



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Алекса Симић

Фантазијске игре базиране на великим језичким моделима

ЗАВРШНИ РАД

- Мастер академске студије –

Нови Сад, 2024.

Садржај:

1. Увод	4
2. Кратак преглед историје интерактивне фикције	6
2.1 Период између 1960-те и 1980-те	6
2.1.1 Обрада природног језика	6
2.1.2 Авантура	6
2.2 Комерцијална ера	7
2.3 Период после 1990-те	7
3. Постојећа решења	8
3.1 <i>TADS</i>	8
3.2 <i>Inform</i>	8
4. Коришћени алати	10
4.1 <i>textX</i>	10
4.1.1 <i>Rules</i>	10
4.1.2 <i>textX</i> уgraђени типови/правила	11
4.1.3 <i>Rule expressions</i>	12
4.2 <i>Tkinter</i>	16
4.3 <i>OpenAI API</i>	18
4.3.1 <i>GPT</i> модел	18
4.3.2 <i>DALL-E</i> модел	20
5. Мета-модел интерактивне фикције	23
5.1 <i>gameDSL.tx</i>	25
5.1.1 Свет игрице	25
5.1.2 Регија (<i>Region</i>)	26
5.1.3 Објекат (<i>Item</i>)	28
5.1.4 Играч (<i>Player</i>)	29
5.1.4 Непријатељ (<i>Enemy</i>)	30
5.1.6 Оружја и оклоп	32
5.1.7 <i>GeneralSettings</i>	35
5.1.8 Кључне речи, идентификатори и коментари	36

5.2 Визуализација мета-модела.....	37
6. Интерпретер	39
6.1 <i>Python</i> класе	39
6.2 <i>interpreter.py</i>	48
7. <i>GPT</i> генерирање.....	53
7.1. Генерирање коначних игара интерактивне функције	53
7.2. Мод бесконачног играња	54
8. Графички интерфејс	56
8.1 <i>gui.py</i>	56
8.2 <i>codeEditorFrame.py</i>	59
8.3 <i>pictureCreatorFrame.py</i>	60
8.4 <i>gptCreationFrame.py</i>	63
8.5 <i>gameFrame.py</i>	64
9. Закључак	69
Литература	70

1.Увод

Интерактивна фикција, најчешће скраћено као ИФ (енг. *Interactive fiction, IF*), представља софтвер који симулира простор односно регије у којима играч користи команде текстуалног облика како би управљао ликовима, кретао се кроз регије, интераговао са стварима које се у тој регији налазе, улазио и бежао из борби против непријатеља и тако даље. Тако посматрано интерактивна фикција се може разумети као интерактивно приповедање. Такође може се разумети и као облик видео игре, било у облику авантуристичке игре (енг. *adventure game*), или игре улога (енг. *role-playing game, RPG*) [1].

Термин интерактивна фикција је некада означавао оне игре где је цео интерфејс био само текстуалан, међутим кроз развој технологија долази до прилагођавања термина који сад означава текстуалну авантуру где је главни начин интеракције са светом путем куцања текста у неком графичком корисничком интерфејсу (енг. *graphical user interface, GUI*).

Овај рад представља језик и алат за креирање игрица типа интерактивне фикције, који је реализован у облику апликације са графичким корисничким интерфејсом. У раду се такође истражује употреба великих језичких модела као подршке или као средства за финално креирање игара интерактивне фикције. Поред креирања игара, велики језички модели се користе и за генерирање бесконачних игара, као и за генерирање слика које доприносе дубљем зарањању у свет интерактивне фикције.

Језик за креирање игрица типа интерактивне фикције представља врсту језика специфичног за одређени домен (енг. *Domain Specific Language, DSL*). Први корак у креирању језика специфичног за домен је анализа жељеног домена. Неопходно је разумети све концепте, односе међу њима, као и њихова могућа ограничења. Након тога, потребно је дефинисати граматику језика коришћењем мета-језика. У овом раду је за ту сврху коришћен мета-језик *textX* [2]. Помоћу ново-дефинисане граматике можемо креирати текстове који прате ту граматику, чиме добијамо нови језик који се користи за описивање регија, интеракција са предметима, команди за кретање јунака и других елемената омогућених ново-дефинисаном граматиком.

Алат за креирање игрица типа интерактивне фикције садржи парсер, интерпретер, *dsl* класе, графички кориснички интерфејс и генератор игара интерактивне фикције. Пређашње наведене ставке алата су написане програмским језиком *Python*.

За мапирање света интерактивне фикције у граф *Python* објеката, односно у граф инстанци *dsl* класа, је задужен парсер. Док је за извршавање описаних интеракција и команди задужен интерпретер који пролази кроз ново-креирани граф.

Графички кориснички интерфејс за креирање игара интерактивне функције представља скуп прозора који омогућавају следеће:

- креирање нове игре интерактивне функције која поштује правила одређена ново-дефинисаном граматиком
- креирање нове игре интеарктивне функције коришћењем великих језичких модела
- преглед свих креираних игара
- играње креираних игара
- додавање слика за регије игара интерактивне функције
- играње креираних игара у моду са slikama
- играње креираних игара у бесконачном моду

Израда графичког корисничког интерфејса је остварена преко стандардне *GUI* библиотеке за *Python*, *Tkinter*.

Креирање игара интерактивне функције помоћу великих језичких модела реализовано је кроз коришћење *OpenAI API*-ја, тачније *gpt* модела [3]. Приликом креирања игара, неопходно је обезбедити промпт, односно опис саме игре, њеног тока, дешавања, приче и других релевантних елемената, као и назив игре која треба да се генерише.

Креирање слика за регије игара интерактивне функције је реализовано поново помоћу *OpenAI API*-ја, тачније *dall-e* модела, који представља модел који претвара текст у слику (енг. *text-to-image model*) [4].

Играње игре у бесконачном моду такође је реализовано помоћу *OpenAI API*-ја, тачније *gpt* модела. Док је комбинација бесконачног мода и мода играња са slikama реализована комбинацијом *OpenAI* модела *gpt* и *dall-e*.

Овај рад почињемо са освртом на историју интерактивне функције пропраћену неким од већ постојећих алата и језика за развој интерактивне функције. Након осврта на историју детаљније ће бити појашњене технологије које су коришћене при развоју апликације овог рада. Неке од технологија су *textX*, *Tkinter*, *OpenAI API*. У наставку долазе на ред функционалности алата, као и детаљна анализи његове структуре и имплементације. На крају се налази се коришћења алата као и прича о могућим побољшањима за будућност.

