, joista ensimmäinen on kuvassa 22 oleva koh-tien selitys JSON:n sisällä. Toinen tiedosto näyttää kuvan 23 tapaan esimerk-kiä, miltä palautuva data voisi näyttää. Konteksti prompti ja JSON-tiedostot yh-distetään samaan objektiin, minkä on merkattu olevan taustatietoa asettamalla rooliksi systeemi. Objekti tullaan sitten aina huomiomaan seuraavien promptien kanssa. Käyttämällä *open()*-funktioita kannattaa ottaa huomioon, että tiedostopolku josta koodi etsii tiedostoja, on oletuksena projektin parent-tai pääkansio.

Lopuksi tehdään vielä prompti, jonka rooli on merkattu käyttäjäksi (kuva 31). Prompti käyttää hyväkseen kontekstin funktiomaista rakennetta, jossa on käy-tetty funktioita "question1" ja "question2" harkiten, jotta käyttäjäksi merkattu viesti voi vain olla "question3" eri muuttujien kanssa, ja kielimalli ymmärtää siitä, mitä sen pitää tehdä.

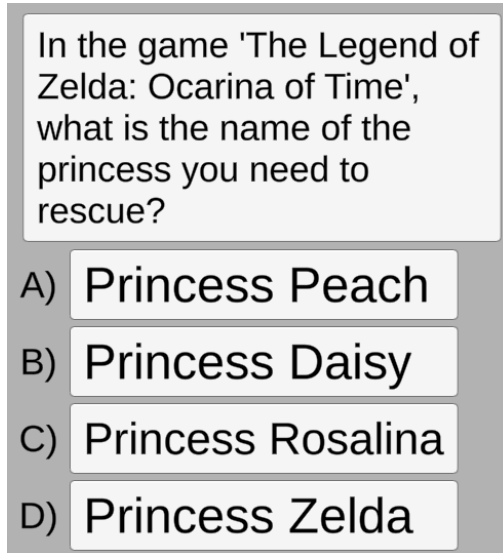
```
def DoAPIRequest(difficulty, theme):
    openai.api_key = API_KEY
    context_prompt = "Pretend you are hosting a gameshow where the player tries to answer the questions. Write a question that is based on the given theme and
    prompt_json1 = open('Assets\prompt_part1.json', 'r')
    prompt_json2 = open('Assets\prompt_part2.json', 'r')
    messages = [ {"role": "system", "content": context_prompt + " question1:" + prompt_json1.read() + " example question2(0, Food): " + prompt_json2.read()} ]
    messages.append(
        {"role": "user", "content": f"question3({difficulty}, {theme}):"},
    )
    chat = openai.chat.completions.create(
        model="gpt-4-0125-preview", messages=messages
    )
    reply = chat.choices[0].message.content
    return reply
```

Kuva 31. Promptin syöttäminen kielimallille pythonilla

Konteksti ja varsinainen prompti yhdistetään samaan listaan, joka periaat-teessa toimii kielimallin muistina. Keskustelumuoitoisessa kielimallin käytössä listaan lisättäisiin jatkuvasti kaikki käyttäjän tekemät promptit ja kielimallin pa-lauttavat datat, mutta tällaisen muistin käyttäminen lisää merkittävästi käytet-tyjen merkkien määrää kutsua tehtäessä, joten sitä ei käytetä tässä projek-tissa. Muistin käyttäminen ei muutenkaan anna mitään hyötyä tämänkaltaisessa projektissa, jossa edelliset palautuvat datat eivät juurikaan liity toisiinsa millään tavalla.

Lopulta pyyntö lähetetään eteenpäin. Siinä mainitaan kielimallin versio ja an-netaan myös lista kaikista prompteista. Kielimalli palauttaa JSON-objektin, minkä jälkeen itse kielimallin palauttava data asetetaan *reply*-muuttujalle. Reply palautuu lopulta takaisin sitä kutsuvaan C# koodiin.

Koodi pystyy nyt tekemään API-kutsuja ja käyttämään kielimallin palauttamaa dataa pelimoottorin sisällä reaaliajassa. Kuvassa 32 näkyy esimerkki pelin toiminnasta. Vaikeustasoksi asetettiin 5 ja aiheeksi videopelit.



Kuva 32. Kielimallin datan käyttö reaaliajassa

Pelin koodi tekee API-kutsun, hakee tarvittavan datan kielimallin palauttamasta JSON-objektista ja muuttaa pelin tekstielementtejä niiden mukaan melko sulavasti. Ainoa sulavuutta estävä asia on API:n odotus, eli n. 10 sekuntia käyttäessä GPT4-kielimallia.

