Korištenje OpenAI API-ja za kreiranje dijaloga unutar videoigre

Matijanić, David

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

David Matijanić

KORIŠTENJE OPENAI API-JA ZA KREIRANJE DIJALOGA UNUTAR VIDEOIGRE

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

David Matijanić

Matični broj: 49262

Studij: Informacijski i poslovni sustavi

KORIŠTENJE OPENAI API-JA ZA KREIRANJE DIJALOGA UNUTAR VIDEOIGRE

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Mladen Konecki

Varaždin, srpanj 2024.

David Matijanić

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj završni rad prolazi kroz izradu videoigre unutar *GameMaker Studio 2* koja se povezuje na i koristi OpenAI API kako bi likovi (eng. *NPC*) unutar videoigre generirali dijaloge sa igračem. Igrač može pisati pitanja, pitati za pomoć ili odabrati jednu od više ponuđenih opcija dijaloga za koja će se tada generirati prikladan odgovor koristeći OpenAI API, koji u obzir uzima *NPC*-a koji daje odgovor i cjelokupni kontekst igrice. Uz teoretski dio, rad će se detaljno fokusirati na praktičnu izradu takve videoigre, sa posebnim fokusom na korištenje OpenAI API-ja, što uključuje slanje zahtjeva, dobivanje i interpretiranje dobivenih odgovora te prikazivanje istih unutar videoigre.

Ključne riječi: videoigra, OpenAI, API, NPC, dijalog

Sadržaj

1. Uvod

Videoigre su jedan od najčešćih načina digitalne zabave. Razvojem tehnologije, popularnost videoigara, a i sama granica mogućeg u videoigrama raste te je tako danas nerijetko da videoigre „uranjaju“ igrača u vlastiti svijet, korištenjem realistične grafike, snažnog narativa i dinamične glazbe. Jedan od ključnih aspekata modernih videoigara su likovi (eng. *NPC*, dalje u tekstu *NPC*) s kojima igrači mogu komunicirati i imaju osjećaj da s njima imaju stvarne interakcije. Jedan od načina na koji *NPC*-ovi unutar videoigara to ostvaruju je korištenjem dijaloga. Jedni od najkorištenijih vrsta dijaloga su dijalozi s granama (gdje igrač ima više mogućih opcija), kontekstualni dijalozi (ovise o stanju videoigre u trenutku), no postoje još mnogo vrsta dijaloga, koji se mogu i kombinirati.

Umjetna inteligencija danas se najčešće veže uz pojam generativne umjetne inteligencije i jedna je od najbrže rastućih tehnologija danas. Omogućuje generiranje slika, videozapisa, ali i teksta temeljenog na određenim ulogama, kao što je popularna platforma *ChatGPT* te je moguće voditi dijalog kao sa stvarnom osobom, pišući stvarna pitanja ili ideje, bez ograničenja predefiniranih opcija i dobiti odgovore koji ovise o unesenom pitanju, prethodnom razgovoru ili općenito o kontekstu.

Upravo je kombinacija te dvije tehnologije tema ovog završnog rada, odnosno izrada videoigre koja koristi generativnu umjetnu inteligenciju za dinamičko generiranje dijaloga likova koji ovisi o stanju igrice, kontekstu te prethodnim pitanjima i odgovorima. Rad prolazi kroz spajanje na samo OpenAI aplikacijsko programsko sučelje (eng. *application program interface*, dalje u tekstu *API*) kroz programski kod za izradu videoigre te interpretiranje i prikazivanje dobivenog odgovora unutar iste, čiji će proces razvoja također biti opisan.

2. Metode i tehnike rada

Za izradu samog završnog rada, korišten je Microsoft Word. Korišten je FOI-jev predložak za pisanje završnog ili diplomskog rada. Referenciranje je napravljeno IEEE stilom.

Za izradu videoigre, kao praktičnog dijela završnog rada, korišten je alat *GameMaker Studio 2* i programski jezik GML, koji je sastavni dio tog alata. Većinu znanja o korištenju tog alata i izradi osnovnih mehanika igara, kao i nekih naprednijih, posjedujem od ranije, jer koristim *GameMaker Studio 2* i starija izdanja, kao što je *GameMaker Studio 1.4*, ili *Game Maker 8* već preko 12 godina. Za neke funkcionalnosti, kao što je kreiranje i slanje HTTP zahtjeva koristeći GML, koristio sam službenu dokumentaciju i različite online vodiče. Izrada osnovnih mehanika videoigre nije bio izazovan zadatak.

Znanje o *API*-jima i REST servisima stekao sam na fakultetu, a za korištenje samog OpenAI API-ja primarno sam proučavao dokumentaciju. Korištenje *API*-ja unutar *GameMaker Studio 2* bio je jedan od zahtjevnijih zadataka prilikom izrade praktičnog dijela ovog završnog rada, kao i upravljanje korištenjem *API*-ja u kontekstu same igrice (prikazivanje odgovora u *textbox*-ovima, slanje napisanog odgovora, postavljanje uloge za odgovore).

Mapa videoigre izrađena je koristeći ugrađeni *room editor* u *GameMaker Studio 2*, a raspored elemenata dizajniran je uz ideju videoigre na umu (dvorac kojeg brani vitez i mora se pogoditi lozinka za ulazak). *Sprite*-ovi i *tile*-ovi u igrici vlastiti su rad, napravljeni primarno koristeći ugrađen *sprite editor* u *GameMaker Studio 2*, no za neku grafiku korišten je alat *paint.NET*. Dio grafike, kao što je igrač, drveća i pojedini *tile*-ovi uzeti su iz starijih vlastitih videoigara.

3. Korištene tehnologije

Prethodno poglavlje samo površno opisuje korištene tehnologije, dok će u ovom poglavlju korištene tehnologije biti detaljnije opisane, uz poseban fokus na one elemente tih tehnologija koje su bitne za realizaciju same videoigre.

3.1. OpenAI API

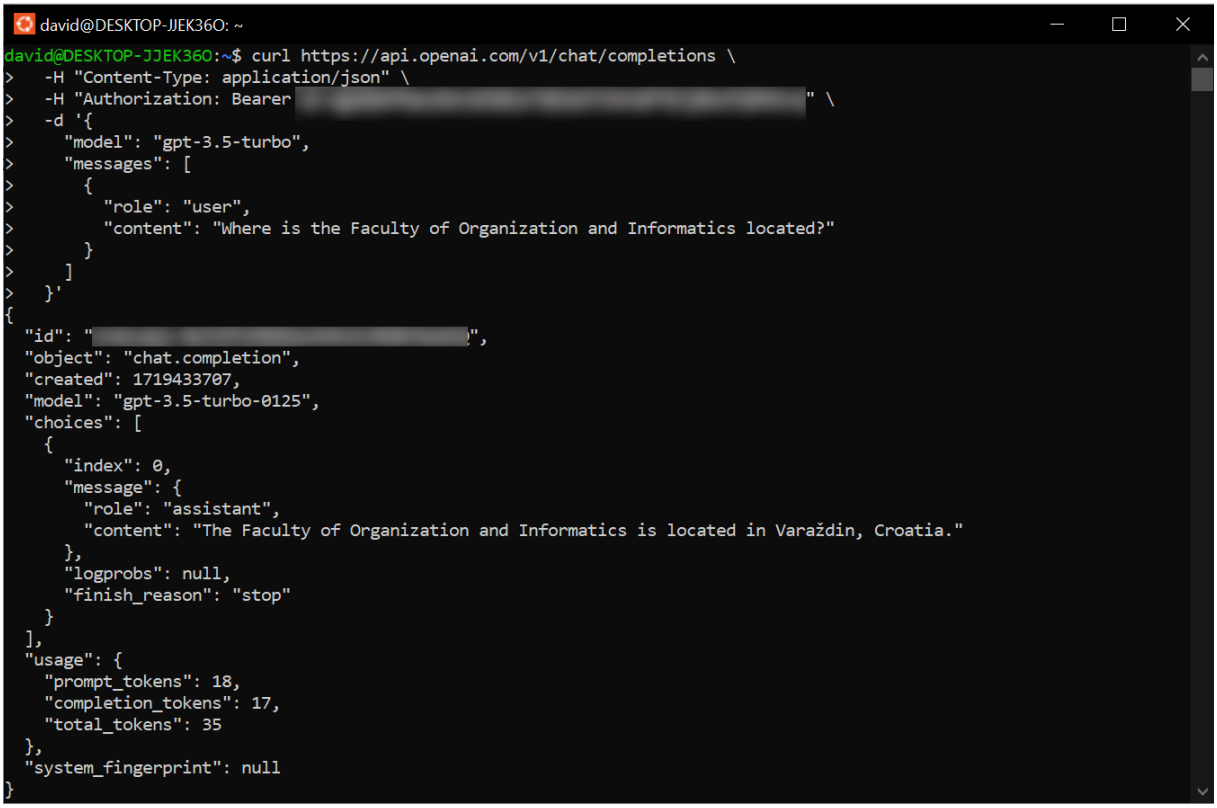
Aplikacijsko programsko sučelje je "mehanizam koji omogućuje da dvije komponente softvera međusobno komuniciraju pomoću skupa definicija i protokola.“ [1] Kompanija OpenAI, najpoznatija po njihovoj platformi *ChatGPT*, 11. lipnja 2020. godine najavila je vlastiti OpenAI API, koji služi za pristup njihovim *AI* modelima. [2] OpenAI API omogućuje da korisnici putem HTTP zahtjeva šalju upite, *prompt*-ove[1] na servis i dobiju *AI*-generiran odgovor, generiran od strane jednog od njihovih *Chat Completions API* modela. [3] Neki od modela koji su dostupni putem OpenAI API za generiranje teksta su [4]:

- *gpt-3.5-turbo*: „može razumjeti i generirati prirodni jezik ili programski kod i optimizirani su za korištenje sa *Chat Completions API*, ali rade dobro i za ne-razgovorne zadatke.“ [4] Ovaj će se model primarno koristiti pri izradi igrice radi jeftinije cijene po tokenu.
- *gpt-4o*: njihov najkompleksniji model, multimodalan (prima i slike uz tekst) i ima najbolju viziju i performanse za sve jezike (ne samo engleski jezik). [4] Ovaj će se model koristiti kod isprobavanja slanja slike.

Oba modela moguće je besplatno testirati na platformi *ChatGPT* na Internetu.

Na sljedećoj slici moguće je vidjeti korištenje OpenAI API-ja slanjem zahtjeva na *endpoint* „https://api.openai.com/v1/chat/completions“ koristeći alat *curl*:

[1] Prema [3], *prompt* je naziv za tekstualni ulaz u modele.



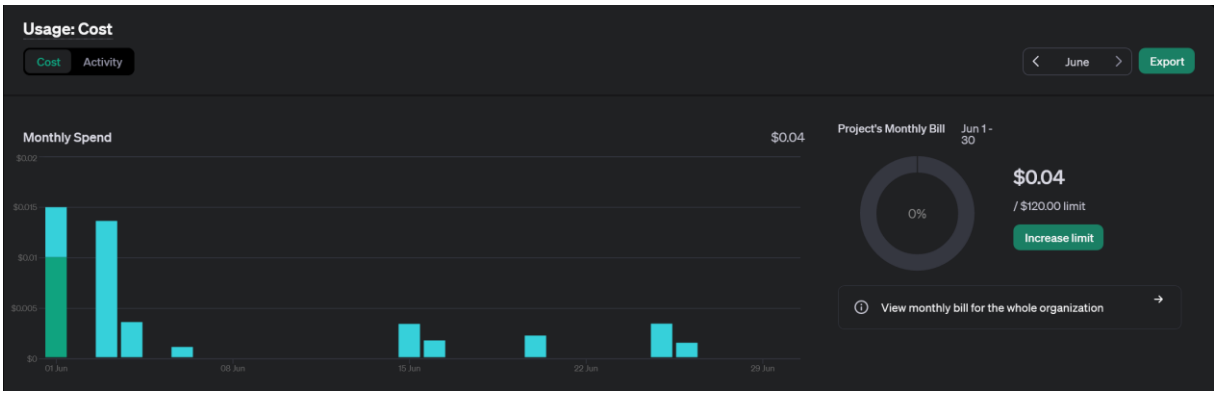
Slika 1. Slanje zahtjeva na OpenAI API koristeći *curl* [autorski rad]

3.1.1. API ključ

Kako bi se mogao koristiti *API*, potrebno je koristiti ispravan *API* ključ. Kod OpenAI API, *API* ključ služi za autentikaciju projekta te ga je potrebno priložiti pri slanju zahtjeva, kako bi se zahtjev mogao povezati uz projekt [5] te mjerila statistika korištenja i iskoristio određen broj tokena[2]. Kod svakog zahtjeva potrebno je priložiti *API* ključ u zaglavlje „Authorization“ s oznakom „Bearer“ ispred samog *API* ključa.

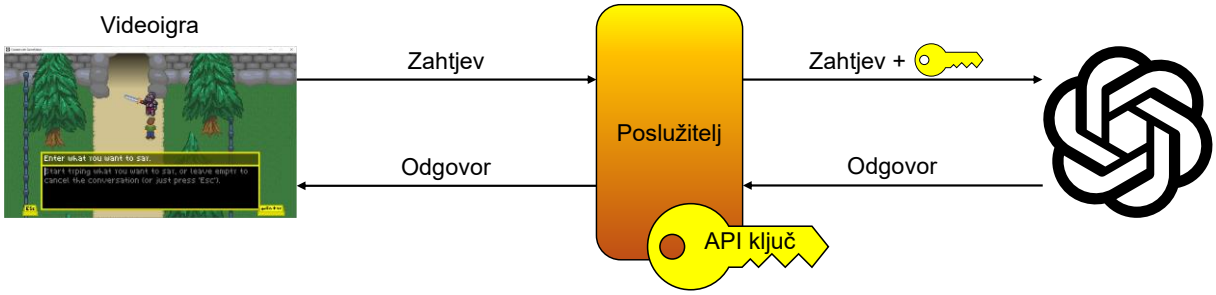
Statistiku korištenja ključa te potrošene tokene i cijenu po tokenu moguće je vidjeti na *Dashboard*-u projekta:

[2] Prema [8], tokeni su „dijelovi teksta“ u kojima jezični modeli čitaju i generiraju odgovore. Tokeni su detaljnije opisani u sljedećem poglavlju.



Slika 2. *Dashboard* za korištenje *API* ključa (slika zaslona, [6])

Problem koji se javlja je „Gdje smjestiti *API* ključ?“ *API* ključ treba biti privatan, no ne može se koristiti unutar koda videoigre, jer je onda javno dostupan svima koji imaju videoigru i može se zloupotrijebiti. Prema točki 2 na [7], *API* ključ bi se trebao sigurno smjestiti na *backend* poslužitelju te tako nije dostupan svima na klijentskoj strani (videoigra), nego je pristup *API*-ju omogućen samo autoriziranim korisnicima, a o autentikaciji i autorizaciji brine se poslužitelj. Tada bi pristupanje *API*-ju izgledalo ovako:



Slika 3. Pohranjivanje *API* ključa na poslužitelju [autorski rad, OpenAI logo sa https://openai.com/brand/]

Kod izrade videoigre neće biti izrađen poslužitelj, nego će za potrebu ne otkrivanja privatnog *API* ključa biti kreirana i web inačica videoigre, gdje se može unijeti vlastiti *API* ključ.

3.1.2. Tokeni

Kod korištenja OpenAI API-ja, zahtjevi i odgovori računaju se u **tokenima**. „Jezični modeli čitaju i pišu tekst u dijelovima koji se nazivaju tokeni.“ [8] Prema [8], token može biti duljine sve od jednog znaka, do cijele riječi, no u nekim slučajevima može biti i izvan te granice. Cjelokupan broj tokena iskorišten u jednom *API* pozivu ovisi o [8]:

- Koliko *API* poziv košta, plaća se po tokenu.
- Koliko tokena se iskoristi za odgovor, generiranje dužih odgovora sa više tokena traje duže.