2. Кратак преглед историје интерактивне фикције

Иако су системи за генерисање поезије, четботови и видео игре настали пре интерактивне фикције, ова форма има дугу и занимљиву историју. Интерактивна фикција појавила се пре него што су персонални рачунари постали доступни широј јавности, у време када су само академици и истраживачи имали приступ рачунарима. Одиграла је значајну улогу у раним данима рекреативног рачунарства, преласком на микрокомпјутере, где је постала важан део тржишта забавног софтвера. Комерцијални успех донео је интерактивну фикцију милионима корисника и оставио дубок траг на индустрију компјутерских игара. Иако данас више није на врху продајних листа, повратак својим хоби коренима и бесплатна дистрибуција путем интернета омогућили су простор за нове, иновативне концепте који су раније били превише неконвенционални за масовно тржиште.[5]

У овом поглављу ћемо причати о историји интерактивне фикције. Подела историја може да се изврши по годинама. Имамо период између 1960-те и 1980-те, комерцијалну еру, период после 1990-те.

2.1 Период између 1960-те и 1980-те

2.1.1 Обрада природног језика

Као сам почетак узимамо програм *ELIZA*. Софтвер *ELIZA* није билп развијан у циљу да постане прва интерактивна фикција, али обрадом природног језика (енг. *Natural language processing*) зарад разумевања корисничких уноса се може формално сматрати најранијим примером интерактивне фикције. *ELIZA* представља вид симулације психотерапеута како у разумевању корисниковах уноса, тако и у одговарању на њих.

2.1.2 Авантура

У периоду око 1975. године програмер и познавалац пећина *William Crowther* је написао прву текстуалну авантуру *Adventure* [1]. Текстуална авантура се састојала од играча и наратора. Играч је истраживао пећину, док је наратор причао пуним реченицама природног језика. Наратор је разумео једноставне команде које су се састојале од скупа двеју простих енглеских речи.

Касније 1977. године студент Станфорд универзитета *Don Woods* долази до извornог кода текстуалне авантуре *Adventure*. *Woods* накнадно проширује текстуалну авантуру *Adventure* и ова верзија се сматра последњом канонском верзијом.

Текстуална авантура *Adventure* је доживела велики успех тих година и инспирисала је многе да се окушају у креирању сопствене текстуалне авантуре.

Најистакнутији људи инспирисани првом интерактивном фикцијом су *Ken Williams* и *Roberta Williams* који су основали *Sierra Online* и који су одлучили да овај жанр игрица прошире са графиком.

2.2 Комерцијална ера

Комерцијална ера је испуњена људском потребом да створи финансијски капитал на нечemu што је ново и у тренду. Велике компаније након што су увиделе како људи стварају своје сопствене авантуре и количину пратилаца које су те авантуре успеле да скапе, почеле су да креирају и објављују класична дела као интерактивну фикцију [6]. Нека од најзначајнијих дела која вреди споменути јесу *Wonderland* заснован на дечијој књизи "Алиса у земљи чуда" и *The Hobbit* који је заснован на књизи са истим називом.

2.3 Период после 1990-те

Са временом популарност текстуалних авантура је почела да опада. Постепено су престајале да буду вид забаве на рачунарима, а полако су постала неки вид уметности. У овом периоду долази до формирања групе под називом *IF Interactive Fiction Community*. Ново-настала група људи је сматрала да је интерактивна фикција нешто више од приповедања. Интерактивна фикција је представљала скуп приповедања, игрица и уметности [6].

У касним 1990-тим долази до експлозије креирања интерактивне фикције. Интерактивне фикције које су вредне поменути су *Phatopia*(1998), *All Roads*(2001) и *Slouching Towards Bedlam* (2003).

Након много година креирања и развијања игара интерактивне фикције 2006. године *Game Developer's Conference* прихвата и признаје игре жанра интерактивне фикције. *William Crowther* и *Don Woods* који су заслужни за прву текстуалну авантуру примају награду за њихов рад и доприносе везане за поље интерактивне фикције.

3. Постојећа решења

Историја развоја интерактивне фикције је богата. Баш због те богате историје креирања интерактивне фикције, као и њене популарности постоји велики број алата односно језика за њен развој. Данас постоје, а и активни су језици за развој интерактивне фикције који датирају још од самог почетака, а постоје и језици који су релативно млади. У свим релативно младим језицима се могу пронаћи инспирације из неких старијих решења. У овом поглављу ћемо проћи и детаљно описати неке од значајнијих решења проблема креирања интерактивне фикције.

3.1 TADS

TADS представља развојно окружење за текстуалне авантуре (енг. *Text Adventure Development System*). Настао 1987. године, а његов творац био је *Michael J. Roberts*. Пре настанка алата као што су *Inform*(1993) и *Hugo*(1995) *TADS* се сматрао једним од најозбиљнијих алата за креирање интерактивне фикције [7].

TADS 2 настаје убрзо након креирања *TADS*-а, 1988. Године. *TADS* 2 синтакса је заснована на језицима *C* и *Pascal* [7]. Један од најпопуларнијих *TADS* заправо и јесте *TADS* 2 алат, што можемо да видимо његовим коришћењем од стране аутора и дан данас.

Следећа верзија *TADS* алата, *TADS* 3, настала је 2006. године великим променама односно комплетним редизајном *TADS engine*-а [8]. *TADS* 3 представља тренутно најновију верзију *TADS* алата. Трећа верзија *TADS*-а помера границе развоја интерактивне фикције у следећим ставкама: потпуно преношење осећања, регије које је састоје од низа под-соба и интеракција *NPC* (*Non-Player Character*) ликова са светом [9].

3.2 Inform

Inform поред пређашње наведеног *TADS*-а представља један од најпопуларнијих алата за развијање интерактивне фикције. Настао је 1993. године, а његов творац је *Graham Nelson*. *Inform* је коришћен у хиљадама израда интерактивних фикција на осам језика [10]. Може да генерише програме дизајниране за *Z-machine* или *Glulx* виртуалне машине. У овом тренутку постоји седам верзија *Inform* алата. Верзије *Inform*-а од један до пет су објављене између 1993. године и 1996. године. У периоду око 1996. године долази до комплетног редизајнирања *Inform* алата и настанка верзије која се следећих година додатно развијала и побољшавала. *Inform* 6 садржи две главне компоненте: *the Inform compiler* и *the Inform library*. *Inform compiler* задужен је за генерисање фајлова за причу из *Inform source* кода. *Inform library* задужен је да решава већину тешких послова око парсирања играчевих команда и око праћења модела света [11].

Graham Nelson 2006. године објављује седму и тренутно последњу верзију *Inform*-а. *Inform* 7, познат и под називом *Natural Inform*, представља језик заснован на природном

језику и на новим сетовима правила [11]. *Inform 7* може да се подели на три главна дела: интегрисано окружење за развој (енг. *The Inform 7 IDE*), *Inform 7 compiler* за нови језик и стандардна правила која формирају библиотеку за *Inform 7*. *Inform 7* се ослања на *Inform library* и *Inform compiler* из *Inform 6*. Компајлер *Inform* седмице врши компајлирање *Inform 7* изворни кода у *Inform 6* изворни код. Затим добијени *Inform 6* изворни код компајлира *Inform* шест компајлер и генерише *Glulx* или *Z-code* фајл.