## 5 PÄÄTÄNTÖ

Kielimallien tai tekoälyn käyttö kokonaisuudessaan voi mullistaa tavan, miten pelit toimivat tulevaisuudessa. Pelien tarinankerronta ja dialogilinjat voivat olla lähes kokonaan kielimallien luomia, missä kielimallille pitää vain antaa tarinan ydinkohdat, joiden perusteella kaikki rakennetaan.

Tällä hetkellä teknologia tuntuu ainakin pari vuotta liian aikaiselta, jotta kielimallien käyttö voisi sulavasti integroitua pelien tarinankerrontaan tai sisällön luontiin. Kielimallien käyttöä rajoittaa niiden viemät tehot ja hyvän kielimallin pyörittäminen paikallisesti ei ole mahdollista ainakaan 99 % tietokoneista tai konsoleista. Tällä hetkellä kielimallien käyttö rajautuu enimmäkseen API:n käyttöön, mikä tarkoittaa myös, että pelit täytyvät olla nettiin yhdistettynä. API-kutsun teko myös kestää jonkin aikaa, mikä tuntuu oudolta esim. NPC:n kanssa keskustelussa, jossa pelaaja odottaa nopeita vastauksia.

Kielimallien ymmärrys abstrakteimmista asioista, kuten vaikeustaso, on myös välillä melko heikkoa. Esim. 3.2 lukua tehdessä GPT3.5 antoi samaa kysymystä monta kertaa, vaikka vaikeustaso nousi usean kerran. Teemaksi oli asetettu maantieto ja vaikeustaso oli 0. GPT3.5 antoi kysymykseksi ”Mikä on Ranskan pääkaupunki?”, mikä on melko sopiva vaikeustason 0 kysymykseksi, mutta vaikeustason noustessa ainoastaan kysymyksen sanat ja sanojen järjestys muuttui. GPT4:n kanssa ei tullut vastaan samaa ongelmaa, joten kielimallien kehityksen pystyy näkemään.

Esimerkkiprojektit menivät hyvin, mutta toimeksiantajan projektissa kielimallien käyttö jäi melko vähäiseksi, mutta projekti oli vielä muutenkin hyvin keskeneräinen. Omassa projektissa kielimallin käyttö oli sisäänrakennettu pelin koodiin ja kaikki suunnitellut asiat saatiin toimimaan. Seuraava mahdollinen vaihe projektille olisi rakentaa muita toiminnallisuuksia olemassa olevan päälle. Käytetyllä GPT:llä on myös mahdollisuus luoda kuvia, joten kielimalli voisi luoda kysymykseen liittyvän kuvan. Kuva ei saisi kuitenkaan olla liian kuvaava, ettei kysymyksen vastausta vahingossa anneta pelaajalle. Pythonin ollessa liitettynä koodiin, voisi sitä käyttämällä myös liittää peliin muita tekoälyä käyttäviä lisäosia, kuten text-to-speech, joka lukisi ääneen kysymyksen ja vastaukset.

Käytettyä kielimallia voisi myös vaihdella ja vertailla eri mallien välisiä tuotoksia. Kielimallit ovat tehokkaampia ja nopeampia, jos ne on koulutettu joltain tiettyä tehtävää tai käyttöä varten. Pelkästään englanninkieliseen tietoon keskittyvät kielimallit saattaisivat toimia yhtä hyvin tai jopa paremmin kuin GPT4, vaikka niiden koulutusparametrien määrä olisi vain pieni osa GPT4:stä. Lähes kaikki suositut kielimallit on koulutettu monelle eri kielelle, joten niiden nopeutta pystyisi vielä optimoimaan. Gemini oli alun perin suunniteltu opinäytetyön API-osuuksien toteuttamiseen, mutta Geminin API ei ole avoinna Suomessa, ainakaan vielä. Geminin API-käyttö olisi ollut ilmaista ja Geminin ”Ultra”-version kielimalli olisi testien perusteella ollut parempi kuin GPT4. Valitettavasti ”Ultra”-versio ei ole julkisesti avoinna vielä tätä opinäytetyötä kirjoittaessa.



Kielimallit ovat kehittyneet merkittävästi pelkästään vuoden sisällä, ja kielimal-  
lien välinen kilpailu tuntuu olevan huipussaan, mikä aiheuttaa nopeampaa ke-  
hitystä. On kiinnostavaa nähdä, kuinka kehittyneitä kielimallit ovat parin vuo-  
den päästä, vai alkaako kielimallit siinä vaiheessa jo muistuttamaan ensim-  
mäisiä AGI (Artificial General Intelligence) -malleja. Keskustelua on jo AGI-kä-  
sitteestä, ja onko esim. GPT4 jo aikainen esiversio AGI-malli.