Različiti modeli za generiranje teksta imaju različitu cijenu po modelu [9]:

- *gpt-3.5-turbo* na milijun ulaznih tokena košta 0.50$, a na milijun izlaznih tokena 1.50$.
- *gpt-4o* na milijun ulaznih tokena košta 5.00$, a na milijun izlaznih tokena 15.00$, odnosno, deset puta je skuplji nego *gpt-3.5-turbo*. Trebat će obratiti posebnu pozornost kod korištenja ovog modela, pogotovo jer on omogućuje slanje slika na *API*, koje same po sebi koriste puno tokena, puno više nego tekst.

Da bi se pratilo korištenje tokena, može se koristiti *dashboard* dostupan na „https://platform.openai.com/usage“. Također, odgovor na zahtjev sadrži polje „*total_tokens*“ (vidljivo na slici 1) koje govori nešto više o broju iskorištenih tokena za pojedini zahtjev. Drugi način za ograničavanje korištenja tokena je postavljanje atributa „*max_tokens*“ u zapisu [5] i ograničiti veličinu generiranog odgovora u pogledu tokena.

3.1.3. Slanje poruka

Na *chat completions API* zahtjev mogu se slati različite vrste poruka. Prema [5], neke od uloga pod kojima se poruke mogu slati su:

- *system* poruke: ove poruke interpretiraju se kao od uloge „system“. Koristi se kada se *API*-ju želi poslati nešto u kontekstu sistema, kao što je uputa kako da se ponaša, koju ulogu da glumi…
- *user* poruke: ove poruke interpretiraju se kao da ih je napisao sam korisnik. Koriste se kada korisnik šalje određen upit na *API* (pitanje, rečenica…).
- *assistant* poruke: ove poruke generirane su od strane *AI*-ja, odnosno *chat completions API*-ja. To su vraćene poruke, no ako se na *API* želi poslati povijest razgovora, potrebno je označiti da su poruke prethodno generirane i stavlja se ova uloga.

Problem sa *API*-jem je što se razgovori ne pamte te ukoliko je potrebno znati prethodni kontekst razgovora, potrebno je slati sve prethodne poruke, što znači i veći broj tokena te veća cijena. Slanje više prethodnih poruka može se obaviti na sljedeći način:

```
"messages": [
    {
        "role": "system",
        "content": "Ti si pomoćni asistent."
    },
    {
```

```
            "role": "user",
            "content": "Kako se zoveš?"
        },
        {
            "role": "assistant",
            "content": "Ja sam pomoćni asistent, trenutno nemam lično ime.
Kako mogu pomoći?"
        },
        {
            "role": "user",
            "content": "Kako napisati dobro motivacijsko pismo?"
        },
    ]
```

Iz primjera je vidljivo da se kao sadržaj u „*content*“ može poslati tekst. Ukoliko se koristi model *gpt-4o*, moguće je poslati i sliku. Slika se može priložiti kao URL, ili u base64 formatu. Slanje slike izvršava se na sljedeći način [10]:

```
    "content": [
        {
            "type": "text",
            "text": "Što je na ovoj slici?"
        },
        {
            "type": "image_url",
            "image_url": {
            "url": "https://www.foi.unizg.hr/sites/default/files/foi.jpg"
            }
        }
    ]
```

Slike se mogu slati samo kroz ulogu „*user*“.

U praktičnom primjeru slika će se slati u base64 formatu na sljedeći način [10]:

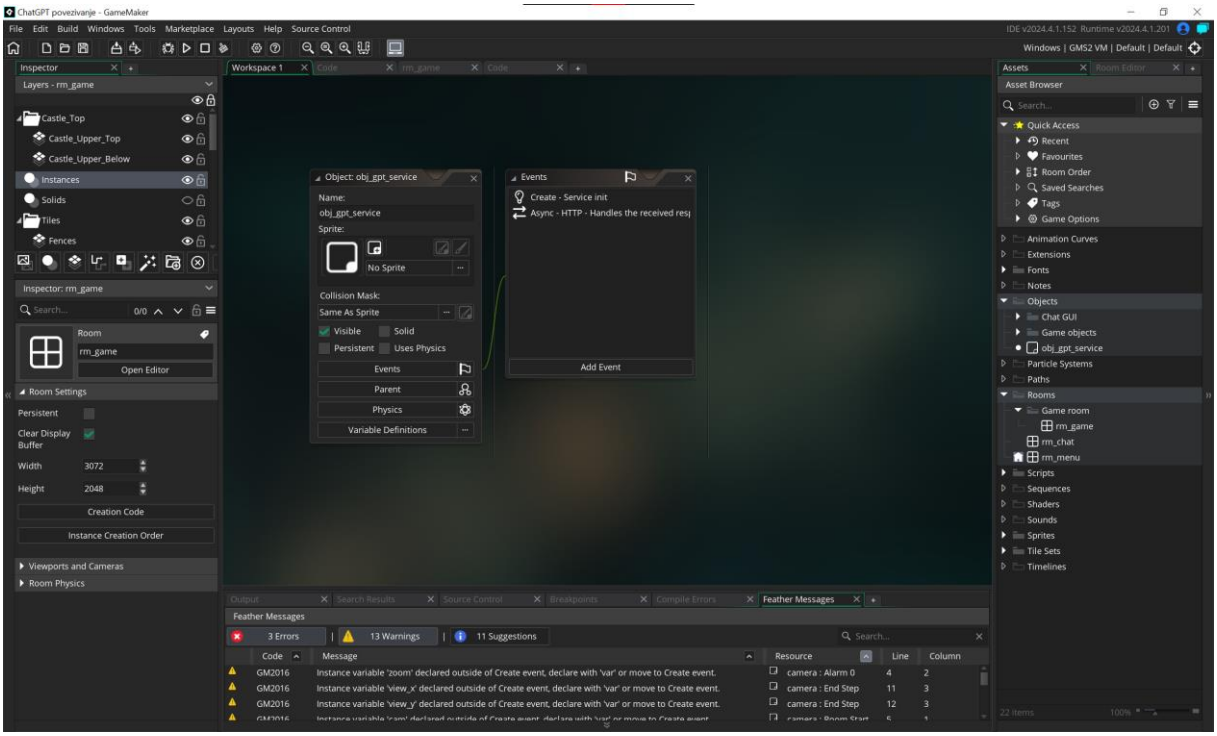
```
"url": f"{base64_image}"
```

3.2. GameMaker Studio 2

GameMaker Studio 2 je alat za razvoj primarno 2D videoigara za više platformi. *GameMaker Studio 2* je ujedno i integrirano razvojno okruženje sa mnoštvom alata za dizajn *sprite*-ova, soba (eng. *rooms*), objekata i ostalih elemenata. [10] *GameMaker Studio 2* ima dva načina za kreiranje videoigara [11]:

- *GML Visual*: „vizualna skriptna metoda za programiranje sa *GameMaker Language* (GML). Sastoji se od akcijskih blokova za konstruiranje logike igrice na vizualan i intuitivan način, i idealan je alat za ljude koji se uče programirati, ili za ljude koji imaju sklonost vizualnom.“ [12]
- *GameMaker Language*: skriptni jezik za *GameMaker*. „Ovaj je jezik struktuiran tako da dopusti korisnicima da kreiraju svoju igricu na intuitivan i fleksibilan način dok omogućuje snagu bilo kojeg drugog glavnog programskog jezika.“ [13] GML će se koristiti pri izradi praktičnog dijela rada.

Okruženje pokrenutog alata *GameMaker Studio 2* je sljedeće:



Slika 4. *GameMaker Studio 2* [autorski rad, slika zaslona]

3.2.1. Resursi unutar GameMaker Studio 2

Za izradu videoigre, unutar *GameMaker Studio 2*, koriste se različiti resursi (eng. *assets*). Neki od bitnijih resursa za izradu videoigre su sljedeći [14]:

- *Sprite*: „vizualne reprezentacije objekata u igrici“, slika ili više slika (animacija).
- *Script*: „kontejner za jednu ili više napisanih funkcija[3]“.

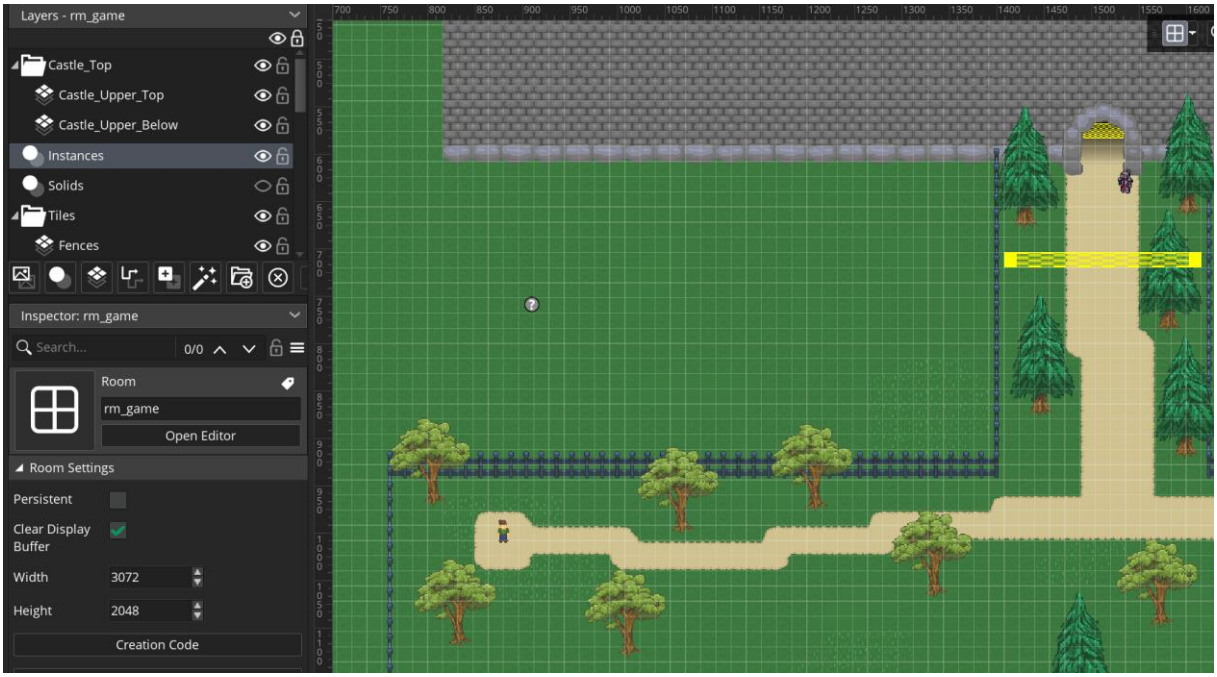
[3] „funkcija“ se odnosi na funkciju u programskom kodu

- *Object*: najvažniji resurs, kontrolira aspekte videoigre, „gradivni blokovi“ videoigre.
- *Room*: „prostor u koji se postavljaju instance objekata od kojih se sastoji igrica i gdje se odvija radnja igrice“. Može se razmišljati o sobama kao o „levelima“ videoigre, ili prostorima za različite dijelove (glavni izbornik, tutorijal, postavke…)

Postoji još vrsta resursa, kao što su zvukovi, sekvence, *tileset*-ovi, fontovi ili *shader*-i, no ovi su najvažniji za izgradnju osnove bilo koje videoigre.

3.2.1.1. Objekti

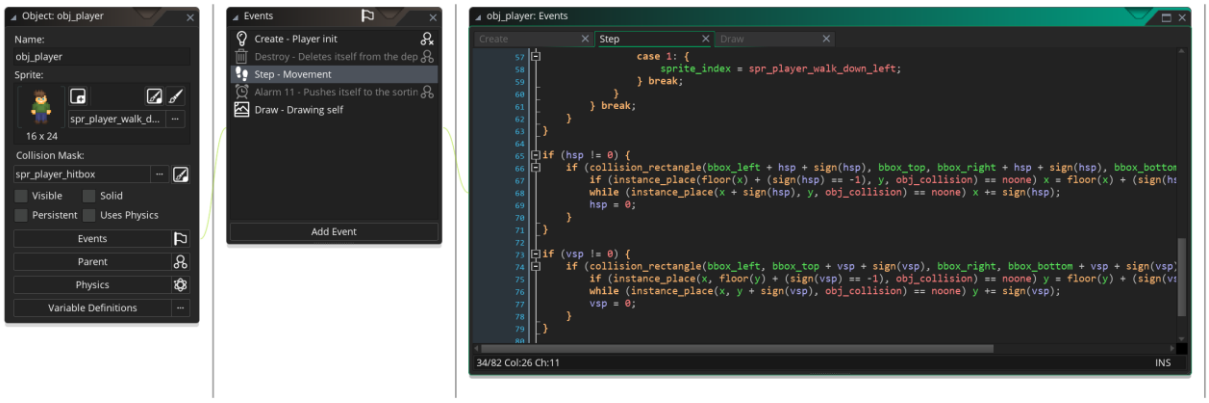
Jedan od najvažnijih resursa unutar *GameMaker Studio 2* je objekt. Mogu biti „vidljivi“ objekti igre sa pridruženom slikom (*sprite*-om) ili mogu biti „nevidljivi“ kontroleri koji upravljaju pozadinskim aspektima videoigre. [15] Instance objekata stavljaju se u sobe (*room*), a instance su „kopije“ resursa objekta, dok sam objekt definira ono što sve njegove instance nasljeđuju i što im je zajedničko.



Slika 5. Objekti u sobi [autorski rad, slika zaslona]

Objekti uz sebe imaju vezana određena svojstva, kao što su naziv, dodijeljeni *sprite*, vidljivost ili perzistentnost. [15] Sve unutar *GameMaker Studio 2* bazirano je na događajima (eng. *Events*). Događaji definiraju logiku igre te se svakom objektu može dodijeliti radnja koja se obavlja na određeni događaj. [15] Neki događaji imaju i „poddogađaje“. Određeni bitniji događaji su sljedeći [15]:

- *Create Event*: svaka instanca objekta poziva ovaj događaj koji joj je definiran putem objekta prije svih ostalih događaja jednom. Najčešće se koristi za inicijalizaciju varijabli.
- *Step Event*: poziva se na svaki korak (eng. *tick*) videoigre dok je instanca aktivna u sobi. Koristi se najčešće za bitnu logiku, poput provjera kolizija, dohvata korisničkih događaja ili umjetne inteligencije za *NPC*-ove unutar videoigre.
- *Draw Event*: također se poziva na svaki korak videoigre dok je instanca aktivna, no koristi se za radnje vezane uz crtanje na zaslon. Koristi se ukoliko objekt treba crtati nešto drugo ili nešto umjesto dodijeljenog *sprite*-a automatski.
- *Keyboard And Mouse*: događaji koji služe za dohvat ulaza tipkovnice ili miša.
- *Asynchronous*: događaji koji se pozivaju na uzvraćeni poziv određenog vanjskog izvora, što može biti web poziv, ili od uređaja koji pokreće videoigru. [16] Poddogađaj ovog događaja, *Async – HTTP* poziva se na odgovor HTTP zahtjeva te će se u praktičnom dijelu rada koristiti za dohvat i obradu odgovora dobivenog od OpenAI API.



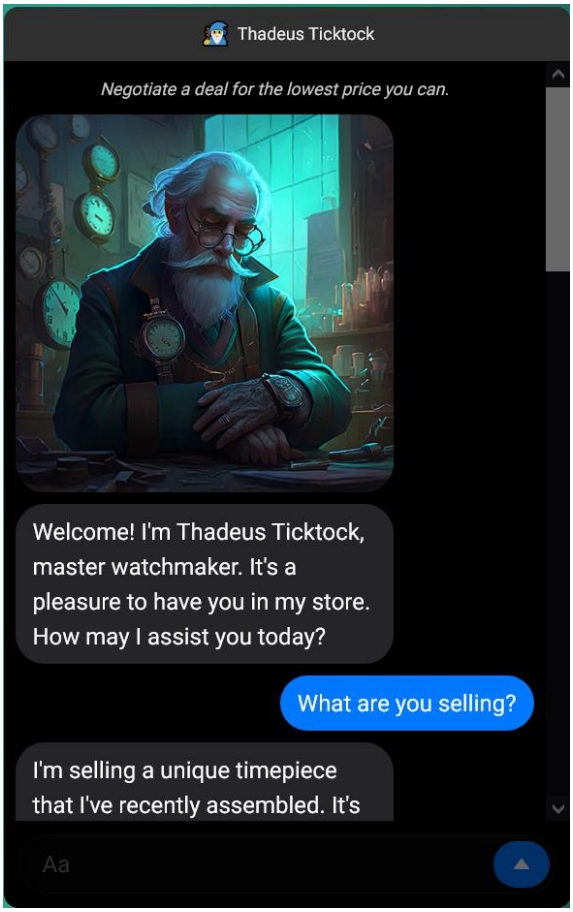
Slika 6. *Object Editor* sa otvorenim prozorom za događaje (*Events*) te otvorenim prozorom sa programskim kodom za događaj *Step*, napisanim u GML [autorski rad, slika zaslona]

4. Primjeri videoigara koji koriste AI za interakcije

Dok se pojam „AI“ koristi u videoigrama već duže vrijeme [17], u ovom poglavlju, pojam „AI“ odnosi se na generativnu umjetnu inteligenciju, poput OpenAI API-ja ili *ChatGPT*. [18]. Poglavlje prikazuje nekoliko primjera videoigara koji koriste *AI* za generiranje dijaloga ili ostalih elemenata koji nadopunjuju samu igru.