4. Коришћени алати

У овом поглављу ћемо се осврнути на алате који су били неопходни за решавање проблема у овом мастер раду. Разматраћемо алате као што су *textX*, *Tkinter* и *OpenAi api*, прецизније *gpt* и *dall-e* модели .

4.1 *textX*

Мета-језик који је коришћен за дефинисање основног језика овог мастер рада је *textX*. *textX* представља језик за дефинисање језика. Инспирисан је *Xtext*-ом [12]. *Xtext* представља мета-језик који је базиран на *Java* програмском језику. *textX* се користи за креирање језика специфичних за домен (енг. *domain-specific language, DSL*) у програмском језику *Python*. Мета-језик *textX* значајно помаже при изградњи новог језика специфичног за домен на једноставан начин или омогућава изградњу додатне подршке или се може искористити при проширењу за неки већ постојећи језик. Мета-модел је аутоматски изграђује у *textX*-у, у форми *Python* класа. Поред мета-модела изграђује се и парсер за ново-креирани језик. Парсер ће парсирати изразе новог језика и аутоматски ће изградити граф од *Python* објеката, односно модел који одговара мета-моделу [2].

Остatak поглавља посвећен је изразима *textX* граматике који су били неопходни за решавање представљеног проблема.

4.1.1 Rules

Правила (енг. *Rules*) су основни градивни блокови *textX*-а. Свако правило је записано у формату који почиње називом правила, након чега следи двотачка. Иза двотачке налази се тело правила, а цела форма завршава се тачка- зарезом. (Слика 4.1.1.a).

```
Hello:  
    'hello' who=ID  
;
```

Слика 4.1.1.a Пример *textX* правила (преузето са <http://textx.github.io/textX/3.1/grammar/>)

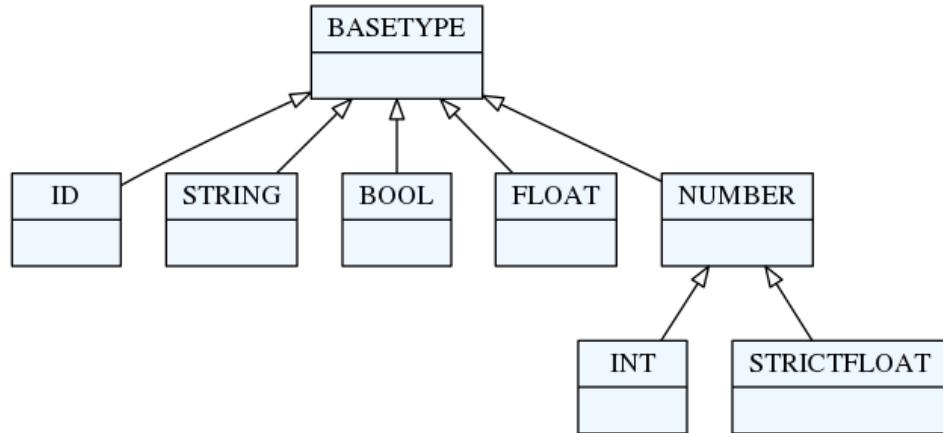
На слици изнад се види правило под називом “Hello”. Из правила следи да објект *Hello* чини *string literal* ‘hello’ праћен *ID*-ем правила. Правило *Hello* истовремено дефинише и *Python* класу *Hello*. Када се ово правило препозна у улазном току, формира се објекат ове класе са атрибутом *who*, који ће бити постављен на вредност која испуњава услове *ID* правила након речи ‘hello’. (Слика 4.1.1.b).

```
hello Alice  
hello Bob  
hello foo1234
```

Слика 4.1.1.b Пример валидних Hello правила (преузето са <http://textx.github.io/textX/3.1/grammar/>)

4.1.2 textX уgraђени типови/правила

У претходном примеру смо видели *ID* правило. Оно је једно од уgraђених правила која формирају базу textX система типова (Слика 4.1.2.a).



Слика 4.1.2.a уgraђени типови у textX-у (преузето са <http://textx.github.io/textX/3.1/grammar/>)

textX има следећа уgraђена правила:

- *ID* правило: представља општи идентификатор који се састоји од слова, цифара и доњих црта. Regex шаблон за опис овог правила је `'[^\\d\\W]\\w*\\b'`. Овакво подударање се конвертује у Python string.
- *INT* правило: проналази integer број и конвертује га у Python int.
- *FLOAT* правило: проналази floating point број и конвертује га у Python float инстанцу. Ово правило може да пронађе и integer бројеве. Због тога се уводи једно правило за floating point бројеве под називом *STRICTFLOAT*.
- *STRICTFLOAT* правило: проналази искључиво floating point бројеве који садрже '.' или 'e|E' и конвертује их у Python float инстанце.
- *BOOL* правило: проналази кључне речи *true* и *false* и конвертује их у Python bool инстанцу.
- *STRING* правило: проналази низ речи које се налазе између наводника и конвертује га у Python str инстанцу.

Уgraђени типови textX-а ће увек приликом парсирања бити конвертовани у неки од предодређених Python типова.

4.1.3 Rule expressions

Изрази правила (енг. *Rule expressions*) заправо јесу суштина правила, односно тело правила. Састоје се од основних израза и оператора.

4.1.3.1 Matches

Match изрази представљају основне градивне блокове за комплексније изразе. *Match* изрази су изрази најнижег нивоа. Након успешног преклапања *Match* изрази конзумирају унос.

Match изрази постоје у два типа:

1. *String match* – је дефинисан као низ карактера који се налази између наводника (Слика 4.1.3.1.а).

```
'blue'  
'zero'  
'person'
```

Слика 4.1.3.1.а Пример *String match*-а (преузето са <http://textx.github.io/textX/3.1/grammar/>)

2. *Regex match* – користи регуларни израз (енг. *regular expression*) који је дефинисан између две косе црте ' / '. За разлику од *String match*-а, који може пронаћи само у потпуности идентичан унос, *Regex match* проналази цео скуп уноса који одговарају датом регуларном изразу (Слика 4.1.3.1.б).

```
/\d*/  
/\d{3,4}-\d{3}/  
/[ -\w]*\b/  
/[^}*]*
```

Слика 4.1.3.1.б Пример *Regex match*-а (преузето са <http://textx.github.io/textX/3.1/grammar/>)

4.1.3.2 Sequence

Секвенца (енг. *Sequence*) представља *textX* израз који се добија писањем подизраза један за другим (Слика 4.1.3.2.а).

```
Colors:  
    "red" "green" "blue"  
;
```

Слика 4.1.3.2.a Пример секвенце (преузето са <http://textx.github.io/textX/3.1/grammar/>)

На Слика 4.1.3.2.a правилу дефинисали смо секвенцу која се састоји од три *string match* израза (*red*, *green*, *blue*). Секвенца ће бити препозната ако и само ако се сви *string match* изрази буду тачно редоследом наведени. У супротном, секвенца неће бити пронађена.