## LÄHTEET

Acharya, A. 2023. An Introduction to Diffusion models for Machine Learning. Encord. WWW-dokumentti. Saatavissa: <https://encord.com/blog/diffusion-models/#:~:text=Diffusion%20models%20are%20a%20class%20of%20generative%20models%20that%20simulate,a%20sequence%20of%20invertible%20operations.> [viitattu 5.1.2024].

Announcing Grok. 2023. xAI. WWW-dokumentti. Saatavissa: <https://x.ai/> [viitattu 5.1.2024].

API Reference – OpenAI API. s.a. OpenAI. WWW-dokumentti. Saatavissa: <https://platform.openai.com/docs/api-reference> [viitattu 26.11.2023].

BloombergGPT: Putting Finance to Work using Large Language Models. s.a. Packtpub. WWW-dokumentti. Saatavissa: <https://www.packtpub.com/article/hub/bloomberggpt-putting-finance-to-work-using-large-language-models> [viitattu 25.11.2023].

Boesch, G. s.a. Autoencoder in Computer Vision – Complete 2024 Guide. Viso.ai. WWW-dokumentti. Saatavissa: <https://viso.ai/deep-learning/autoencoder/#:~:text=An%20autoencoder%20model%20consists%20of,from%20the%20latent%20space%20representation.> [viitattu 26.11.2023].

Elias, J. 2023. Google's newest A.I. model uses nearly five times more text data for training than its predecessor. CNBC. WWW-dokumentti. Päivitetty 17.3.2023. Saatavissa: <https://www.cnbc.com/2023/05/16/googles-palm-2-uses-nearly-five-times-more-text-data-than-predecessor.html> [viitattu 5.1.2024].

Fernando, J. 2022. What Are Autoregressive Models? Investopedia. s.a. WWW-dokumentti. Päivitetty 6.4.2022. Saatavissa: <https://www.investopedia.com/terms/a/autoregressive.asp> [viitattu 26.11.2023].

Hetler, A. 2023. ChatGPT. TechTarget. WWW-dokumentti. Päivitetty 12.2023. Saatavissa: <https://www.techtarget.com/whatis/definition/ChatGPT> [viitattu 5.1.2024].

Kapronczay, M. 2022. A Beginner's Guide to Language Models. Built In. WWW-dokumentti. Saatavissa: <https://builtin.com/data-science/beginners-guide-language-models> [viitattu 5.1.2024].

LLM Parameters. s.a. Deepchecks. WWW-dokumentti. Saatavissa: <https://deepchecks.com/glossary/llm-parameters/> [viitattu 5.1.2024].

Lutkevich, B. 2020. What is BERT (Language Model) and How Does It Work? TechTarget. s.a. WWW-dokumentti. Päivitetty 1.2020. Saatavissa: <https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model#:~:text=BERT%2C%20which%20stands%20for%20Bidirectional,calculated%20based%20upon%20their%20connection.> [viitattu 25.11.2023].

Merritt, R. 2022. What Is a Transformer Model? Nvidia. WWW-dokumentti. Saatavissa: <https://blogs.nvidia.com/blog/what-is-a-transformer-model/> [viitattu 5.1.2024].

Newton, W. 2023. What is BioGPT and what does it mean for healthcare? Clinical Trials Arena. WWW-dokumentti. Julkaistu 9.2.2023. Saatavissa: <https://www.clinicaltrialsarena.com/news/biogpt-healthcare/> [viitattu 25.11.2023].

Ozdemir, S. 2023. Quick Start Guide to Large Language Models: Strategies and Best Practices for Using ChatGPT and Other LLMs. Addison Wesley Professional. E-kirja. Saatavissa: [https://books.google.fi/books/about/Quick\\_Start\\_Guide\\_to\\_Large\\_Language\\_Mode.html?id=U6HWzwEACAAJ&redir\\_esc=y](https://books.google.fi/books/about/Quick_Start_Guide_to_Large_Language_Mode.html?id=U6HWzwEACAAJ&redir_esc=y) [viitattu 19.11.2023].

Pichai, S; Hassabis, D. 2023. Introducing Gemini: our largest and most capable AI model. Google. WWW-dokumentti. Saatavissa: <https://blog.google/technology/ai/google-gemini-ai/#capabilities> [viitattu 5.1.2024].