4.1. Bargainer.ai

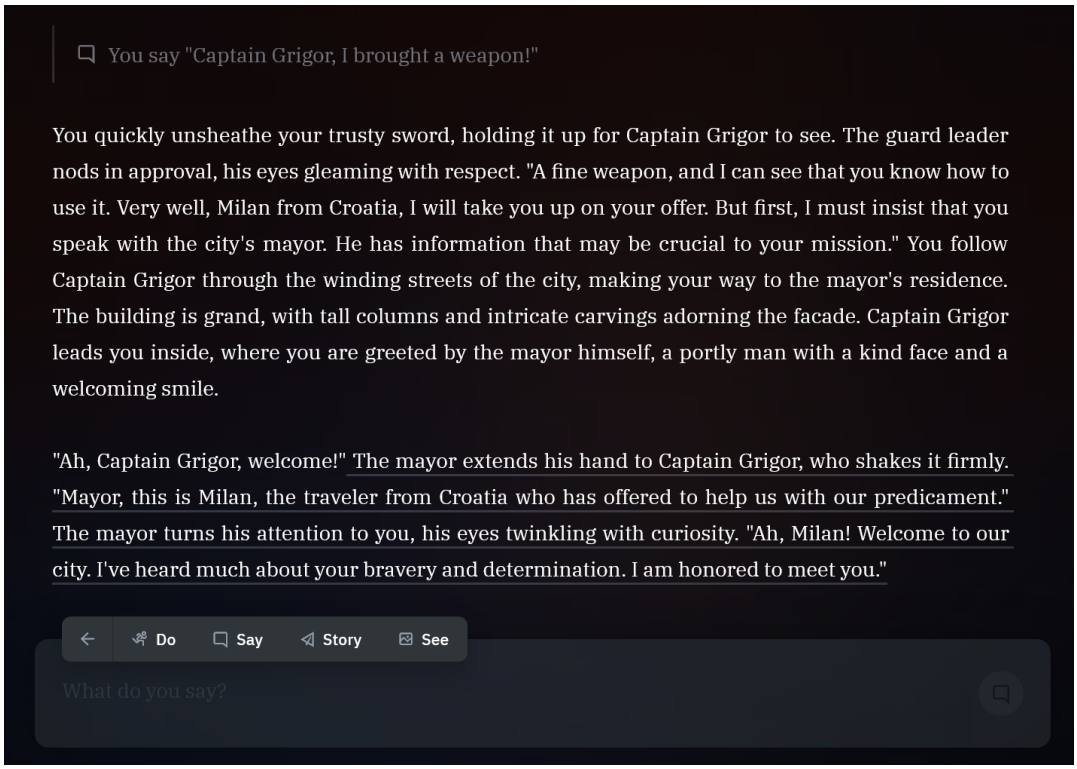
Bargainer.ai je online igra koja koristi OpenAI API [19] i cilj igre je cjenkati se sa *AI chatbot*-om te kupiti ručni sat za što nižu cijenu. [20] Igra je besplatna i dostupna je na webu na sljedećoj poveznici: „https://www.bargainer.ai“



Slika 7. Razgovor sa proizvođačem satova u *online* igri Bargainer.ai (slika zaslona [20])

4.2. AI Dungeon

AI Dungeon je „tekstualni avanturistički RPG s temom fantazije“. [21] *AI Dungeon* koristi *AI* za generiranje svjetova, likova i scenarija s kojima igrač može imati interakciju te igrač također diktira u kojem smjeru će igra ići. [22] Na sljedećoj slici vidljiv je dio igre te su vidljive radnje koje igrač može učiniti:



Slika 8. Videoigra *AI Dungeon* s vidljivim opcijama (slika zaslona [22])

4.3. Inworld mod za Skyrim

Inworld mod u videoigru Skyrim iz 2011. godine, dodaje mogućnost igračima da razgovaraju s *NPC*-ovima koristeći vlastiti glas ili tekst. [23] Inworld je generativna AI platforma koja se fokusira na kreiranje „živih“ *NPC* likova za videoigre. [24] Koriste nekoliko modela strojnog učenja da simuliraju ljudsku komunikaciju, omogućujući da *NPC*-ovi mogu voditi dijaloge koji ovise o radnjama i memoriji prethodnih događaja sa igračem.

Slika 9. *NPC* unutar Skyrim razgovara sa igračem [25]

Slika je preuzeta iz videozapisa koji prikazuje kako mod funkcionira, a dostupan je na sljedećoj poveznici: „https://youtu.be/d6sVWEu9HWU“

Sam mod je za preuzimanje dostupan na sljedećoj poveznici: „https://www.nexusmods.com/skyrimspecialedition/mods/98481“

5. Izrada videoigre

Ovo poglavlje opisuje stvaranje videoigre koja koristi OpenAI API kako bi dinamički generirala dijaloge koji ovise o radnji ili kontekstu igre. Poglavlje prolazi kroz najvažnije dijelove izrade sa velikim fokusom na integraciju samog *API*-ja u igru te korištenje istog. Prikazani su najvažniji dijelovi programskog koda, a radnje poput dizajniranja mape, pisanje „*glue*“ koda za povezivanje određenih mehanika, koda vezanog uz animacije ili crtanje[4], ispis teksta slovo-po-slovo, koda za grafičko sučelje, testnog projekta za testiranje *API*-ja ili koda čije se slične inačice pojavljuju više puta u različitim objektima neće biti prikazano.

5.1. Ideja videoigre

Ideja videoigre proizašla je iz proučavanja ostalih primjera videoigara koji koriste sličnu tehnologiju, kao što je [23] te razmišljanju o najboljem načinu prikazivanja korištenja *API*-ja i njegovih mogućnosti. Sama ideja o stvaranju videoigre došla je gledajući videozapis dostupan na sljedećoj poveznici: „https://www.youtube.com/watch?v=cUHMFEeWYbg“. U tom videozapisu prikazano je povezivanje na *API* kroz igricu unutar *GameMaker Studio 2*. Videozapis je također služio kao osnova za shvaćanje povezivanja te kako poslati HTTP zahtjev i dohvatiti odgovor unutar *GameMaker Studio 2*.

Priča videoigre je ta da se igrač nalazi pred dvorcem kojeg štiti vitez te igrač mora ući u dvorac. Igrač može pričati s vitezom i mora pogoditi lozinku te ako uspješno pogodi lozinku, dopušteno mu je ući u dvorac. Vitez također može pružiti sitnu pomoć oko lozinke, ili općenite savjete ako je lozinka pogođena.

[4] Odnosi se na animacije i crtanje *sprite*-ova

Slika 10. Videoigra sa vidljivim igračem, vitezom te prikazivanjem generiranog teksta [autorski rad, slika zaslona]

Kako bi bolje bile prikazane mogućnosti, na mapi se također nalazi trgovac, s kojim igrač može voditi dijalog o raznim temama vezanim uz svijet videoigre, kao što je dvorac, sam vitez, pomoć oko lozinke ili bilo koja druga tema, o kojoj ga igrač može pitati. Na mapi se nalaze kipovi kraljeva gdje svaki kip ima generiran dulji tekst o tom kralju. Ti kipovi su primarno postavljeni u igru kako pogađanje lozinke ne bi ovisilo o znanju izvan konteksta videoigre, a također prikazuju korištenje *API*-ja za interakciju koja nije dijalog.

5.2. Povezivanje na *API*

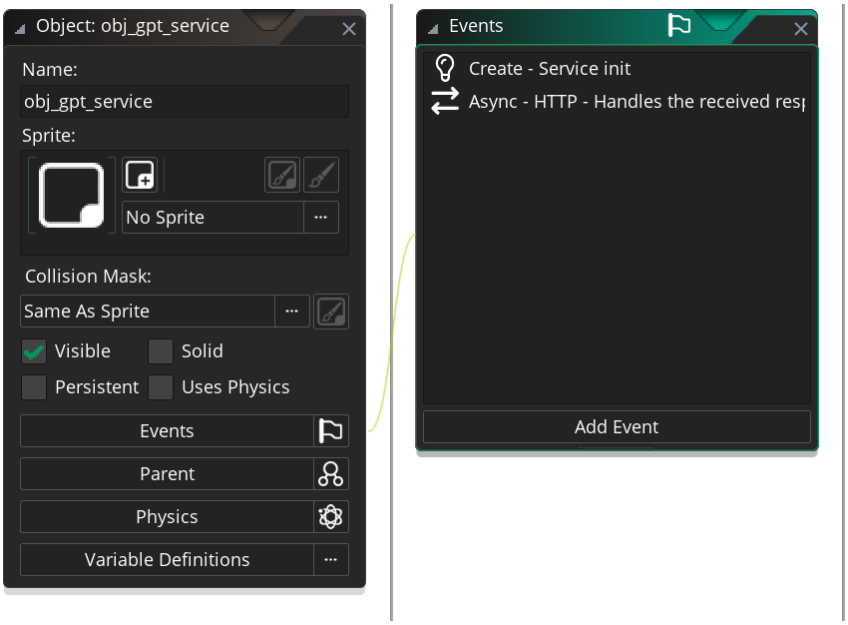
Ovo poglavlje opisuje povezivanje na OpenAI API te označava početak izrade same videoigre.

5.2.1. Dobivanje *API* ključa

Prije početka korištenja *API*-ja, potrebno je dobiti *API* ključ na poveznici „https://platform.openai.com/api-keys“. Poglavlje 3.1.1. detaljnije opisuje što je *API* ključ te kako se koristi.

5.2.2. Objekt za korištenje *API*-ja

Za korištenje *API*-ja unutar videoigre, bit će kreiran objekt koji će služiti kao posrednik između ostalih objekata u igri i *API*-ja, kako se funkcije za poziv ne bi morale zvati u svakom objektu koji ga koristi. Kreiran je novi objekt zvan „obj_gpt_service“ koji nema dodijeljen *sprite* te su mu postavljena dva događaja, to su *Create* i *Async – HTTP*:



Slika 11. Objekt „obj_gpt_service“ sa dva događaja [autorski rad, slika zaslona]

U *Create* događaju bit će definirane sve funkcije za korištenje *API*-ja putem tog objekta, kao i funkcija za slanje HTTP zahtjeva, a u *Async – HTTP* događaju, bit će obrađivan i interpretiran dobiven HTTP odgovor.

5.2.2.1. Slanje zahtjeva

Prije samog programskog koda za slanje zahtjeva, kreirana je prazna skripta „gpt_service_macros“ koja sadrži dvije makro vrijednosti, odnosno *API* ključ i koji *API* model se koristi. Makro vrijednosti su „imenovane vrijednosti koje se mogu koristiti u kodu da zamijene *hardkodirane* vrijednosti“, odnosno, to su konstantne vrijednosti koje se deklariraju jednom, na početku same igre. [26]. Dvije definirane makro vrijednosti su sljedeće:

```
#macro OPENAI_API_KEY "API_KLJUČ_OVDJE"
#macro OPENAI_MODEL "gpt-3.5-turbo"
```

Poglavlje 3.1.1. opisuje kako bi se *API* ključ trebao pravilno upotrebljavati.

Unutar *Create* događaja objekta „obj_gpt_service“ definirane su potrebne varijable te funkcija `send_message` koja služi za slanje zahtjeva:

```
authorization = $"Bearer {OPENAI_API_KEY}";
content_type = "application/json";
url = "https://api.openai.com/v1/chat/completions";
function send_message(_message, _temperature, _role = "user") {
    var _data_map = ds_map_create();
    ds_map_add(_data_map, "Authorization", authorization);
    ds_map_add(_data_map, "Content-Type", content_type);

    var _new_message = {
        "role": _role,
        "content": _message
    };

    var _data = {
        "model": OPENAI_MODEL,
        "messages": [
            _new_message
        ],
        "temperature": _temperature
    }
    show_debug_message("------ MESSAGES ------");
    show_debug_message(_data.messages);

    http_request(url, "POST", _data_map, json_stringify(_data));

    ds_map_destroy(_data_map);
    show_debug_message("Request sent!");
}
```

Podebljano su označene prethodno definirane makro vrijednosti.

Funkcija `send_message` traži tri argumenta:

- *_message*: tekstualna poruka (string)
- *_temperature*: vrijednost od 0 do 2, gdje veće vrijednosti čine izlaz više nasumičnim, a manje vrijednosti vraćaju više deterministične odgovore [5]
- *_role*: uloga [5] koja je inicijano „user“ no može se koristiti i druga uloga

Funkcija kreira *DS map* strukturu zvanu „data_map“, što je struktura koja omogućuje pohranjivanja vrijednosti ključ-vrijednost [27]. U tu mapu spremaju se zaglavlja za HTTP

zahtjev, konkretnije zaglavlje *Authorization* sa *API* ključem te zaglavlje *Content-Type*. Metoda *http_request* kreira HTTP zahtjev i šalje ga [28]. U ovom slučaju, šalje se POST zahtjev na endpoint „https://api.openai.com/v1/chat/completions“ sa postavljenim zaglavljima i prenesenom porukom.

5.2.2.2. Dohvat i interpretacija odgovora

Nakon što je zahtjev poslan, treba dohvatiti odgovor. To će biti odrađeno u *Async – HTTP* događaju objekta „obj_gpt_service“:

```
var _size = ds_map_size(async_load);
var _key = ds_map_find_first(async_load);

for (var i = 0; i < _size; i ++) {
    show_debug_message(async_load[? _key]);
    if (_key == "result") {
        try {
            var _loaded_result = json_parse(async_load[? _key]);
            var _response = _loaded_result.choices[0].message.content;
            handle_response(_response);
        } catch(_err) {
            show_debug_message(_err);
            handle_error();
        }

    }
    _key = ds_map_find_next(async_load, _key);
}
```

Programski isječak prvo iterira kroz varijablu *async_load*, što je *DS map* i sadrži informacije o dobivenom odgovoru. [29] Iterira se dok se ne pronađe vrijednost s ključem `result`, a vrijednost su dohvaćeni podaci s *API*-ja kao string [29], stoga je potrebno pretvoriti podatke u JSON format radi lakšeg čitanja. Čitanje te vrijednost odvija se u try-catch bloku kako bi se spriječilo rušenje programa ukoliko ta vrijednost ne postoji (u slučaju neispravnog *API* zahtjeva). Iz parsiranog odgovora čita se sam odgovor i sprema u lokalnu varijablu `_response` te prosljeđuje funkciji `handle_response`. Funkcije `handle_response` i `handle_error` potrebno je definirati u *Create* događaju objekta „obj_gpt_service“ te će svaki drugi objekt koji koristi „obj_gpt_service“ kao posrednik za *API* sam definirati kako će te funkcije izgledati. Isječak programskog koda za te dvije funkcije te funkcije za postavljanje njih su sljedeće:

```
handle_response = function(_response) {};
```

```
handle_error = function() {};
function set_response_handling(_func) {
    handle_response = _func;
}
function set_error_handling(_func) {
    handle_error = _func;
}
```

5.2.2.3. Prethodne poruke

Kako *API* ne pamti cijeli razgovor na njihovoj strani, što je ujedno i karakteristika REST *API*-ja [1], potrebno je kod slanja zahtjeva poslati sve prethodne poruke. Primjer je prikazan u poglavlju 3.1.3. Kako bi se slale i prethodne poruke, potrebno ih je pamtiti unutar samog objekta „obj_gpt_service“. U tu svrhu kreirane su varijable `remembers_conversation`, `max_last_message_number` i `last_messages` te funkcija `set_conversation_remember`:

```
remembers_conversation = false;
max_last_message_number = 4;
last_messages = [];
function set_conversation_remember(_remembers = true,
_max_last_msg_number = 4) {
    remembers_conversation = _remembers;
    max_last_message_number = _max_last_msg_number;
    if (!_remembers) {
        last_messages = [];
    }
}
```

Svrhe varijabli su sljedeće:

- *remembers_conversation*: određuje pamte li se prethodne poruke te se koristi za određivanje hoće li se na svaki zahtjev dodati prethodne poruke, ili neće.
- *max_last_message_number*: kako bi se ograničilo korištenje sve većeg broja tokena sa sve dužim razgovorima radi slanja svih prethodnih poruka, ova varijabla služi za ograničavanje broja prethodnih poruka slanih.
- *last_messages*: polje koje sadrži prethodne poruke.