Једини *string* израз, односно унос, који ће бити препознат у скалду са пређашње наваденим правилом секвенце је: *red green blue*

4.1.3.3 Ordered choice

Уређени избор (енг. *Ordered choice*) представља сет могућих израза који су раздвојени усправном цртом ' | '. Правило уређеног избора ће покушати да пронађе унос који се поклапа са изразима, крећући се са леве на десну страну све док не дође до успешног поклапања (Слика 4.1.3.3.a).

```
Color:  
    "red" | "green" | "blue"  
;
```

Слика 4.1.3.3.a Пример уређеног избора (преузето са <http://textx.github.io/textX/3.1/grammar/>)

На Слика 4.1.3.3.a приказано правило уређеног избора ће покушати да пронађе прво израз *red* па *green* па *blue* у том наведеном редоследу.

4.1.3.4 Optional

Опциони изрази (енг. *Optional*) су изрази који ће покушати да пронађу унос идентичан њима. Ако идентичан израз не буде пронађен неће доћи до формирања грешке, јер су опциони изрази увек успешни (Слика 4.1.3.4.a).

```
MoveUp:  
    'up' INT?  
;
```

Слика 4.1.3.4.a Пример опционог израза (преузето са <http://textx.github.io/textX/3.1/grammar/>)

На Слика 4.1.3.4.a примеру се види да је *INT match* опциони израз. Читањем правила уочавамо да је кључна реч (енг. *keyword*) *up* обавезна, а пратећи *integer* број није. Следећи уноси ће бити пронађени поштујући горе наведено правило: *up 45, up 1, up*

Опциони изрази могу бити доста комплекснији. На слици испод приказан је уређени избор између облих заграда који је опциони (Слика 4.1.3.4.б).

```
MoveUp:  
  'up' ( INT | FLOAT )?
```

Слика 4.1.3.4.б Пример комплексног опционог израза (преузето са <http://textx.github.io/textX/3.1/grammar/>)

4.1.3.5 Assignments

Доделе (енг. *Assignments*) су део процеса дедукције мета-модела. Свака ствара атрибут мета-класе креираном правилом. Додела се састоји од леве стране (енг. *left-hand side*, *LHS*) и од десне стране (енг. *right-hand side*, *RHS*). Лева страна означава назив атрибути мета-класе. Десна страна представља или референцу на друга правила или представља неки једноставан *match* израз (Слика 4.1.3.5.а).

```
Person:  
  name=Name ',' surname=Surname ',' age=INT ',' height=INT ';'
```

Слика 4.1.3.5.а Пример доделе (преузето са <http://textx.github.io/textX/3.1/grammar/>)

На Слика 4.1.3.5.а приказани пример представља правило и мета-класу *Person*. Правило *Person* ће испарсирати и инстанцирати објекте *Person* који садрже следећа четири атрибути:

1. *name* – користи правило *Name* да пронађе унос. Вредност ће представљати референцу на инстанцу *Name* класе.
2. *surname* – користи правило *Surname* да пронађе унос.
3. *age* – користи већ уграђени основни тип *INT* да пронађе бројеве у уносу. Вредност коју добијемо за *age* ће бити конвертована у *python int* тип.
4. *height* – користи исту логику као и *age* у тачки изнад. Разлика је што ће вредност бити додељена *height* атрибуту *Person* инстанце.

Запета са наводницима у *Person* правилу представља синтаксичку буку којом се обезбеђују специфичнија подударања.

Следећи уноси задовољавају критеријуме *Person* правила:

Jovan, Jovic, 21, 193;
Johnny, Smoe, 44, 168;

Постоје четири типа доделе:

1. Обична додела (*Plain assignment*) подразумева употребу оператора '='. Овај тип доделе приписује вредност пронађену правилима са десне стране левој страни
2. Булова додела (*Boolean assignment*) подразумева коришћење оператора '?='. Овај тип доделе приписује вредност True левој страни ако су правила на десној страни задовољена. У супротном, левој страни се додељује вредност False.
3. Додела нула или више (*Zero or more assignment*) користи оператор '*='. Лева страна постаје листа вредности које десна страна пронађе. Ако десна страна не пронађе ниједан унос који задовољава правило, лева страна ће бити празна листа.
4. Додела једног или више (*One or more assignment*) користи оператор '+='. Овај тип доделе је сличан претходном, с тим што је овде неопходно да постоји бар једно подударање уноса са правилом. У претходном типу доделе, подударање није обавезно, док се овде захтева да буде пронађено најмање једно.

4.1.3.6 References

Правила имају могућност да референцирају једно друго. Најчешће појављивање референци јесте баш на десној страни доделе. Постоје две врсте референцирања:

1. Референца правила подударања (Match rule reference) представља позивање другог правила. Када се креира инстанца тог позваног правила, она се додељује атрибуту са леве стране доделе, формирајући однос родитељ-дете (Слика 4.1.3.6.а).

```
Structure:  
  'structure' '{'  
    elements*=StructureElement  
  }  
;
```

Слика 4.1.3.6.а Пример референце подударања (преузето са <http://textx.github.io/textX/3.1/grammar/>)

На Слика 4.1.3.6.а примеру *StructureElement* ће покушати да пронађе нула или више уноса који одговарају његовим критеријумима. За свако подударање долази до креирања инстанце *StructureElement* која се додаје у листу *elements*. Листа *elements* се налази на левој страни доделе. Родитељ сваког *StructureElement*-а из листе *elements* ће бити постављен на *Structure*.

1. Референца правила везивања (Link rule reference) записује се као референцирано правило унутар угластих заграда. Ова референца покушава да пронађе идентификатор већ постојећег објекта који испуњава њене критеријуме. Када пронађе одговарајући објекат, идентификатор се конвертује у Python референцу тог постојећег објекта. *textX* аутоматски извршава овај процес разрешавања референци (Слика 4.1.3.6.6).

```
ScreenType:  
    'screen' name=ID "{"  
    "}"  
    ;  
  
ScreenInstance:  
    'screen' type=[ScreenType]  
    ;
```

Слика 4.1.3.6.6 Пример референце везивања (преузето са <http://textx.github.io/textX/3.1/grammar/>)

textX подразумева да *name* атрибут користи сам идентификатор тог траженог већ постојећег објекта. Такође све инстанце објеката исте класе налазе се у једном простору имена (енг. *namespace*). На Слика 4.1.3.6.6 примеру *type* атрибут је веза до *ScreenType* објекта.

4.2 Tkinter

Python има много *GUI* оквира (енг. *frameworks*), али *Tkinter* је једини оквир који је директно укључен у стандардну Python библиотеку [13]. Назив *Tkinter* је скраћеница од *Tk Interface*-а, где *Tk* представља вишеплатформски алат намењен за креирање *GUI*-а. Неки од основних елемената *Tkinter GUI*-а су: прозор (енг. *Window*), прозор највишег нивоа (енг. *Top-level window*), *widget*, *frame* [14].