Raghuath, A. 2023. GPT-4 Parameters Explained. Hix.AI. WWW-dokumentti. Päivitetty 26.12.2023. Saatavissa: <https://hix.ai/hub/chatgpt/gpt-4-parameters> [viitattu 5.1.2024].

Ritika. 2023. PaLM Model: Breaking Ground with 540 Billion Parameters. Labellerr. s.a. WWW-dokumentti. Julkaistu 10.5.2023. Saatavissa: <https://www.labellerr.com/blog/pathways-language-model-palm-scaling-to-540-billion-parameters-for-breakthrough-performance/> [viitattu 25.11.2023].

Roy, S. 2023. Building a Domain-Specific Large Language Model. Analytics Steps. WWW-dokumentti. Päivitetty 18.9.2023. Saatavissa: <https://www.analyticssteps.com/blogs/building-domain-specific-large-language-model> [viitattu 26.11.2023].

Welcome to Gemini era. s.a. Google DeepMind. WWW-dokumentti. Saatavissa: <https://deepmind.google/technologies/gemini/#introduction> [viitattu 5.1.2024].

What is a transformer model? s.a. IBM. WWW-dokumentti. Saatavissa: <https://www.ibm.com/topics/transformer-model> [viitattu 5.1.2024].

## KUVALUETTELO

- Kuva 1. Kielimallien tulokset eri testeissä. Kiviranta, O. 7.1.2024
- Kuva 2. Just ask. Kiviranta, O. 25.11.2023.
- Kuva 3. Perus prompti ja rakenteen esimerkki. Kiviranta, O. 16.12.2023.
- Kuva 4. Käsintehty esimerkki. Kiviranta, O. 16.12.2023.
- Kuva 5. Funktio avainsanojen kanssa. Kiviranta, O. 16.12.2023.
- Kuva 6. ChatGPT:n luoma tarina. Kiviranta, O. 16.12.2023.
- Kuva 7. ChatGPT API kutsu Pythonilla. Kiviranta, O. 26.1.2024.
- Kuva 8. Kuvien luonnin API. Kiviranta, O. 10.2.2024.
- Kuva 9. Valmis kuva. Kiviranta, O. 10.2.2024.
- Kuva 10. NuGetForUnity asennus. Kiviranta, O. 21.1.2024
- Kuva 11. Python.NET asennus. Kiviranta, O. 21.1.2024
- Kuva 12. Python.NET käyttö. Kiviranta, O. 21.1.2024
- Kuva 13. Api Compability Level. Kiviranta, O. 27.1.2024
- Kuva 14. APIData-luokan JSON rakenne. Kiviranta, O. 27.1.2024
- Kuva 15. API:sta palautuvan datan luku. Kiviranta, O. 26.1.2024
- Kuva 16. JSON-tiedostojen luku C#:lla. Kiviranta, O. 7.1.2024
- Kuva 17. Tapahtumien haku JSON-tiedostosta. Kiviranta, O. 27.1.2024
- Kuva 18. Väliaikaisen tapahtumalistan käsittely. Kiviranta, O. 7.1.2024
- Kuva 19. Esimerkki tietovisapeli. Kiviranta, O. 7.1.2024
- Kuva 20. Kysymyksen ja vastauksien lista. Kiviranta, O. 7.1.2024
- Kuva 21. Muuttujia. Kiviranta, O. 7.1.2024
- Kuva 22. Tietovisapelin promptin selitys. Kiviranta, O. 7.1.2024
- Kuva 23. Tietovisapelin promptin esimerkki. Kiviranta, O. 7.1.2024
- Kuva 24. Promptin viimeinen osa. Kiviranta, O. 7.1.2024
- Kuva 25. Promptin testaus ChatGPT3.5 vaikeustasolla 0. Kiviranta, O. 10.1.2024

Kuva 26. Promptin testaus PaLM2 vaikeustasolla 0. Kiviranta, O. 10.1.2024

Kuva 27. Promptin testaus PaLM2 vaikeustasolla 10. Kiviranta, O. 10.1.2024

Kuva 28. Promptin testaus GPT3.5 vaikeustasolla 10. Kiviranta, O. 10.1.2024

Kuva 29. Kielimallien vertailu vaikeustasolla 0. Kiviranta, O. 10.1.2024

Kuva 30. Kielimallien vertailu vaikeustasolla 10. Kiviranta, O. 10.1.2024

Kuva 31. Promptin syöttäminen kielimallille pythonilla. Kiviranta, O. 27.1.2024

Kuva 32. Kielimallin datan käyttö reaaliajassa. Kiviranta, O. 27.1.2024