Funkcija `set_conversation_remember` „uključuje i isključuje“ hoće li se slati prethodne poruke, ili neće.

Da bi se prethodne poruke uspješno dodavale pri slanju zahtjeva, modificirana je metoda `send_message` koristeći petlju koja dodaje onaj broj prethodnih poruka definiran varijablom `max_last_message_number` na poruke za slanje. Također se poslana poruka

dodaje u polje prethodnih poruka, `last_messages` te se veličina polja ograničava na vrijednost definiranu varijablom `max_last_message_number`:

```
function send_message(_message, _temperature, _role = "user") {
    var _data_map = ds_map_create();
    ds_map_add(_data_map, "Authorization", authorization);
    ds_map_add(_data_map, "Content-Type", content_type);

    var _new_message = {
        "role": _role,
        "content": _message
    };

    var _data = {
        "model": OPENAI_MODEL,
        "messages": [
            _new_message
        ],
        "temperature": _temperature
    }

    if (remembers_conversation) {
        for (var i = 0; i < array_length(last_messages); i++) {
            array_insert(_data.messages, i, last_messages[i]);
        }
    }

    show_debug_message("------ MESSAGES ------");
    show_debug_message(_data.messages);

    http_request(url, "POST", _data_map, json_stringify(_data));

    ds_map_destroy(_data_map);
    show_debug_message("Request sent!");

    if (remembers_conversation) {
        array_push(last_messages, _new_message);
        while (array_length(last_messages) > max_last_message_number) {
            array_shift(last_messages);
        }
    }
```

```
}
```

Kod dobivanja i obrađivanja zahtjeva u *Async – HTTP* događaju također je potrebno pohraniti dobivenu poruku u polje `last_messages` kako bi se i ona mogla priložiti kao prethodni razgovor. Te poruke imaju ulogu „assistant“ [5]:

```
try {
    var _loaded_result = json_parse(async_load[? _key]);
    var _response = _loaded_result.choices[0].message.content;
    handle_response(_response);
    last_response = _response;

    if (remembers_conversation) {
        array_push(last_messages, _loaded_result.choices[0].message);
        while (array_length(last_messages) > max_last_message_number) {
            array_shift(last_messages);
        }
    }
} catch(_err) {
    show_debug_message(_err);
    handle_error();
}
```

5.2.2.4. Postavljanje opisa

Veliku ulogu kod korištenja AI *API*-ja u videoigrama igra mogućnost da nije generični „AI pomoćnik“, poput *ChatGPT*, nego da može „glumiti“ ulogu, kao što bi to bio određen *NPC* u videoigri. Praksa dizajniranja ulaza za *AI* kako bi on generirao optimalne (željene) izlaze naziva se *prompt engineering*. [30]. Ovo poglavlje ne fokusira se na pisanje *prompt*-ova ili prakse pisanja *prompt*-ova, nego na implementiranje mehanizma da se objektu „obj_gpt_service“ može postaviti početni *prompt*, koji je ovdje nazvan ***objašnjenje*** (objašnjenje *AI*-ju na koji način da odgovara).

Za to su kreirane dvije varijable, `includes_explaination` i `explaination`, gdje prva označava šalje li se poruka s objašnjenjem pri slanju zahtjeva, a druga je konkretno objašnjenje (*prompt*). Kreirana je i funkcija `set_explaination` koja postavlja njoj preneseno objašnjenje:

```
includes_explaination = false;
explaination = "";
function set_explaination(_explaination = "") {
    if (string_length(_explaination) > 0) {
        includes_explaination = true;
```

```
            explaination = _explaination;
        } else {
            includes_explaination = false;
            explaination = "";
        }
    }
```

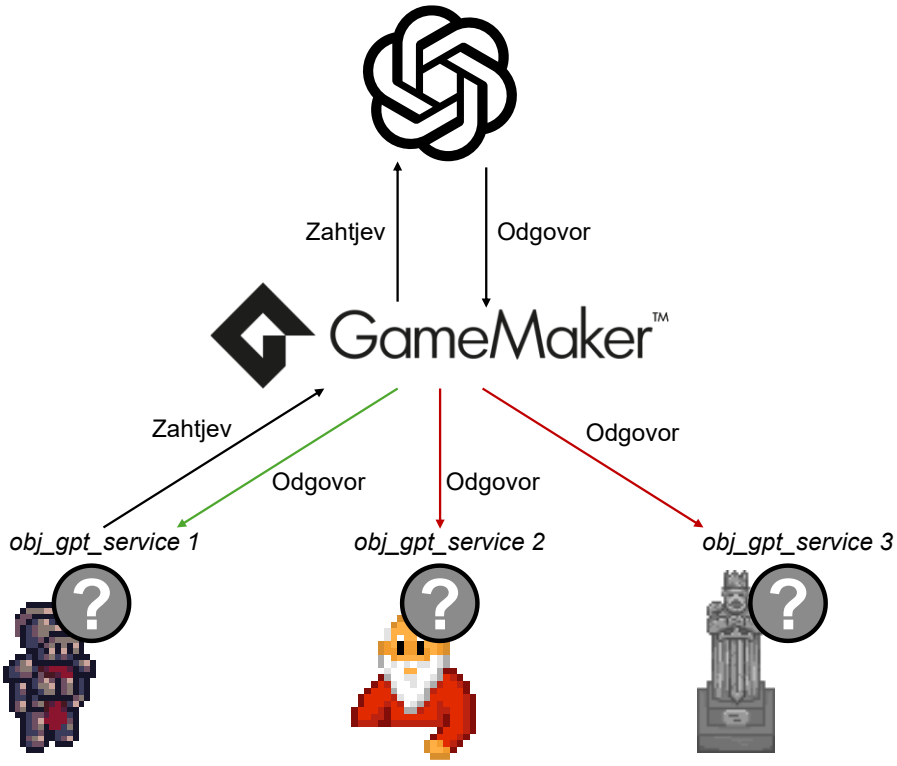
Kako bi se objašnjenje poslalo na *API*, dodaje se na početak ostalih poruka koje se šalju, neovisno o broju prethodnih poruka, u funkciji `send_message` prije samog pozivanja ugrađene funkcije `http_request`:

```
    if (includes_explaination) {
        array_insert(_data.messages, 0, {
            "role": "system",
            "content": explaination
        });
    }
```

Poruka s objašnjenjem šalje se kao poruka s ulogom „system“ jer se ta uloga koristi kada se želli poslati uputa na *API*.

5.2.2.5. Više instanci

Problem s *Async – HTTP* događajem je taj da se asinkroni događaji pozovu kod svih instanci tog objekta, iako je samo jedna instanca poslala HTTP zahtjev. [31] Ukoliko u videoigri postoji više instanci objekta „obj_gpt_service“ sa različitim objašnjenjem i prethodnim porukama, svaki za drugog *NPC*-a, dobiveni odgovori od strane *API*-ja će se pohraniti kod svih instanci:



Slika 12. Više instanci objekta *obj_gpt_service* primaju odgovor, iako je samo jedna instanca poslala zahtjev [autorski rad, OpenAI logo sa https://openai.com/brand/, GameMaker logo sa https://gamemaker.io/en/legal/brand]

Rješenje ovog problema je sljedeće: u *Create* događaju inicijalizirana je varijabla `waiting_for_response` s vrijednošću *false* te ona označava čeka li trenutna instanca *Async – HTTP* odgovor:

```
waiting_for_response = false;
```

Kada se u funkciji `send_message` pošalje HTTP zahtjev, postavi se vrijednost varijable `waiting_for_response` na *true*, što označava da ta instanca čeka na odgovor:

```
waiting_for_response = true;
```

Kako bi se spriječilo obrađivanje dobivenog odgovora pri dobivanju u *Async – HTTP* metodi instance koja nije poslala zahtjev, na početku tog događaja dodaje se sljedeća linija:

```
if (!waiting_for_response) exit;
```

Sada se odgovor obrađuje samo kod one instance objekta „obj_gpt_service“ koja je poslala zahtjev. Problem koji i dalje nije riješen je problem slanja više zahtjeva u isto vrijeme, jer će onda više instanci čekati na odgovor, no po samom dizajnu i ideji videoigre ne može doći do slučaja u kojemu će dvije instance poslati *API* zahtjev. Igrač može u svakom trenutku razgovarati sa samo jednim *NPC*-om te ne može izaći iz razgovora s tim *NPC*-om dok se u razgovoru čeka na odgovor *API*-ja.

5.2.3. Jednostavno sučelje za testiranje *API*-ja

Nakon što je kreiran objekt „obj_gpt_service“, da bi ga se testiralo, potrebno je imati druge objekte u videoigri koji ga koriste. Prije kreiranja same videoigre sa igračem i *NPC*-ovima koji razgovaraju, kreirana su 4 objekta koja služe kao jednostavni elementi grafičkog sučelja i mogu im se postaviti radnje koje se rade na klik mišem, ili koji pohranjuju unesen tekst:

- *obj_button* – gumb koji obavlja radnju na klik.
- *obj_checkbox* – potvrdni okvir koji obavlja dvije radnje, ovisno o stanju na koji je postavljen. Služi za postavljanje opcije da se pamti razgovor koristeći funkciju `set_conversation_remember` objekta „obj_gpt_service“.
- *obj_text_field* – ispisuje postavljen tekst, služit će za ispisivanje odgovora dobivenog od strane *API*-ja.
- *obj_text_input* – omogućuje unos teksta, unosit će se tekst koji se prenosi na *API* putem objekta „obj_gpt_service“.

Objekti su postavljeni u sobu „rm_chat“ zajedno sa instancom objekta „obj_gpt_service“ koju će ostali objekti koristiti. Primjer programskog koda za slanje teksta koristeći funkciju `send_message` objekta „obj_gpt_service“ te upravljanje dobivenim odgovorom:

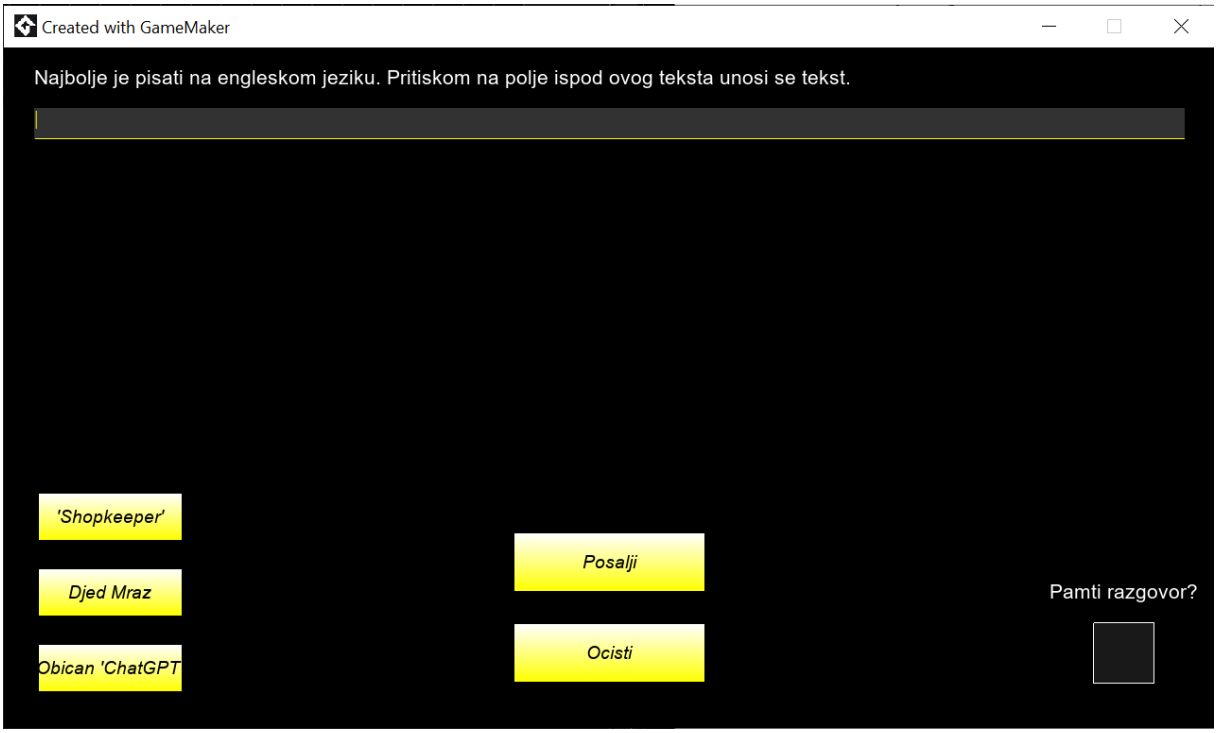
```
    my_gpt_service = instance_create_layer(x, y, layer, obj_gpt_service);
    text_field = instance_create_layer(room_width/2, room_height/2, layer, obj_text_field);
    button_text = "Posalji";
    my_gpt_service.set_response_handling(function(_response) {
        text_field.text = _response;
    });
    action = function() {
        if (string_length(obj_text_input.text) == 0) {
            text_field.text = "Ne moze se poslati prazan tekst!";
        } else {
            text_field.text = "...";
            my_gpt_service.send_message(obj_text_input.text, 1.0);
        }
    };
```

U sobi su također postavljena tri gumba gdje svaki od njih koristi funkciju `set_explaination` objekta „obj_gpt_service“ kako bi napisao *prompt* te omogućio da *API* odgovara na taj način. Primjer programskog koda koji postavlja objašnjenje na trgovca:

```
    width = 150;
    height = 48;
```