Прозор је основни елемент *Tkinter GUI*-а, који служи као контејнер за све остale *GUI* компоненте. У прозору се могу налазити елементи попут текстуалних поља (енг. *text boxes*), етикета (енг. *labels*), и дугмади (енг. *buttons*). *Tkinter* омогућава креирање апликацијских прозора и управљање њиховим својствима, као што су величина, наслов и икона.

Прозор највишег приоритета је прозор који функционише као подређени примарном прозору. Овај прозор је опремљен стандардним оквиром и контролама које

пружа *desktop* менаџер. Може се слободно премештати по радној површини, а често је омогућена и промена његове величине.

Вицети (енг. *Widgets*) су различити визуелни елементи који се користе за креирање корисничког интерфејса. Међу најчешћим визуелним елементима су прозори, дугмићи, текстуална поља, етикете и слично. У *Tkinter*-у, након креирања визуелног елемента, потребно је да се тај елемент постави на одређену позицију у прозору. У супротном, елемент неће бити видљив. Постоји неколико метода за позиционирање визуелних елемената, међу којима су:

- *Pack* – над вицетом позива се функција *pack()*. Користећи *pack()* методу, визуелни елементи се аутоматски пакују унутар прозора на одређени начин. Приликом паковања, ако су одређени елементи већ постављени, *Tkinter* аутоматски израчунава потребан простор и поставља нове елементе на одговарајућу позицију.
- *Grid* – над вицетом позива се функција *grid(row= , column=)*. Ова метода користи "решетку" или "мрежу" за одређивање позиције и распореда визуелних елемената унутар фрејма или прозора. Помоћу параметара *row* и *column*, можемо прецизно одредити у којем реду и колони ће се поставити одређени елемент.
- *Place* – над вицетом позива се функција *place(x= , y= , width= , height=)*. Ова метода омогућава прецизно управљање позицијом и величином визуелних елемената, што је посебно корисно за сложене распореде или када је потребно тачно одредити позицију на прозору.

Frame у *Tkinter*-у је основна јединица за организовање комплексних распореда (енг. *complex layouts*). То је правоугаона површина смештена унутар прозора, која може да садржи друге компоненте *GUI*-а.

У *Tkinter*-у је заступљено формирање веза дете-родитељ. Приликом креирања неке апликације ако поставимо лабелу унутар *frame*-а долази до формирања везе родитељ-дете. *Frame* постаје родитељ лабели, а она постаје његово дете.

Једна од битнијих метода у *Tkinter*-у је *mainloop()*. Ова метода покреће главни циклус *GUI* апликације, омогућавајући кориснику да интерагује са апликацијом и реагује на различите догађаје. Главни циклус је одговоран за континуирано ажурирање и приказивање *GUI*-ја апликације.

4.3 OpenAI API

У овом поглављу ћемо објаснити шта је то заправо OpenAI API, који су његови модели и чему служи. OpenAI API је интерфејс за програмско повезивање који омогућава приступ моћним моделима машинског учења развијеним од стране OpenAI компаније. Постоји више врста модела. Постоје текстуални модели, аудио модели, сликовни модели. Неки од OpenAI најпознатијих модела су: *GPT-4o*, *GPT-4o mini*, *GPT-3.5-turbo*, *DALL-E 2*, *DALL-E 3*, *Whisper*, *TTS HD*. Ови модели омогућавају креирање и интеграцију напредних функција као што су генерисање текста, обрада природног језика и стварање слика. API је дизајниран тако да програмерима омогући лако коришћење ове моделе у својим апликацијама [15]. За потребе овог мастер рада најинтересантнији модели су *GPT* и *DALL-E* модели. У наставку ти модели ће бити детаљније појашњени.

4.3.1 GPT модел

GPT (енг. *Generative pre-trained Transformer*, *GPT*) представља модел вештачке интелигенције који генерише садржај. Као такав модел он је припадник *AIGC* групе модела (енг. Artificial Intelligence Generated Content, *AIGC*) [16]. Својство припадника *AIGC* групе модела је да њихови корисници могу да креирају садржај односно слике, текст, видео записи аутоматски на основу њихових унетих захтева.

GPT је интелигентан робот за четовање који има способност да да детаљан одговор на све захтеве који су му прослеђени. Одлично се показао на различитим задацима разумевања језика као и на задацима генерисања. Неки од задатака су: вишејезично машинско превођење, детектовање и уклањање грешака у коду, прихватање односно признавање грешака. *GPT* модел се разликује од других модела по томе што је у стању да памти разговор са корисником, односно памти корисникove захтеве што значајно помаже у континуираном дијалогу.

Развој *GPT*, генеративног предтренираног трансформатора, модела је заснован на унапређивању кроз верзије. Од прве *GPT-1* верзије до најновије *GPT-4* верзије. Свака новија верзија *GPT* модела је донела значајна побољшања.

GPT-1 је настало 2018. године. Поставио је темеље за генерисање текста користећи трансформаторску архитектуру. Представља прву верзију модела који је трениран за генерисање језика користећи трансформаторску архитектуру путем ненадзоране обуке. Након почетне обуке, модел је додатно прилагођаван за специфичне задатке.[17]

GPT-2 је настало 2019. године. Увео је идеју мултитаск учења, односно учења на више задатака одједном. Има више параметара и података за обуку у односу на

GPT-1. Ово је омогућило да се модел примени на већину задатака који су претходно захтевали додатну обуку. [18]

GPT-3 је настао 2020. године. Даље је побољшао могућност модела да се прилагоди задатцима који захтевају мало или нимало обуке. У односу на *GPT*-2 увећан му је број параметара за 100 пута. То га чини првим моделом који је прешао број од 100 милијарди параметара. [19]

ChatGPT (*InstructGPT*) је пилот верзија заснована на моделу *GPT*-3.5, где су истраживачи користили појачање учења са повратним информацијама од људи како би модел боље разумео и следио намере корисника.

GPT-4 је најновија верзија *GPT* модела. Настала је 2023. године. *GPT*-4 верзија је мултимодална, што значи да може примати и сликовне и текстуалне улазе, а генерише текстуалне излазе. *ChatGPT* са *GPT*-4 постиже перформансе на нивоу људских способности на различитим професионалним и академским тестовима. [20]

	<i>GPT</i> -1	<i>GPT</i> -2	<i>GPT</i> -3	<i>GPT</i> -4
Датум објаве	јун 2018	фебруар 2019	мај 2020	март 2023
Параметри модела	117 милиона 12 слојева 768 димензија	1.5 милијарди 48 слојева 1600 димензија	175 милијарди 96 слојева 12 888 димензија	Необјављено
Величина контекста	512 токена	1024 токена	2048 токена	8195 токена
Величина података пре обуке	око 5 GB	40 GB	45 TB	Необјављено
Извор података	<i>BooksCorpus</i> , <i>Wikipedia</i>	<i>WebText</i>	<i>Common Crawl</i> ...	Необјављено
Циљ учења	учење без надзора	учење са више задатака	учење у контексту	Мултимодално учење

Тавела 4.3.1.a Поређење *GPT* верзија [16]

У Тавела 4.3.1.a се детаљније могу видети неке од разлика између верзија *GPT* модела.

За потребе овог мастер рада од *GPT* модела интересантним су се показали: *GPT-4o*, *GPT-4o mini*, *GPT-3.5 Turbo*.

GPT-3.5 Turbo модел има лимит од 200.000 TPM (токена по минути), 500 RPM (захтева по минути, енг. *requests per minute*) и 10,000 RPD (захтева по дану, енг. *requests per day*). Цена при коришћењу овог модела је \$0.50 по милиону улазних токена и \$1.50 по милиону излазних токена. Овај модел се у тестирањима показао као довољно способан за креирање игара типа интерактивне фикције поштујући правила задата *textX*-ом. Мана му је што су *GPT-4* модели напреднији и јефтинији.

GPT-4o mini модел има исте лимите као и претходно наведени *GPT-3.5 Turbo* модел. Цена при коришћењу овог модела је \$0.15 по милиону улазних токена и \$0.60 по милиону излазних токена. Овај модел се у тестирањима показао као сасвим способан за креирање игара типа интерактивне фикције поштујући правила задата *textX*-ом. Овај модел представља добар баланс цене и квалитета новијег модела.

GPT-4o модел има лимит од 30.000 TPM (токена по минути) и 500 RPM. Цена при коришћењу овог модела је \$2.50 по милиону улазних токена и \$10.00 по милиону излазних токена. Овај модел се у тестирањима показао као изузетно маштовит и способан за креирање игара типа интерактивне фикције поштујући правила задата *textX*-ом. Мана му је што је доста скупљи од *GPT-4o mini*, али јесте доста бољи.

Након тестирања за потребе овог мастер рада је одлучено да се користи *GPT-4o*, тачније *gpt-4o-2024-08-06* модел. Упркос вишеј цени, претпостављена количина потребних токена ће бити довољно мала, те цена по милиону токена неће представљати одлучујући фактор.

4.3.2 *DALL·E* модел

DALL·E је модел вештачке интелигенције који генерише слике на основу текстуалних описа. Као припадник групе AIGC (енг. Artificial Intelligence Generated Content, AIGC), *DALL·E* омогућава корисницима да креирају визуелне садржаје аутоматски, користећи текстуалне захтеве.[21]

DALL·E модел је револуционисао начин на који се генеришу слике, пружајући могућност стварања визуелних приказа који су раније захтевали људску креативност и уметничку вештину. Овај модел је показао изванредну способност да креира сложене, кохерентне и естетски задовољавајуће слике на основу детаљних текстуалних описа, што га чини корисним алатом у разним креативним и професионалним областима.[22]

Прва верзија *DALL-E* модела, објављена 2021. године, демонстрирала је способност креирања широког спектра слика, од реалистичних до фантастичних, само на основу унетог текстуалног упuta. Овај модел је користио трансформаторску архитектуру и био је трениран на великом скупу текстуалних описа и слика. Пример генерисане слике се може видети доле испод на Слика 4.3.2.a.



Слика 4.3.2.a Пример *DALL-E* 1 слике преузето са <https://thenextweb.com/news/heres-how-openais-magical-dall-e-generates-images-from-text-syndication>. Улаз при генерирању је „је фотеља у облику авокада“

DALL-E 2, представљен 2022. године, је значајно побољшао квалитет слика које модел може да генерише. Ова верзија је донела боље разумевање текстуалних описа, прецизнију контролу над стилом и садржајем слика, као и већу резолуцију финалних визуелних резултата. *DALL-E* 2 је уједно увео могућност измене постојећих слика (енг. *inpainting*), омогућавајући корисницима да прилагођавају делове слика на основу нових текстуалних захтева. Пример генерисане слике се може видети доле испод на Слика 4.3.2.b.



Слика 4.3.2.б Пример *DALL-E* 2 слике преузето са [23]. Улаз при генерирању је „Абрахам Линколн додирује своје ножне прсте док Џорџ Вашингтон ради згибове. Линколн је босоног. Вашингтон носи чизме“

DALL-E 3 је најновија верзија *DALL-E* модела, представљена 2023. године. Ова верзија је наставила да побољшава способност модела за генерисање слика на основу текстуалних описа, уводећи неколико значајних унапређења у односу на претходне верзије. Нека од тих унапређења су: прецизнији и креативнији визуелни прикази, боље разумевање контекста и сложених текстуалних захтева, интеграција са другим алатима за креирање садржаја и тако даље. Пример генерисане слике се може видети доле испод на Слика 4.3.2.в.



Слика 4.3.2.в Пример *DALL-E* 3 слике преузето са [24]. Улаз при генерисању је "A illustration from a graphic novel. A bustling city street under the shine of a full moon. The sidewalks bustling with pedestrians enjoying the nightlife. At the corner stall, a young woman with fiery red hair, dressed in a signature velvet cloak, is haggling with the grumpy old vendor. the grumpy vendor, a tall, sophisticated man is wearing a sharp suit, sports a noteworthy moustache is animatedly conversing on his steampunk telephone."

OpenAI API у понуди има *DALL-E* 2 и *DALL-E* 3 моделе. *DALL-E* 2 има резолуције слика 256×256, 512×512 и 1024×1024 са ценама \$0.016, \$0.018 и \$0.020 по слици. *DALL-E* 2 има резолуције слика 1024×1024, 1024×1792 и 1792×1024 са додатним атрибутом квалитета слика који може бити *Standard* или *HD*. Цена за *DALL-E* 3 је респективно \$0.040, \$0.080, \$0.080, \$0.120 у зависности од квалитета и резолуције слике.

Приликом тестирања *DALL-E* 2 модел се показао сасвим способним за креирање слика. *DALL-E* 2 модел јесте дупло јефтинији од *DALL-E* 3 модела, али пресудну улогу при избору модела чинило је потребно време при генерисању фотографија. *DALL-E* 2 због могућности генерисања слика формата 256×256 је имао огромну предност због чињенице да при бесконачном игрању у игри интерактивне функције при креирању нових и нових регија време потребно за генерисање изгледа тих регија било значајно брже од *DALL-E* 3 модела.

5. Мета-модел интерактивне фикције

Ово поглавље служи за детаљно објашњавање мета-модела и интерпретера за игре интерактивне фикције. Мета-модел описује структуру света и начин интеракције с њим, као и реализацију језика који користимо. Дефинише правила, акције, команде и све могуће догађаје унутар света игре. Овај мета-модел представља језик специфичан за домен, развијен за потребе овог мастер рада користећи раније поменути мета-језик *textX*. Интерпретер је програм који извршава код написан у одређеном програмском језику. У овом случају, ради се о *DSL-у* за интерактивну фикцију. За потребе овог мастер рада, интерпретер ће бити *Python* програм који ће извршавати операције над моделом.