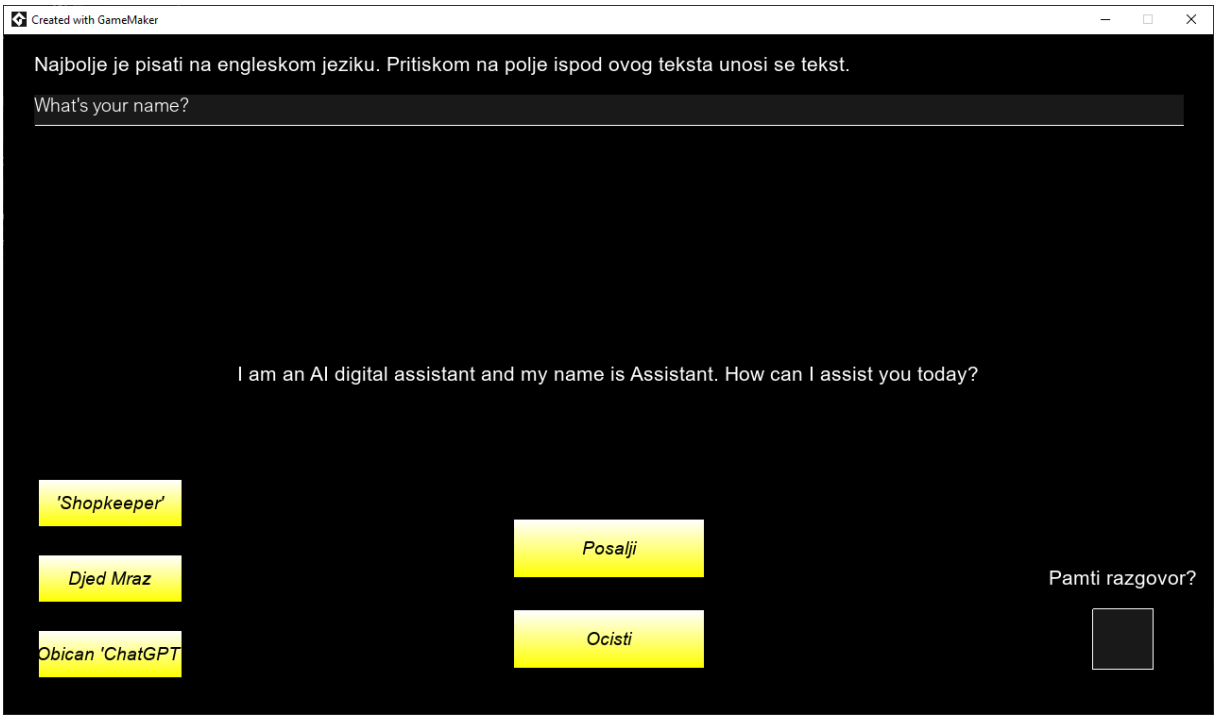
```
    button_text = "'Shopkeeper'";
    action = function() {
        obj_gpt_service.set_explaination("You are a shopkeeper in a small, cozy shop in a tiny village in a video game. Your responses should mirror that and be a bit grand, but still down to earth. Also, do not answer in more than 3 sentences at any time.");
        with (obj_text_field) {
            if (x == 40 && y == 112) {
                instance_destroy();
            }
        }
        with (instance_create_layer(40, 112, layer, obj_text_field)) {
            horizontal_center = fa_left;
            text = "Sada ce odgovori biti u stilu kao da ih govori 'shopkeeper' u maloj trgovini u igrici.";
            alarm[0] = 160;
        }
    }
```

Izgled konačnog sučelja je sljedeći:



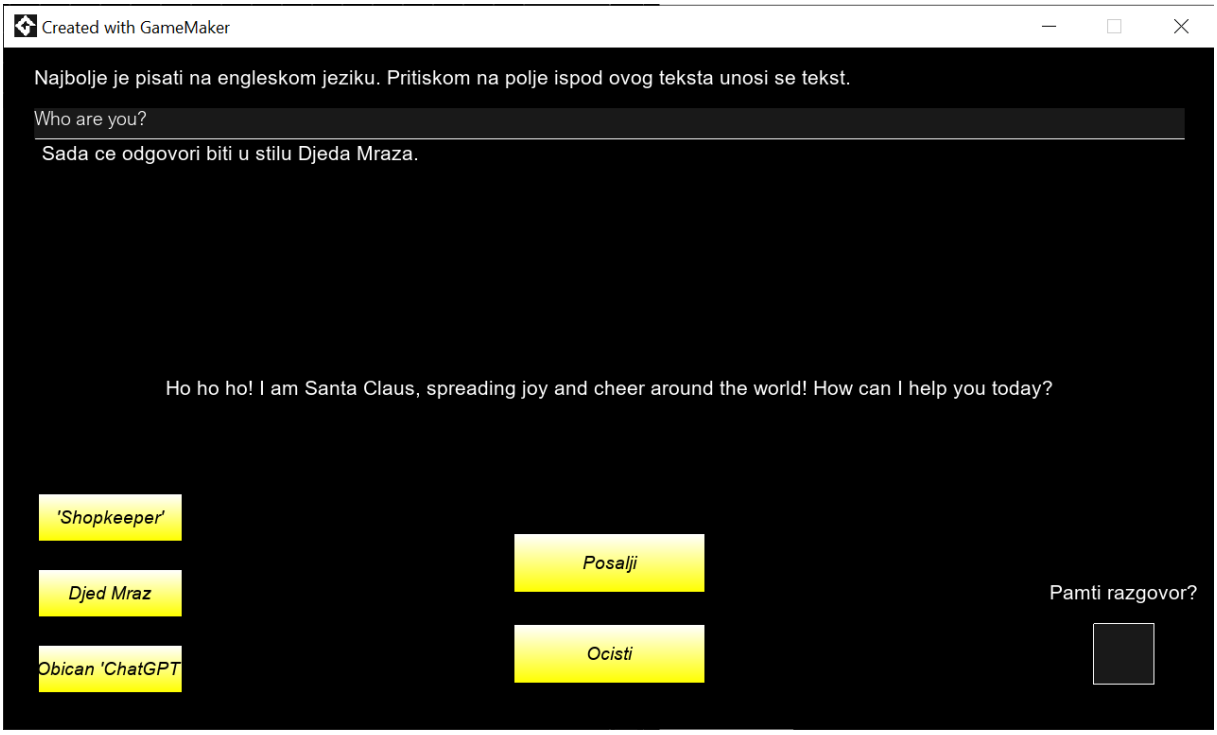
Slika 13. Sučelje za korištenje *API*-ja [autorski rad, slika zaslona]

Nakon napisanog pitanja te pritiska gumba „Posalji“, nakon kratkog vremena dobiva se odgovor:



Slika 14. Dobiven odgovor od strane *API*-ja [autorski rad, slika zaslona]

Pritiskom na gumb „Djed Mraz“ odgovori će biti generirani na takav način:



Slika 15. *API* šalje odgovore u stilu Djeda Mraza [autorski rad, slika zaslona]

5.3. Izrada osnovnih mehanika

Uspješno je izrađen objekt „obj_gpt_service“ koji služi za korištenje *API*-ja i prikazano je da se uspješno povezuje na *API*. Idući korak je izrada osnovne mehanike videoigre, kao što je igrač, kamera i *NPC*-ovi.

5.3.1. Igrač i kolizije

Za igrača je napravljen novi objekt „obj_player“ koji upravlja pokretima i animacijom igrača. Za kolizije je napravljen objekt „obj_collision“ koji predstavlja područje kroz koje igrač ne može proći.

Objekt „obj_collision“ je prazan objekt, ima samo postavljen *sprite* na sliku koja je puna, a služi kao maska za koliziju. Jedino svojstvo koje je osim *sprite*-a promijenjeno je to da je vidljivost (eng. *visibility*) isključena.

Objekt „obj_player“ ima osnovni programski kod za pokret. Varijable su inicijalizirane u *Create* događaju:

```
walk_speed = 1.2;
run_speed = 2;
hsp = 0;
vsp = 0;
can_move = true;
```

Varijabla `can_move` služi za onemogućavanje pokreta ukoliko je razgovor sa *NPC*-om u tijeku, kada će biti postavljena na *false*.

Osnovi kod za pokret i kolizije sa objektom „obj_collision“ je napravljen u *Step* događaju koji se izvršava svaki takt videoigre:

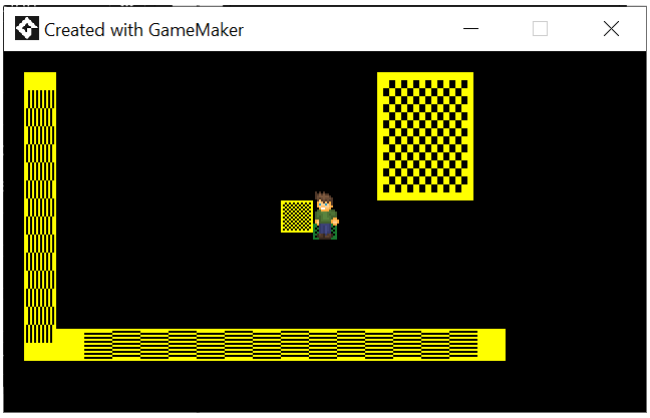
```
var _left_move_input = keyboard_check(vk_left) && can_move;
var _right_move_input = keyboard_check(vk_right) && can_move;
var _up_move_input = keyboard_check(vk_up) && can_move;
var _down_move_input = keyboard_check(vk_down) && can_move;
var _sprint_input = keyboard_check(vk_shift) && can_move;
var _horizontal_movement = _right_move_input - _left_move_input;
var _vertical_movement = _down_move_input - _up_move_input;
var _movement_angle = point_direction(0, 0, _horizontal_movement,
_vertical_movement);
var _modified_walk_speed = (_horizontal_movement == 0 &&
_vertical_movement == 0) ? 0
                           : (_sprint_input ? run_speed : walk_speed);
hsp = lengthdir_x(_modified_walk_speed, _movement_angle);
```

```
    vsp = lengthdir_y(_modified_walk_speed, _movement_angle);
    if (hsp != 0) {
        if (collision_rectangle(bbox_left + hsp + sign(hsp), bbox_top,
bbox_right + hsp + sign(hsp), bbox_bottom, obj_collision, false, true)) {
            if (instance_place(floor(x) + (sign(hsp) == -1), y,
obj_collision) == noone) x = floor(x) + (sign(hsp) == -1);
            while (instance_place(x + sign(hsp), y, obj_collision) ==
noone) x += sign(hsp);
            hsp = 0;
        }
    }
    if (vsp != 0) {
        if (collision_rectangle(bbox_left, bbox_top + vsp + sign(vsp),
bbox_right, bbox_bottom + vsp + sign(vsp), obj_collision, false, true)) {
            if (instance_place(x, floor(y) + (sign(vsp) == -1),
obj_collision) == noone) y = floor(y) + (sign(vsp) == -1);
            while (instance_place(x, y + sign(vsp), obj_collision) ==
noone) y += sign(vsp);
            vsp = 0;
        }
    }
    x += hsp;
    y += vsp;
```

Igrač se sada može kretati u 8 smjerova. Programski kod za izmjenu slike igrača nije prikazan, no funkcionira tako da provjeri smjer kretanja te kreće li se igrač i promjenom vrijednosti ugrađene sljedećih varijabli: `sprite_index` koja određuje koji *sprite* će se prikazivati [32], `image_speed` koji određuje brzinu animacije kod prikazanog *sprite*-a [33] te `image_index` koji određuje koja pod-slika *sprite*-a se prikazuje [34].

Zadnja stvar koju je potrebno napraviti je postaviti masku kolizije (eng. *collision mask*) na igrača i *sprite* za „obj_collision“. Maska kolizije je područje koje se koristi za kalkulaciju „dodiruju“ li se dva objekta, ili ne. [35] Za „obj_collision“ je maska kolizije postavljena na *Full image* i *Rectangle*, a za igrača je postavljeno svojstvo “mask_index” na novi *sprite* koji mu definira masku, kako se ne bi moralo na svakom od 8 *sprite*-ova za igrača, ovisno o smjeru u kojemu hoda, definirati ista maska.

Kolizije i igrač mogu se staviti u novu sobu te ako se prikažu sve instance „obj_collision“ te igračeva maska za kolizije, rezultat je sljedeći:



Slika 16. Igrač i instance kolizija [autorski rad, slika zaslona]

5.3.2.Kamera

„Kamera“ se odnosi na novi objekt, „camera“ koji će služiti za praćenje igrača i postavljanje pogleda tako da je igrač uvijek u sredini. U konačnoj videoigri objekt „camera“ ima više uloga, kao što je postavljanje fontova, crtanje svih instanci sortirano odozgo prema dolje kako bi se ispravno prikzivali oni „niže“ preko onih „više“, ali i glavna funkcionalnost, što je praćenje igrača. Objekt „camera“ bit će postavljen na *Persistent*, kako bi postojao i ako se promijeni soba, tako da ga je potrebno samo jednom postaviti u početnu sobu igre, a ne u svaku posebno [15]. Da bi se igrač pratio, u *Room Start* događaju kamere, što je događaj koji se pokreće pri svakom pokretanju sobe, inicijaliziraju se sve potrebne varijable:

```
cam = camera_create();
globalvar view_w, view_h, view_x, view_y, zoom;
view_w = 320;
view_h = 180;
view_x = 0;
view_y = 0;
zoom = 1;
view_camera[0] = cam;
view_enabled = true;
view_visible[0] = true;
display_set_gui_size(320, 180);
```

U *End Step* događaju je logika za praćenje igrača:

```
var _view_width = floor(view_w*zoom);
var _view_height = floor(view_h*zoom);
if (instance_exists(obj_player)) {
    var _player_x = obj_player.x - (_view_width*0.5);
```

```
        var _player_y = (obj_player.bbox_top + obj_player.bbox_bottom)*0.5
- (_view_height*0.5);
        view_x = _player_x;
        view_y = _player_y;
    }
    camera_set_view_pos(view_camera[0], view_x, view_y);
    camera_set_view_size(view_camera[0], _view_width, _view_height);
```

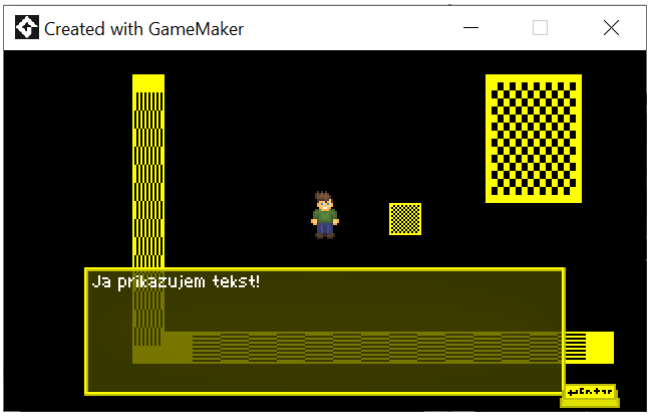
Pošto je kamera dinamičan resurs, (u varijabli `cam`), mora se pri prestanku korištenja uništiti pozivajući funkciju `camera_destroy`. [36] U *Room End* događaju poziva se ta funkcija:

```
    camera_destroy(view_camera[0]);
```

Sada se igrač nalazi u sredini ekrana te ako je soba veća od prozora, vidi se samo dio određene veličine i igrač je uvijek u sredini.

5.3.3. Tekstualni prozor

Tekstualni prozor služi za prikazivanje teksta igraču unutar igre. Tekstualni prozor ima sljedeći izgled:



Slika 17. Tekstualni prozor u videoigri [autorski rad, slika zaslona]

Za tekstualni prozor napravljen je novi objekt „obj_textbox“ koji ispisuje tekst slovo po slovo i može mu se postaviti boja. Tekst se može preskočiti pritiskom gumba „Shift“ ili „Enter“, a kada je cijeli tekst ispisan, instanca objekta „obj_textbox“ uništava se. Da bi se tekstualnom prozoru postavio tekst koji se ispisuje i boja teksta, u *Creation Code* događaju:

```
    text = "Ovo je dugačak tekst koji se još nije ispisao do kraja, ima još
toga za ispisati!";
    color = c_lime;
```

Tekstualni prozor sa drugim tekstom i bojom izgleda ovako:

Slika 18. Tekstualni prozor sa drugom bojom i tekstom koji se još ispisuje [autorski rad, slika zaslona]

U suštini, „obj_textbox“ samo ispisuje tekst i crta pozadinu prozora u *Draw GUI* događaju:

```
    var _buffer = 8;
    var _x = display_get_gui_width()/2 - sprite_width/2;
    var _y = display_get_gui_height() - sprite_height - _buffer;
    var _tb_border_size = 2;
    draw_sprite_ext(spr_textbox, 0, _x, _y, 1, 1, 0, color, 1);
    var _text_buffer = 4;
    var _text_sep = 10;
    var _text_width = sprite_width - 2*_text_buffer;
    draw_set_font(global.pixel_font);
    var _text_height = string_height(" ");
    draw_text_ext(_x + _text_buffer, _y + _text_buffer, showing_text,
_text_sep, _text_width);
```

Osim ovoga, objekt „obj_textbox“ crta i pomoćne gumbe te koristeći *Alarm* događaj, koji se aktiviraju svakih nekoliko taktova, prikazuje sve više teksta:

```
    var _current_string = string_copy(text, 1, string_length(showing_text)
+ 1)
    showing_text =_current_string;
    alarm[0] = text_reveal_speed;
```

Radi potrebe videoigre, u tekstualni prozor dodano je još nekoliko funkcionalnosti. Jedna od funkcionalnosti je ispisivanje i mogućnost biranja opcija:

Slika 19. Tekstualni prozor u kojemu se mogu birati opcije [autorski rad, slika zaslona]

Odabrana opcija pohranjuje se u varijablu `chosen_action` objekta „obj_textbox“.