Почетак креирања језика специфичног за домен заправо представља анализу тог домена. У овом случају вршимо анализу домена за игре интерактивне фикције да би боље схватили постојеће концепте, њихове односе, могућности као и ограничења која имају.

Захваљујући анализи домена, долазимо до следећих закључака: потребно је креирати свет у којем постоји један играч, играч мора бити у могућности да препозна регију у којој се налази, интерагује са објектима у тој регији, слободно се креће, напада непријатеље, бежи од њих, да се опреми са оружјем, да скине оружје, да има увид у своје статистичке поене, да има могућност да повећа одређене карактеристике и то представља неке основне ствари. Такође, неопходно је да регије буду повезане логичким односима који омогућавају играчу да се креће у складу са својим изборима. На основу извршене анализе овог домена створене су следеће команде:

- **MoveCommand** – Овом командом играч исказује жељу да се помери у одређеном правцу. Ако се у том правцу налази регија, односно ако су регије повезане, играчу ће бити дозвољено да се помери у том правцу. Уколико нема повезане регије у изабраном правцу, играч остаје на тренутној локацији.
- **OpenCommand** – Овом командом играч покушава да отвори објекат у регији у којој се налази. Ако се објекат заиста налази у регији и може бити отворен, он ће га успешно отворити, а сви предмети (*item-и*) из тог објекта биће премештени у регију у којој се играч налази. Успешним отварањем објекта, предмети из њега падају у регију, а сам објекат нестаје.
- **TakeCommand** – Овом командом играч покушава да узме објекат, односно *item* који се налази у његовој близини. Ако успешно узме објекат из регије, тај објекат ће се појавити у његовој торби и нестати из регије одакле је узет. Уколико објекат има својство *isStatic* постављено на истинито, играч неће моћи да га покупи.
- **DropCommand** – Овом командом играч исказује жељу да испусти објекат који се налази у његовој торби. Ако се изабрани објекат заиста налази у његовом поседу, команда ће бити успешно извршена, а објекат ће бити премештен из његове торбе у регију у којој се играч тренутно налази.

- ***UseCommand*** – Овом командом играч покушава да искористи објекат, односно *item* који је у његовом поседу. Ако се објекат заиста налази у његовом поседу, он ће бити искоришћен и уклоњен из тренутне сесије игре.
- ***EquipCommand*** – Овом командом играч покушава да опреми објекат, односно *item* који се налази у његовој торби. Ако је објекат у поседу играча и може бити опремљен, команда ће бити успешно извршена и тај објекат ће постати део његове опреме.
- ***UnequipCommand*** – Овом командом играч исказује жељу да скине опремљени објекат. Ако је изабрани објекат тренутно опремљен, команда ће бити успешно извршена, објекат ће бити уклоњен са играча и враћен у његову торбу.
- ***InfoCommand*** – Овом командом играч може да добије додатне информације о објектима који се налазе у његовој торби. Уколико објекат постоји команда ће бити извршена.
- ***InventoryCommand*** – Овом командом играч добија приказ свих објеката који се налазе у његовој торби.
- ***HealthCommand*** – Овом командом играч проверава своје тренутно здравље.
- ***AttackCommand*** – Овом командом играч покушава да нападне противника у околини. Ако се противник налази у истој регији као и играч, команда ће бити успешно извршена и нанети ће штету противнику на основу оружја или способности које играч користи.
- ***FleeCommand*** – Овом командом играч изражава жељу да побегне из борбе. Играч ће напустити тренутну регију и преместити се у регију из које је дошао, избегавајући сукоб.
- ***IncCommand*** – Овом командом играч може повећати ниво одређене вештине или способности. Ако играч има доволно поена или ресурса за унапређење, команда ће успешно повећати изабрану вештину.
- ***StatsCommand*** – Овом командом играч добија преглед свих својих тренутних статистика, укључујући ниво здравља, мане, вештина и других важних параметара који утичу на игру.
- ***ExitCommand*** – Овом командом играч може изаћи из тренутне сесије игре. Команда ће прекинути игру и вратити играча на почетни мени или у потпуности затворити игру, зависно од поставке.

Након што смо описали команде потребне за интеракцију играча са светом, неопходно је дефинисати и описати сам тај свет у којем се играч налази. Свет ће бити представљен као скуп повезаних регија. Свака регија састоји се од назива регије, описа њеног изгледа, објеката које садржи, веза које је повезују са другим регијама, временских непогода, као и одређених додатних услова који утичу на њен приступ.

Значи потребне су нам регије. Потребни су нам објекти и непријатељи у тим регијама. Потребан нам је играч који ће да интерагује са тим објектима. На основу извршене анализе овог домена створени су следећи ентитети интерактивног света:

- **Region** – Представља локацију у којој се играч може наћи. Поред играча, у регији се налазе и објекти и непријатељи са којима он може да интерагује. Свака регија је дефинисана својим именом, које служи као јединствени идентификатор, и детаљним описом који представља изглед и атмосферу те регије. Регија такође садржи објекте и непријатеље који су у њој присутни, везе према другим регијама које омогућавају кретање из једне у другу, и опционалне услове који морају бити испуњени да би играч могао да приступи тој регији.
- **Item** – Представља објекат који се налази у регији или у играчевој торби. Играч може да интерагује са предметима који су у његовој близини. *Item-и* могу бити смештени и унутар других *item-a*. Сваки *item* је дефинисан именом, које служи као јединствени идентификатор, и описом који представља његов изглед. Осим тога, *item* може садржати друге објекте унутар себе, имати функцију активације (описује шта се догађа када играч користи тај предмет), и својство статичности, које одређује да ли играч може да га узме или не.
- **Weapon/Armor** – Представља тип објекта који се користе за побољшање играчевих борбених способности. *Weapon* (оружје) је предмет који играч користи за напад на противнике, док *Armor* (оклоп) служи за заштиту од непријатељских напада. Ови објекти су дефинисани именом, које служи као јединствени идентификатор, описом који представља њихов изглед, као и карактеристикама везаним за њихов тип. Значи *Weapon/Armor* има специфичне статистике, као што су штета коју оружје наноси или степен заштите који оклоп пружа. Ови објекти могу бити опремљени од стране играча и утичу на његову борбену ефикасност и одбрану
- **Player** – Представља самог играча у игри. Дефинисан је именом, које служи као јединствени идентификатор, описом и карактеристикама играча. Неке од карактеристика су: количина здравља, број поена левела и вештина, предмети у његовом инвентару, које типове оружја и оклопа може да носи, као и његова тренутна или почетна локација у свету игре.

Након извршене комплетне анализе домена у наставку вршимо дефинисање језика и његове граматике помоћу *textX* мета-језика.