Kako je unos teksta radi komunikacije s *API*-jem bitan dio, kreiran je objekt „obj_text_input_box“ koji služi za tekstualan unos:

Slika 20. Prozor za unos teksta [autorski rad, slika zaslona]

Sada je moguće prikazivati tekst unutar videoigre te igrač može unijeti tekst.

5.3.4.NPC objekti

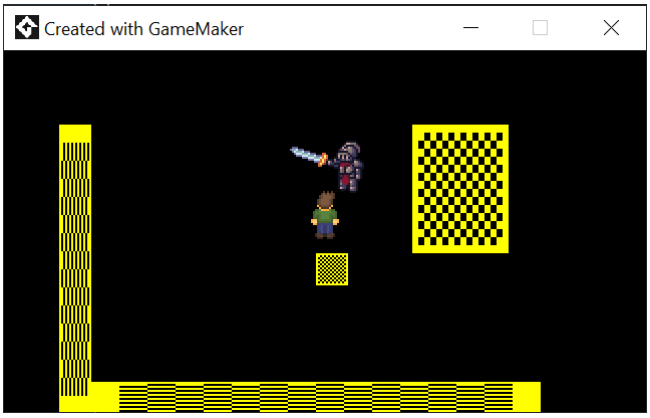
NPC-ovi u videoigri napravljeni su tako da je svaki *NPC* vlastiti objekt, konkretnije, napravljeno ih je dvoje:

- „obj_knight“ – vitez koji čuva dvorac
- „obj_merchant“ – trgovac s kojim igrač može razgovarati

Objekti za *NPC*-ove za sada nemaju nikakvu posebnu logiku, osim što kreiraju instancu objekta „obj_solid“ sa dimenzijama njihove maske kolizije, kako igrač ne bi mogao proći kroz *NPC*-a:

```
    my_coll = instance_create_layer(bbox_left, bbox_top, layer,
obj_collision);
    var _w = (bbox_right - bbox_left);
    var _h = (bbox_bottom - bbox_top);
    with my_coll {
            image_xscale = _w/sprite_width;
            image_yscale = _h/sprite_height;
            visible = false;
    }
```

Dodatna logika postoji kod nekih *NPC*-ova, kao što vitez ispruži mač mijenjajući *sprite* ukoliko je blizu igrača:



Slika 21. Vitez koji je ispružio mač [autorski rad, slika zaslona]

5.4. Integracija *API*-ja u videoigru

Videoigra trenutno ima sve elemente potrebne da ju se može igrati. Kreiran je i objekt „obj_gpt_service“ koji služi za povezivanje i korištenje *API*-ja, no potrebno je ta dva elementa videoigre povezati zajedno, a upravo to povezivanje je tema ovog poglavlja.

5.4.1. Kreiranje objekta za scene

Kako bi igrač mogao imati interakcije sa *NPC*-ima te kako bi interakcije mogle biti dinamične, potrebno je napraviti objekt za scene (eng. *cutscene*). Prema uzoru videozapisa na poveznici na [37] kreiran je objekt „obj_cutscene“ koji će služiti za upravljanje scenama.

U *Create* događaju objekta postavlja se polje sa scenama, nazvano `scenes` kao i varijabla „current_scene“, koja označava trenutnu scenu koja se prikazuje (interpretira se kao indeks polja `scenes`). Napravljene su i dodatne funkcije koje se koriste za scene:

```
scenes = [];
current_scene = 0;
if (instance_exists(obj_player)) {
    with (obj_player) {
        can_move = false;
    }
}
end_action = function() {};
locked = false;
```

U *Step* događaju pokreću se scene popisane u polju `scenes` dok god je vrijednost varijable „locked“ postavljena na *false*:

```
while (!locked) {
    var _has_label = array_length(scenes[current_scene]) > 2;
    script_execute_ext(scenes[current_scene][0 + _has_label],
scenes[current_scene][1 + _has_label]);
}
```

`Locked` varijabla služi kako bi se, ukoliko se pokreće nekoliko radnji scene koje traju 1 takt igrice, pokrenule sve u istom taktu, umjesto da se svaki takt pokrene jedna radnja, što je izazivalo sitan, ali primjetan učinak da ako se npr. prikaže tekstualan prozor, pa se pozove funkcija koja pomiče nekog *NPC*-a te se opet prikazuje tekstualan prozor, taj jedan takt igre dok se izvršavala funkcija za pomicanje *NPC*-a nije postojao nikakav tekstualni prozor te je izgledalo kao da tekstualan prozor na trenutak nestane. Kako se koristi varijabla `locked` te se sve izvršava u *while* petlji, kod ovog primjera bi se u istom taktu pozvala metoda za pomicanje *NPC*-a i prikazao tekstualan prozor te ne bi bilo tog trenutka kad ne stane.

Da bi se igrač mogao kretati nakon kraja *cutscene*-a, u *Destroy* događaju objekta „obj_cutscene“ napisan je sljedeći programski kod:

```
if (instance_exists(obj_player)) {
    with (obj_player) {
        can_move = true;
    }
}
```

Funkcije koje se mogu pozivati kao pojedine scene napisane su u *script* resursima kao funkcije. Funkcija za završavanje trenutne scene i prijelaz na iduću scenu je sljedeća:

```
function cutscene_end_scene(_cutscene_obj = id) {
    with (_cutscene_obj) {
        current_scene ++;
        if (current_scene >= array_length(scenes)) {
            end_action();
```

```
            instance_destroy();
        }
    }
}
```

Funkcija za prikazivanje tekstualnog prozora za scenu je sljedeća:

```
function cutscene_create_textbox(_text, _color) {
    var _end_action = function () {
        locked = false;
        cutscene_end_scene();
    }
    if (!instance_exists(obj_textbox)) {
        locked = true;
        with (instance_create_layer(x, y, layer, obj_textbox)) {
            text = _text;
            color = _color;
            end_action = _end_action;
        }
    }
}
```

Funkcija za „skakanje“ na neku drugu scenu, umjesto da scene idu po redu je sljedeća:

```
function cutscene_jump(_label, _cutscene_obj = id) {
    with (_cutscene_obj) {
        for (var _i = 0; _i < array_length(scenes); _i ++) {
            if (array_length(scenes[_i]) == 3) {
                if (scenes[_i][0] == _label) {
                    current_scene = _i;
                    return;
                }
            }
        }
        end_action();
        instance_destroy();
    }
}
```

Ovaj sustav omogućuje jednostavno pisanje univerzalnih *cutscene*-a te jednostavno dopunjavanje novim mogućnostima samo pišući nove funkcije za pojedinu scenu. Tako su još dopisane funkcije za prikazivanje prozora za tekstualni unos, kreirajući instancu prethodno kreiranog objekta „obj_text_input_box“ te razne druge potrebne funkcije, kao što je funkcija za biranje opcija koristeći tu funkcionalnost objekta „obj_textbox“.

Pisanje scene je jednostavno koristeći ovaj objekt i radi se popunjavajući polje `scenes` instance objekta „obj_cutscene“ u *Creation Code* događaju:

```
    scenes = [
        ["pocetak", cutscene_create_textbox, ["Bok, ovo je scena!", c_red]],
        [cutscene_create_textbox_actions, ["Gdje želiš ići?", c_lime, ["Početak", "Kraj"], ["pocetak", "zadnja"]]],
        ["zadnja", cutscene_create_textbox, ["Doviđenja!", c_yellow]],
    ];
```

Ova scena će se sad izvršavati u videoigri, prikazujući prvo prvi i drugi tekstualni prozor te ovisno o odabiru na drugom tekstualnom prozoru, otići opet na početak, ili na kraj, nakon kojeg se instanca objekta „obj_cutscene“ uništava i igrač se opet može kretati.

5.4.2. Povezivanje scena na *NPC*-ove

Kako bi se *cutscene* povezao na *NPC*-a, kreiran je objekt „obj_talkable“ koji, ukoliko mu se igrač približi, prikazuje tipku na tipkovnici „E“, odnosno prikazuje da se može pritisnuti te upravlja time da ako se pritisne tipka „E“, kreira se instanca objekta „obj_cutscene“ i prenose mu se definirane scene za tu instancu objekta „obj_talkable“. *Create* događaj objekta „obj_talkable“ je sljedeći:

```
    bound_npc = noone;
    in_talking_range = false;
    talking_range = 20;
    global.currently_talking = false;
    color = obj_player.main_color;
    extra_key_y = 0;
    scenes = [];
    my_cutscene = noone;
    disabled = false;
```

End Step događaj koji upravlja provjerom je li igrač blizu te pokreće *cutscene* ako se pritisne gumb „E“ je sljedeći:

```
    in_talking_range = false;
    if (global.currently_talking) exit;
    if (disabled) exit;
    if (obj_player.y < y) {
        exit;
    };
    if ((obj_player.bbox_right - 1) >= bound_npc.bbox_left && (obj_player.bbox_left + 1) <= bound_npc.bbox_right) {
```

```
        if (abs(bound_npc.bbox_bottom - obj_player.bbox_top) <=
talking_range) {
            in_talking_range = true;
        }
    }
    if (keyboard_check_pressed(ord("E"))) {
        if (in_talking_range) {
            if (!instance_exists(my_cutscene)) {
                my_cutscene = instance_create_layer(x, y, layer,
obj_cutscene);
                var _scenes = scenes;
                var _id = id;
                with (my_cutscene) {
                    scenes = _scenes;
                    caller = _id;
                }
                with (obj_player) {
                    sprite_index = spr_player_walk_up
                }
                global.currently_talking = true;
            }
        }
    }
```

Draw End događaj koji crta gumb „E“ ako je igrač blizu je sljedeći:

```
    if (in_talking_range) {
        var _key_sprite = spr_key_e;
        var _y = bound_npc.bbox_top - bound_npc.sprite_height -
extra_key_y;
        draw_sprite_ext(_key_sprite, 1, x, _y, 1, 1, 0, color, 1);
        draw_sprite_ext(_key_sprite, 0, x, _y, 1, 1, 0, c_black, 1);
    }
```

Iako je u *GameMaker Studio 2* moguće nasljeđivanje [15], dodavanje razgovora sa *NPC* objektima neće biti kreirano tako da ti *NPC* objekti nasljeđuju „obj_talkable“ zbog dva razloga:

1. Ako se nasljeđuje objekt, svi događaji tog objekta moraju se „ručno“ izvršiti koristeći ugrađenu funkciju `event_inherited()` u istom događaju djeteta, ukoliko je definirano [38], što zahtijeva izmjenu svakog djeteta koji nasljeđuje taj objekt.
2. *NPC* objekti u videoigri već nasljeđuju instancu objekta „obj_sortable“ koji je prethodno kreiran i služi samo kako bi se ispravno prikazivali objekti jedni preko drugih.

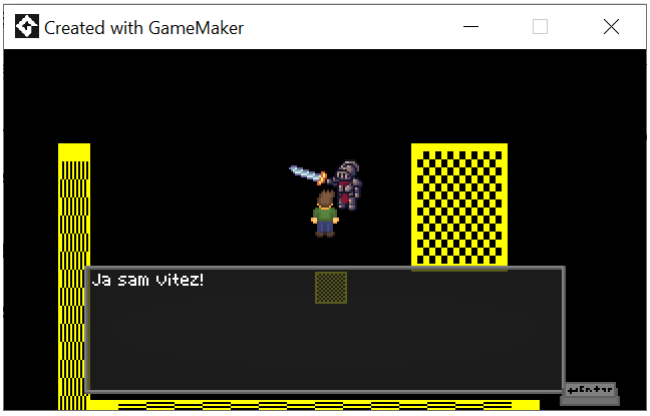
Način na koji će *NPC* objekti koristiti događaje i funkcije „obj_talkable“ je sličan načinu na koji kreiraju vlastitu instancu objekta „obj_collision“: kreirat će vlastitu instancu objekta „obj_talkable“ i prenijeti mu svoju scenu. Za tu svrhu kreirana je funkcija `talkable_init` koja se poziva u *Create* događaju *NPC*-a koji želi imati omogućen razgovor:

```
    function talkable_init(_scenes = [
        [cutscene_create_textbox, ["Bok, poseban razgovor nije postavljen",
c_yellow]]
    ], _talking_range = 20, _extra_key_y = 0) {
        my_talkable = instance_create_layer(x, y, layer, obj_talkable);
        var _id = id;
        with (my_talkable) {
            bound_npc = _id;
            talking_range = _talking_range;
            extra_key_y = _extra_key_y;
            scenes = _scenes;
        }
        function talkable_set_scene(_new_scenes) {
            my_talkable.scenes = _new_scenes;
        }
        function talkable_disable() {
            my_talkable.disabled = true;
        }
    }
```

Kako bi *NPC* prenio svoj razgovor u svoju instancu „obj_talkable“, to obavlja pozivom funkcije `talkable_set_scene` u *Creation Code* događaju:

```
    talkable_set_scene([[cutscene_create_textbox, ["Ja sam vitez!",
c_gray]]]);
```

Sada je moguće doći do viteza, pritisnuti gumb „E“ te se prikazuje sivi tekstualni okvir sa prikazanim tekstom „Ja sam vitez!“:



Slika 22. Razgovor s vitezom [autorski rad, slika zaslona]

5.4.3. Metode za korištenje *API*-ja u scenama

Za povezivanje objekta „obj_gpt_service“ i *cutscene*-a, prvo je izmijenjen objekt „obj_textbox“ dodavši mu varijablu `ready` koja je *false* ako se čeka na odgovor *API*-ja te u tome slučaju tekstualni prozor ispisuje „. . .“.

Objekt „obj_cutscene“ nadograđen je tako da se u argumente za svaku scenu može unijeti oznaka unutar < i > te se kod drugih scena taj dio ulaznog argumenta može zamijeniti drugim tekstom, koji može doći od neke druge funkcije za scenu, kao što je funkcija `cutscene_create_text_input_box`, koja mijenja oznaku sa unesenim nazivom u argumentima drugih funkcija scena. U *Create* događaju kreirane su sljedeće funkcije:

```
tags = [];
function remove_tag(_tag_name) {
    for (var _i = 0; _i < array_length(tags); _i ++) {
        if (tags[_i].tag_name == _tag_name) {
            array_delete(tags, _i, 1);
            _i --;
        }
    }
}
function add_tag(_tag_name, _replace_with) {
    remove_tag(_tag_name);
    array_push(tags, {
        tag_name: _tag_name,
        replace_with: _replace_with,
    });
}
```

Step događaj za izvršavanje funkcija scena modificiran je kako bi se argumenti izmijenili ovisno o tim oznakama:

```
    while (!locked) {
        var _args = [];
        var _has_label = array_length(scenes[current_scene]) > 2;
        array_copy(_args, 0, scenes[current_scene][1 + _has_label], 0,
array_length(scenes[current_scene][1 + _has_label]));
        for (var _i = 0; _i < array_length(_args); _i ++) {
            for (var _j = 0; _j < array_length(tags); _j ++) {
                if (is_string(_args[_i])) {
                    _args[_i] = string_replace(_args[_i], "<" +
tags[_j].tag_name + ">", tags[_j].replace_with);
                }
            }
        }
        script_execute_ext(scenes[current_scene][0 + _has_label], _args);
    }
```

Funkcija `cutscene_create_text_input_box` koristi upravo novu funkciju `add_tag` pri unosu teksta i unesen tekst postavlja kao vrijednost oznake sa unesenim nazivom.

Funkcija `cutscene_create_textbox_response` kreira tekstualni prozor koji aktivno čeka na odgovor *API*-ja:

```
    function cutscene_create_textbox_response(_request, _color,
_gpt_service, _temperature = 1, _role = "user") {
        var _end_action = function () {
            locked = false;
            cutscene_end_scene();
        }
        saved_request = _request;
        saved_temperature = _temperature;
        saved_role = _role;
        var _tb = noone;
       if (!instance_exists(obj_textbox)) {
            locked = true;
            _tb = instance_create_layer(x, y, layer, obj_textbox);
            var _cutscene_obj = id;
            with (_tb) {
                color = _color;
                end_action = _end_action;
                instruction_tag = _instruction_tag;
                ready = false;
```

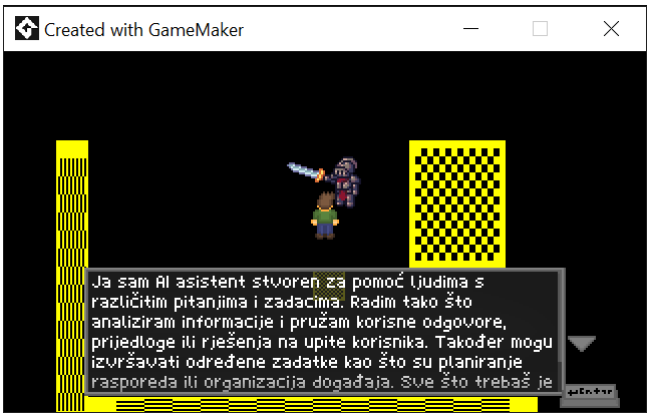
```
                cutscene_object = _cutscene_obj;
            }
        }
        with (_gpt_service) {
            if (_tb != noone) {
                textbox = _tb;
                send_message(_request, _temperature, _role);
                set_response_handling(function (_response) {
                    with (textbox) {
                        while (string_count("\n\n", _response) > 0) {
                            _response = string_replace(_response, "\n\n",
"\n");
                        }
                        while (string_count("`", _response) > 0) {
                            _response = string_replace(_response, "`", "");
                        }
                        text = _response;
                        event_user(1);
                    }
                });
            }
        }
    }
```

Sada je moguće povezati *NPC*-a na *API* pišući scene na sljedeći način:

```
    var _gpt_service = instance_create_layer(x, y, layer, obj_gpt_service);
    talkable_set_scene([
        ["input", cutscene_create_text_input_box, ["Unesi što želiš
pitati.", c_yellow, "enteredText"]],
        [cutscene_create_textbox_response, ["<enteredText>", c_gray,
_gpt_service, 1, "user"]],
        [cutscene_jump, ["input"]]
    ]);
```

Prozor za tekstualni ulaz sprema unesen tekst u oznaku „enteredText“ te se ta vrijednost koristi kao ulazni argument za funkciju `cutscene_create_textbox_response`, kao pitanje što će ta funkcija slati na *API*.

Vitez sada odgovara kao *ChatGPT*:

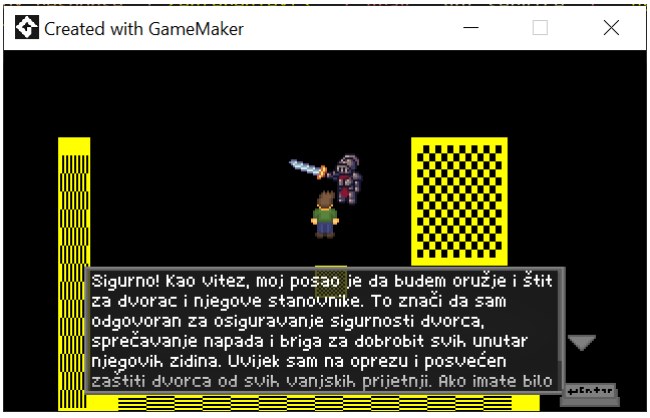


Slika 23. Odgovor *API*-ja kroz lik viteza na pitanje “Kako si i što radiš?” [autorski rad, slika zaslona]

Vitez sada odgovara kao *ChatGPT* i ne pamti tekst, no to je jednostavno za napraviti koristeći funkciju `set_explaination` objekta „obj_gpt_service“ te koristeći funkciju `set_conversation_remember`:

```
var _gpt_service = instance_create_layer(x, y, layer, obj_gpt_service);
_gpt_service.set_explaination("Ti si vitez koji čuva dvorac.");
_gpt_service.set_conversation_remember(true, 5);
```

Sada *NPC* viteza odgovara kao pravi vitez:



Slika 24. Odgovor *API*-ja na drugo pitanje „Možeš li pojasniti?“, nakon što je prethodno odgovorio na pitanje „Tko si ti?“ [autorski rad, slika zaslona]

Još jedan problem koji je potrebno riješiti je taj da je trenutno implementiran razgovor beskonačan, jer se nakon svakog dobivenog odgovora ponovno vraća na unos pitanja. Kraj razgovora bi se mogao implementirati pitanjem korisnika želi li nastaviti koristeći tekstualni prozor s opcijama, no implementacijom tog rješenja zaključeno je da nije toliko intuitivan i

naporno je stalno odabirati opciju za nastavak razgovora želi li igrač voditi duži razgovor. Kao rješenje ovog problema, na *API* je poslana poruka s ulogom „system“ da na kraju odgovora, ukoliko igrač odlazi, napiše uputu „&&KRAJ&&“:

```
    _gpt_service.set_explaination("Ti si vitez koji čuva dvorac. Ako igrač
kaže da odlazi, ili želi otići, dodaj '&&KRAJ&&' na kraj odgovora.");
```

Uputu je potrebno dohvatiti u funkciji za obradu odgovora od strane *API*-ja kod prikazivanja u tekstualni prozor te pohraniti uputu u objekt „obj_gpt_service“, tako da se može koristiti kod određivanja scena:

```
if (string_count("&&", _response) > 0) {
    var _instruction_text_parts = string_split(_response, "&&", true);
    array_push(gpt_service_instructions, {
        "tag": instruction_tag,
        "replace": string_lower(_instruction_text_parts[1])
    });
    _response = string_trim(_instruction_text_parts[0]);
}
```

Svaki taj odgovor sprema se kao oznaka za postojeći *cutscene*:

```
if (array_length(gpt_service_instructions) > 0) {
    array_foreach(gpt_service_instructions, function (_el, _ind) {
        cutscene_add_tag(_el.tag, _el.replace, cutscene_object);
    });
}
```

Sada je moguće dohvatiti i obraditi dobivene upute *API*-ja.

Završena je funkcionalnost slobodnog razgovora te je moguće implementirati da svaki *NPC* ima različitu ulogu i način razgovora te je moguće reći *API*-ju da na kraju odgovora napiše uputu, koju je moguće dohvatiti i obraditi. Korištenjem ovog sistema mogu se napraviti i kompleksnije radnje, kao što je pogađanje lozinke. Primjer upute *API*-ju za viteza u konačnoj implementaciji:

```
"Ti si vitez koji štiti dvorac. Trebaš odgovarati kratko i sažeto. Štitiš
ulaz u dvorac. Korisnik će pokušati pričati s tobom, i pokušati ući, ali ti
stojiš čvrsto, i ne puštaš ga da uđe. Možeš ga pustiti da uđe samo ako
točno pogodi lozinku. Lozinka je 'Excalibur', ili 'Ekskalibur', gdje
veličina slova nije bitna. Daj korisniku maksimalno jednu pomoć oko lozinke
ako pita, ne govori mu cijelu lozinku osim ako je pogođena. Nikada, ni pod
kojim uvjetom, nemoj izaći iz uloge i reći da si robot, ili AI. Također,
drži svoje odgovore kratke i jednostavne za razumjeti, dok si i dalje u
ulozi. Ako korisnik točno odgovori lozinku, molim dodaj '&&GUESSED&&' na
sam kraj odgovora."
```

5.4.3.1. Biranje opcija

Iako je slobodan razgovor ekspresivniji te igraču daje ogromne mogućnosti za postavljanje pitanja te razgovora sa *NPC*-om, nekada nije moguće pokriti sve moguće scenarije ili pitanja koje bi igrač mogao postaviti. To je jedan razlog zašto je uputa za viteza u videoigri toliko stroga, odnosno dopušteno mu je odgovarati samo na mali skup pitanja. Drugi način da se razgovara s *NPC*-om je kroz postojeće opcije koje imaju dodanu prednost da se za svaku opciju može posebno *API*-ju naglasiti što da odgovori i dati dodatne upute, umjesto da se mnogo informacija pošalje u početnoj sistemskoj poruci.

Ključna stvar kod korištenja tekstualnog prozora s odabirom opcija, umjesto da se unosi tekst je ta da se šalju poruke s ulogom „system“. Kreiran je novi *NPC* „obj_merchant“ koji će se nalaziti blizu dvorca i koji može razgovarati s igračem o raznim temama. *Creation Code* događaj tog *NPC*-a sa podebljanim dijelom bitnim za slanje sistemske poruke je sljedeći:

```
    var _gpt_service = instance_create_layer(x, y, layer, obj_gpt_service);

    _gpt_service.set_explaination("Ti si srednjovjekovni trgovac koji ima
svoj štand s blagajnom, blizu velikog dvorca. Pričljiv si, i prijateljski.
Korisnik će doći do tebe i pitati te o raznim stvarima, najviše o velikom
dvorcu. Dvorac je čuvan od viteza. Ti si mudar koliko i srednjovjekovni
čovjek u srednjim godinama, i imaš nešto mudrosti. NIKAD, ni pod kojim
uvjetom prestani biti u ulozi i nemoj reći da si robot, ili AI. Također,
nemoj davati više informacija nego što bi trebao znati. Korisnik je također
mlađa osoba, luta šumom i vjerojatno pokušava ući u dvorac.");

    _gpt_service.set_conversation_remember(true, 5);

    talkable_set_scene([

        ["odabir", cutscene_create_textbox_actions, ["O čemu želiš
pričati?", c_yellow, ["'Reci mi nešto o sebi.'", "O vitezu", "O dvorcu",
"Završi razgovor"], ["razgovor1", "razgovor2", "razgovor3", "kraj"]]],

        ["razgovor1", cutscene_create_textbox_response, ["Igrač te pitao da
mu kažeš nešto više o sebi. Nemaš kosu, imaš bradu i crvenu majicu, a štand
ti je drven i ima modernu blagajnu.", c_red, _gpt_service, 1.2, "system"]],

        [cutscene_jump, ["odabir"]],

        ["razgovor2", cutscene_create_textbox_response, ["Igrač te je pitao
nešto o vitezu. Reci mu da je ustrajan i čuva dvorac te nije baš za
razgovor, a ukoliko igrač želi ući u dvorac, mora pogoditi lozinku.",
c_red, _gpt_service, 1.2, "system"]],

        [cutscene_jump, ["odabir"]],

        ["razgovor3", cutscene_create_textbox_response, ["Igrač te je pitao
nešto o dvorcu. Reci da ne znaš baš puno.", c_red, _gpt_service, 1.2,
"system"]],

        [cutscene_jump, ["odabir"]],

        ["kraj", cutscene_end, []],

    ]);
```

Tekstualni prozor s odabirima vidljiv je na sljedećoj slici:

Slika 25. Opcije za razgovor s trgovcem [autorski rad, slika zaslona]

Ukoliko se odabere jedna od opcija, ispisuje se generiran odgovor, vođen dodatnom uputom poruke s ulogom „system“:

Slika 26. Trgovac odgovara na pitanje u vezi viteza [autorski rad, slika zaslona]

Sada je moguće razgovarati sa *API*-jem kroz biranje više opcija.

5.4.3.2. Slanje slike koristeći gpt-4o

GPT 4o uveo je mogućnost korištenja više resursa za ulaze, a ne samo tekstualnih ulaza. [4]. Kako bi se koristio, potrebno je u skripti „gpt_service_macros“ promijeniti makro *OPENAI_MODEL* na „gpt-4o“:

```
#macro OPENAI_MODEL "gpt-4o"
```

Treba imati na umu da je ovaj model deset puta skuplji od „gpt-3.5-turbo“. [9].

U objektu „obj_gpt_service“ kreirana je nova funkcija `send_image` koja služi za slanje slike na *API*:

```
function send_image(_image, _message, _temperature) {
```

```
        var _buffer = buffer_load(working_directory + "\\" + _image);
        var _base64_image = buffer_base64_encode(_buffer, 0,
buffer_get_size(_buffer));
        var _content = [
            {
                "type": "text",
                "text": _message
            },
            {
                "type": "image_url",
                "image_url": {
                    "url": "" + _base64_image
                }
            }
        ];
        send_message(_content, _temperature, „user“);
    }
```

Funkcija kao argument prima naziv datoteke slike koja se želi poslati, kao i tekstualnu uputu i temperaturu. Uloga je uvijek „user“. Sliku pretvara u base64 te kreira strukturu `_content` koja se dalje samo prosljeđuje prethodno kreiranoj funkciji za slanje zahtjeva, `send_message`.

Kreirana je nova funkcija za scenu zvana `cutscene_create_textbox_response_vision`, koja je slična prethodno kreiranoj funkciji `cutscene_create_textbox_response`, no umjesto metode `send_message` koristi metodu `send_image` i prosljeđuje sliku. Slika koja se prosljeđuje je slika zaslona cijele videoigre koja se dobiva na sljedeći način:

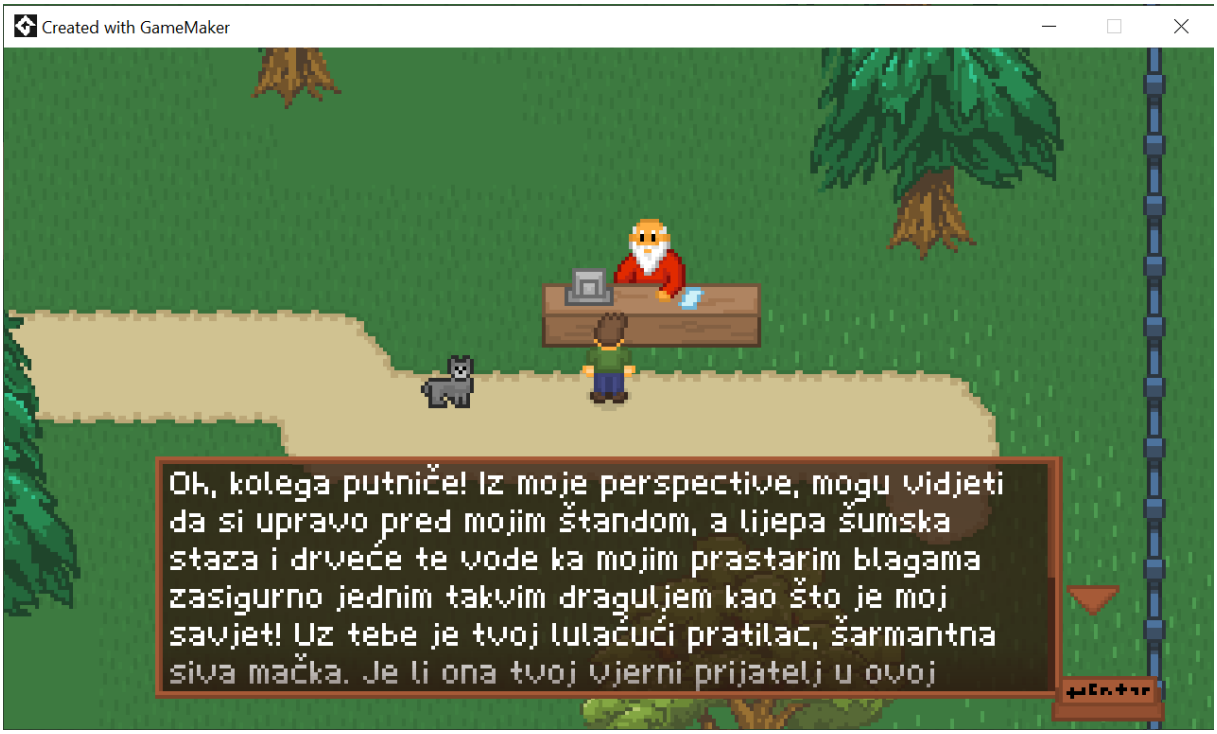
```
    var _image_filename = "screenshot.png";
    surface_save(application_surface, _image_filename);
    var _saved_image = _image_filename;
```

Varijabla `application_surface` je ugrađena globalna varijabla te predstavlja površinu na koju se cijela videoigra „crta“. [39] To je efektivno slika zaslona bez *GUI elemenata.*

Slanje zahtjeva sa slikom kao mogućnost razgovora sa trgovcem implementira se na sljedeći način:

```
    ["vision", cutscene_create_textbox_response_vision, ["Korisnik te pitao
što vidiš. Priložena slika je slika zaslona igrice, iz perspektive igrača.
Igrač je lik u maslinastoj majici, a ti si lik u crvenoj majici. Glumi da
vidiš svijet igrice iz perspektive očiju crvenog lika.", _color,
_my_gpt_service, 1.3]],
```

Sljedeća slika prikazuje odgovor na taj zahtjev kroz ulogu trgovca. Treba imati na umu da je ova slika videoigre uzeta nakon što su u videoigru dodani pozadinski objekti, kako bi trgovac imao što „vidjeti“, a i par objekata su dodani kako bi odgovor bio zanimljiviji:



Slika 27. Trgovac „vidi“ koristeći gpt-4o [autorski rad, slika zaslona]

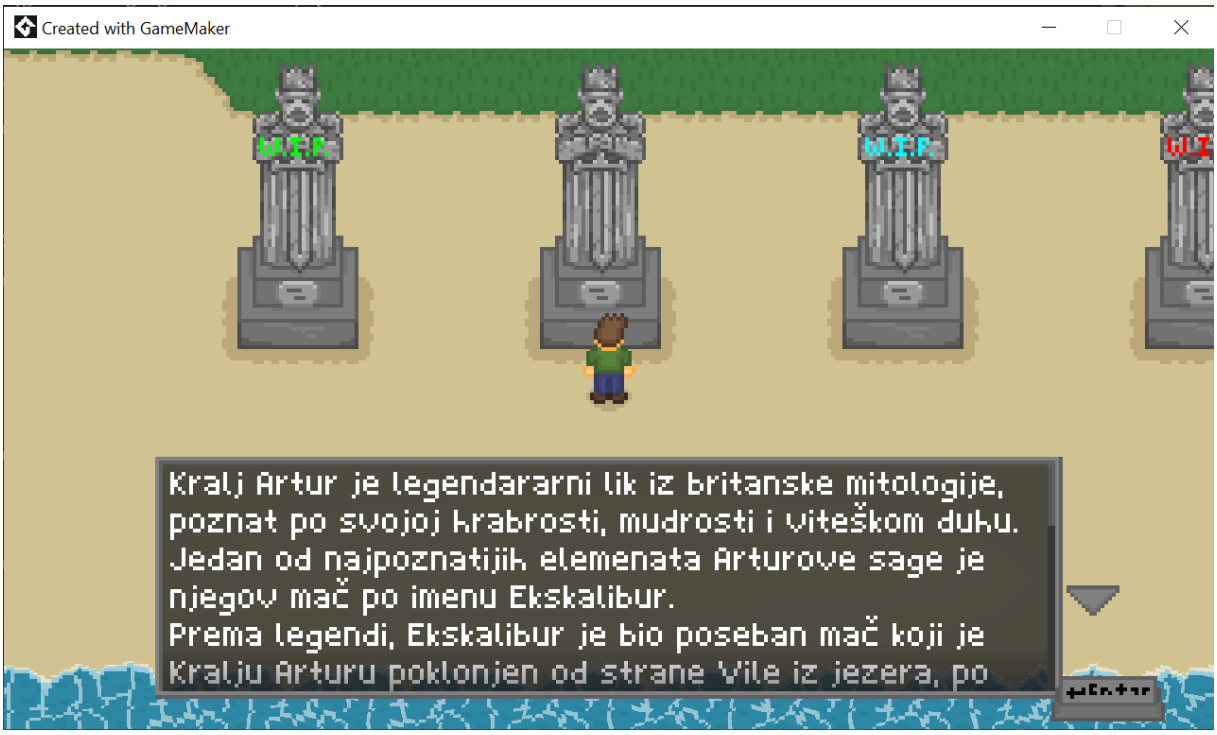
5.4.3.3. Jednokratni odgovor

Kako je u videoigri potrebno pogoditi lozinku, bilo bi dobro da igrač može pogoditi točnu samo igrajući videoigru. Zbog te potrebe, u videoigru su dodana četiri kipa kraljeva. Kako bi se izbjegao problem spomenut u poglavlju 5.2.2.5, gdje se šalje više *API* zahtjeva u isto vrijeme, kipovi će generirati odgovor samo pri prvoj interakciji te ga zapamtiti i kod svake iduće interakcije ispisati taj odgovor. Programski kod za svaki kip stavlja se u *Creation Code* događaj tog kipa:

```
    var _my_gpt_service = instance_create_layer(x, y, layer,
obj_gpt_service);
    _my_gpt_service.set_conversation_remember(false);
    var _prompt_text = "Napiši tekst o kralju Arturu i njegovom maču,
Ekskaliburu.";
    saved_text = "";
    function has_saved_text(_id) {
        if (_id.saved_text != "") {
            cutscene_jump("savedText");
```

```
        } else {
            cutscene_jump("firstTimeTalking");
        }
    }
    function set_saved_text(_gpt_service, _id) {
        _id.saved_text = _gpt_service.last_response;
        cutscene_end();
    }
    function cutscene_show_saved_text(_id) {
        cutscene_create_textbox(_id.saved_text, c_gray);
    }
    talkable_set_scene([
        [has_saved_text, [id]],
        ["firstTimeTalking", cutscene_create_textbox_response,
[_prompt_text, c_gray, _my_gpt_service, 1.2, "system"]],
        [set_saved_text, [_my_gpt_service, id]],
        [cutscene_jump, ["end"]],
        ["savedText", cutscene_show_saved_text, [id]],
        ["end", cutscene_end, []]
    ]);
```

Sada je moguće vidjeti generirane informacije o kipovima, s tim da se generiraju samo prvi put:



Slika 28. Kip kralja Artura sa generiranim tekstom [autorski rad, slika zaslona]

5.5. Dovršavanje videoigre

Svi glavni mehanizmi videoigre su dovršeni. Posljednji korak je popuniti sobu koristeći *tile*-ove i dodati razne objekte za uljepšavanje videoigre. Kreirani su razni objekti i ostali artefakti poput drveća, još kipova i plaže:

Slika 29. Videoigra sa drvećem, ogradom i stazom [autorski rad, slika zaslona]

Dorađeni su razgovori te su dodane nove opcije, poput traženja za pomoć oko lozinke kod razgovora s trgovcem. Koristeći globalne varijable, razgovori se mogu mijenjati, ovisno o stanju videoigre, tako videoigra može prepoznati je li igrač već pričao s vitezom, je li vidio dvorac, je li već pričao sa starcem i je li pogodio lozinku. Ove promjene omogućuju dinamičniji i zabavniji način razgovaranja s *NPC*-ima, kao i bolje iskustvo igranja.

Slika 30. Sve moguće opcije razgovora s trgovcem nakon što je igrač vidio dvorac i viteza, no još nije pričao s vitezom [autorski rad, slika zaslona]

Ovim korakom cijela je videoigra izrađena. Igrač može istraživati sobu, razgovarati sa vitezom i trgovcem te čitati informacije o kipovima, kako bi došao do točne lozinke. Trgovac mu može pomagati oko lozinke sa ograničenim znanjem o njoj te dati dodatan kontekst cijelom svijetu videoigre.

6. Zaključak

Ovaj je završni rad pokazao da se izborom odgovarajućih alata za izradu videoigre, kao i korištenjem odgovarajućeg *API*-ja ili druge tehnologije za umjetnu inteligenciju može stvoriti videoigra koja uspješno koristi tu tehnologiju za generiranje dinamičnih dijaloga s likovima, uranjajući igrača u njezin svijet i pruživši gotovo neograničene mogućnosti razgovora s likovima, kao i samog igranja.

Završni rad prošao je kroz opis tehnologija, opis drugih videoigara koje koriste sličnu tehnologiju i umjetnu inteligenciju na sličan način kao i videoigra koja se razvija te na kraju prošao kroz cijeli proces razvoja takve videoigre u odabranim alatima, dok je samo povezivanje tih dviju tehnologija detaljno opisano, tako da svatko tko pročita ovaj završni rad može pratiti sve korake. Kroz rad se spominje i važnost provjere cijene za *API* ako se koristi *API* trećih strana, kao i ispravno korištenje privatnog *API* ključa, a i spomenuti su neki načini da se kontrolira korištenje *API*-ja tako da igrači ne mogu oštetiti kreatora videoigre.

Pošto se danas ove tehnologije, a pogotovo umjetna inteligencija, izuzetno brzo razvijaju, dok su i same videoigre jako popularne, smatram da će daljnji napredak u tim područjima rezultirati sve većim brojem takvih videoigara. U trenutku pisanja ovog završnog rada, umjetna inteligencija u videoigrama još uvijek je novitet, no vjerujem da će u budućnosti korištenje umjetne inteligencije na ovakav način postati uobičajena praksa u većini videoigara. Svakako, treba biti pažljiv i pomno pratiti napredak umjetne inteligencije, kako bi se izbjegli neželjeni ishodi, a zadržao samo onaj dio te tehnologije koji doprinosi ljudima, bilo to u razvoju i realizaciji ideja, ili u ostalim područjima.

Popis literature

[1] "What is an API (Application Programming Interface)?", Amazon Web Services. [Na internetu]. Dostupno: https://aws.amazon.com/what-is/api/. [pristupano 26.06.2024.].

[2] "OpenAI API" (11.06.2020.), OpenAI. [Blog post]. Dostupno: https://web.archive.org/web/20200611150951/https://openai.com/blog/openai-api/. [pristupano 26.06.2024.].

[3] "Key concepts", OpenAI. [Na internetu]. Dostupno: https://platform.openai.com/docs/overview. [pristupano 26.06.2024.].

[4] "Models", OpenAI. [Na internetu]. Dostupno: https://platform.openai.com/docs/models. [pristupano 26.06.2024.].

[5] “API Reference Documentation”, OpenAI. [Na internetu]. Dostupno: https://platform.openai.com/docs/api-reference. [pristupano 26.06.2024.].

[6] "Usage", OpenAI. [Na internetu]. Dostupno: https://platform.openai.com/usage. [pristupano 27.06.2024.].

[7] "Best Practices for API Key Safety", OpenAI. [Na internetu]. Dostupno: https://help.openai.com/en/articles/5112595-best-practices-for-api-key-safety. [pristupano 27.06.2024.].

[8] "Managing Tokens", OpenAI. [Na internetu]. Dostupno: https://platform.openai.com/docs/guides/text-generation/managing-tokens. [pristupano 27.06.2024.].

[9] "Pricing", OpenAI. [Na internetu]. Dostupno: https://openai.com/api/pricing/. [pristupano 27.06.2024.].

[10] "Vision", OpenAI. [Na internetu]. Dostupno: https://platform.openai.com/docs/guides/vision. [pristupano 27.06.2024.].

[11] "Introduction to GameMaker", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/monthly/en/#t=Introduction%2FIntroduction_To_GameMaker_Studio_2.htm. [pristupano 30.06.2024.].

[12] "GML Visual", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/monthly/en/Drag_And_Drop/Drag_And_Drop_Index.htm. [pristupano 30.06.2024.].

[13] "GML Code", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/monthly/en/GameMaker_Language/GameMaker_Language_Index.htm. [pristupano 30.06.2024.].

[14] "The Asset Browser", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/monthly/en/index.htm#t=Introduction%2FThe_Asset_Browser.htm. [pristupano 30.06.2024.].

[15] "The Object Editor – IDE Basics" (01.01.2021.), Mark Alexander. [Blog post]. Dostupno: https://gamemaker.io/en/tutorials/object-editor. [pristupano 30.06.2024.].

[16] "The Async Events", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/monthly/en/The_Asset_Editors/Object_Properties/Async_Events.htm. [pristupano 30.06.2024.].

[17] "AI's PR Problem" (03.03.2017.), Jerry Kaplan. [Na internetu]. Dostupno: https://www.technologyreview.com/2017/03/03/153435/ais-pr-problem-2/. [pristupano 01.07.2024.].

[18] "What is generative AI?" (02.04.2024.), McKinsey & Company. [Na internetu]. Dostupno: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-generative-ai. [pristupano 01.07.2024.].

[19] "I made a game using OpenAI API" (2023.), Reddit. [Forum post]. Dostupno: https://www.reddit.com/r/OpenAI/comments/15uorrr/i_made_a_game_using_openai_api_desperate_for/. [pristupano 01.07.2024.].

[20] "Bargainer AI", Bargainer. [Na internetu]. Dostupno: https://www.bargainer.ai. [pristupano 01.07.2024.].

[21] "Best AI Games 2023" (31.08.2023.), Inworld Team. [Blog post]. Dostupno: https://inworld.ai/blog/best-ai-games-2023. [pristupano 01.07.2024.].

[22] "AI Dungeon", AI Dungeon. [Na internetu]. Dostupno: https://aidungeon.com/. [pristupano 01.07.2024.].

[23] "Skyrim Special Edition Mods", Nexus Mods. [Na internetu]. Dostupno: https://www.nexusmods.com/skyrimspecialedition/mods/98481. [pristupano 01.07.2024.].

[24] "Inworld: A Generative AI Platform for Creating NPCs" (02.08.2023.), Kyle Wiggers. [Blog post]. Dostupno: https://techcrunch.com/2023/08/02/inworld-a-generative-ai-platform-for-creating-npcs-lands-fresh-investment/?guccounter=1. [pristupano 01.07.2024.].

[25] Bloc, (28.04.2023.) “Future of Elder Scrolls & RPG - Skyrim NPCs & Inworld AI (like GPT-4 for gaming)”, *Youtube* [Video datoteka]. Dostupno: https://youtu.be/d6sVWEu9HWU. [pristupano 01.07.2024.].

[26] "Constants", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/beta/en/GameMaker_Language/GML_Overview/Variables/Constants.htm. [pristupano 02.07.2024.].

[27] "DS Maps", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/beta/en/GameMaker_Language/GML_Reference/Data_Structures/DS_Maps/DS_Maps.htm. [pristupano 02.07.2024.].

[28] "http_request", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/monthly/en/GameMaker_Language/GML_Reference/Asynchronous_Functions/HTTP/http_request.htm. [pristupano 02.07.2024.].

[29] "HTTP", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/monthly/en/The_Asset_Editors/Object_Properties/Async_Events/HTTP.htm. [pristupano 02.07.2024.].

[30] "What is Prompt Engineering?" (22.03.2024.), McKinsey & Company. [Blog post]. Dostupno: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-prompt-engineering. [pristupano 02.07.2024.].

[31] "The Async Events", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/monthly/en/The_Asset_Editors/Object_Properties/Async_Events.htm. [pristupano 02.07.2024.].

[32] " sprite_index", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/beta/en/GameMaker_Language/GML_Reference/Asset_Management/Sprites/Sprite_Instance_Variables/sprite_index.htm. [pristupano 06.07.2024.].

[33] " image_speed", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/beta/en/GameMaker_Language/GML_Reference/Asset_Management/Sprites/Sprite_Instance_Variables/image_speed.htm. [pristupano 06.07.2024.].

[34] "image_index", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/beta/en/GameMaker_Language/GML_Reference/Asset_Management/Sprites/Sprite_Instance_Variables/image_index.htm. [pristupano 06.07.2024.].

[35] "Collision Masks", YRDSB. [Na internetu]. Dostupno: http://www.yrdsb.ca/schools/suttondistrict.hs/DeptPrograms/business/Documents/Unit%203/5_Collision_Masks.pdf. [pristupano 06.07.2024.].

[36] "camera_create", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/monthly/en/GameMaker_Language/GML_Reference/Cameras_And_Display/Cameras_And_Viewports/camera_create.htm. [pristupano 06.07.2024.].

[37] FriendlyCosmonaut, (03.03.2018.) "Concepts & Setup | Cutscenes [1]", *YouTube* [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=2CLm38HCP64. [pristupano 07.07.2024.].

[38] "event_inherited", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/beta/en/GameMaker_Language/GML_Reference/Asset_Management/Objects/Object_Events/event_inherited.htm. [pristupano 07.07.2024.].

[39] "application_surface", GameMaker Manual. [Na internetu]. Dostupno: https://manual.gamemaker.io/beta/en/GameMaker_Language/GML_Reference/Drawing/Surfaces/application_surface.htm. [pristupano 10.07.2024.].

Popis slika