5.1 *gameDSL.tx*

У наставку овог поглавља детаљније ћемо обрадити читав мета-модел, односно језик који омогућава дефинисање изгледа света игре и начина на који играч интерагује са њим. Размотрићемо структуру и синтаксу језика, као и различите елементе који се користе за креирање и описивање интерактивног света и свих могућности које играч има у том окружењу.

5.1.1 Свет игрице

Основно правило зове се *GameWorld* и оно је задужено да повеже све објекте, сва оружја, све оклопе, све регије и играча у једно. Дефинисан је листом постојећих региона *regions*, листом постојећих објекта односно *item-a* *items*,

играчем односно *player*-ом, листом свих непријатеља, листом свих оружја, листом свих оклопа, листом подешавања и стартном *start_position* и финалном *final_position* позицијом. Правило *GameWorld* се може видети на Слика 5.1.1.

```
GameWorld:  
    regions += Region  
    items += Item  
    player = Player  
    enemies += Enemy  
    weapons += Weapon  
    armors *= Armor  
    settings += GeneralSettings  
    'start_position' start_position = [Region]  
    'final_position' final_position = [Region]  
;
```

Слика 5.1.1 Дефинисање интерактивног света игре

5.1.2 Регија (*Region*)

Регија односно *Region* представља део света у који можемо да ставимо играча, објекте, оружја и оклопе. Дефинисан је именом које представља јединствени идентификатор смештеног након кључне речи '*region*' пропраћеним блоком ограниченим витичастим заградама '{', '}' у којем се налази блок ограничен малим заградама који се завршава са '#' чиме се поставља да је у питању произвољан редослед (Слика 5.1.2.a).

```
Region:  
    'region' name = GWID '{'  
    (   
        'portrayal' portrayal = STRING  
        ('contains' contains *= [Containable][','])?  
        '::'  
        connections *= Connection[',']  
        '::'  
        ('requirements' requirements += Requirement[','])?  
        ('environmental_dmg' environmental_dmg = EnvironmentalDamage)?  
    )#  
    '}'  
;
```

Слика 5.1.2.a Дефинисање регије

Унутар блоковског тела се налазе: *portrayal* – опис регије, *contains* – листа ствари које регија садржи, *connections* – листа конекција ка другим регијама, *requirements* – листа услова за приступ регији, *environmental_dmg* – количина здравља које регија скида играчу након његовог уласка у њу.

Правило *Containable* (Слика 5.1.2.б) представља листу врста ентитета.

```
Containable:  
    Item | Weapon | Armor  
;
```

Слика 5.1.2.б Дефинисање врста ентитета

Правило *Connection* (Слика 5.1.2.в) је описано са смером *direction* и са дестинацијом *target*. Смер је дефинисан на начин који је приказан на слици испод (Слика 5.1.2.г).

```
Connection:  
    direction = Direction '->' target = GWID  
;
```

Слика 5.1.2.в Дефинисање правила Connection

```
Direction:  
    'N' | 'S' | 'E' | 'W'  
;
```

Слика 5.1.2.г Дефинисање правила Direction

Правило *Requirement* (Слика 5.1.2.đ) представља само *item* који се користи као услов уласка у регију.

```
Requirement:  
    item = GWID  
;
```

Слика 5.1.2.đ Дефинисање правила Requirement

```
EnvironmentalDamage:  
    'damage' amount = INT  
;
```

Слика 5.1.2.ђ Дефинисање правила EnvironmentalDamage

Правило *EnvironmentalDamage* (Слика 5.1.2.ђ) описано је са кључном речју *damage* која је пропраћена целим бројем који представља количину здравља која ће се скинути играчу.

5.1.3 Објекат (*Item*)

Item представља објекат који се налази у нашем свету. Већину њих је могуће покупити, испустити, отворити или искористити. Дефинисан је називом који представља јединствени идентификатор који је смештен након кључне речи *item*, након ког се налази тело односно блок објекта које је смештено унутар витичастих заграда Слика 5.1.3.a.

```
Item:  
  'item' name = GWID '{'  
    (   
      activations *= ActivationProperties  
      'portrayal' portrayal = STRING  
      ('contains' contains *= [Containable][','])?  
      'isStatic' isStatic = BOOL  
    )#  
  }'  
;
```

Слика 5.1.3.a Дефинисање *Item-a*

Унутар блоковског тела налази се листа активационих својстава *activations* која може да има нула или више *ActivationProperties-а*, као и атрибут *isStatic* који очекује појављивање кључне речи *isStatic* пре њеног самог навођења. *isStatic* је атрибут који нам говори да ли је тај објекат статичан, односно да ли играч може да га покупи или је он фиксан део регије. Има вредност *BOOL*.

ActivationProperties представља активације које један *Item* односно објекат може да има. Састоји се од кључне речи *activation* која је затим пропраћена правилом *ActivationAction*. *ActivationAction* представља правило којем можемо да бирамо између две акције враћања здравља играчу или враћања мане играчу. Тачније речено имамо избор између *RestoreHealthAction* и *RestoreManaAction*. *ActivationProperties* и *ActivationAction* се могу видети на Слика 5.1.3.б, а *RestoreHealthAction* и *RestoreManaAction* се могу видети на Слика 5.1.3.в.

```

ActivationProperties:
    'activation' action = ActivationAction
;

ActivationAction:
    RestoreHealthAction | RestoreManaAction
;

```

Слика 5.1.3.6 Дефинисање правила *ActivationProperties* и *ActivationAction*

```

RestoreHealthAction:
    'heal' amount = INT
;

RestoreManaAction:
    'restoreMana' amount = INT
;

```

Слика 5.1.3.8 Дефинисање правила *RestoreHealthAction* и *RestoreManaAction*

Правила *RestoreHealthAction* и *RestoreManaAction* су описана респективно са кључним речима *heal* и *restoreMana* која су пропраћена целим бројем којим је назначено колико здравља или мане ће играчу бити додљено након окидања једног од ова два правила.

5.1.4 Играч (*Player*)

Правило *Player* дефинише сва својства повезана са играчем унутар игре. Дефинисан је називом који представља јединствени идентификатор који је смештен након кључне речи *player*. Правило *Player* се може видети на (Слика 5.1.4.a). Ово правило укључује следеће аспекте:

- Опис играча: Дефинише се преко атрибута *portrayal*, који садржи текстуални опис.
- Искуство: Параметри *currentExperience*, *neededExperienceForLevelUp*, и *levelScalingPercentage* одређују тренутно искуство, потребно искуство за напредовање на следећи ниво, и проценат скалирања искуства по нивоу.
- Ниво: Атрибут *level* дефинише тренутни ниво играча унутар игре.

- Позиција: Атрибут *position* дефинише играчеву тренутну позицију унутар игре.
- Инвентар: Инвентар играча је дефинисан унутар витичастих заграда {}, где се наводе сви предмети које играч носи.
- Статистике: Опционални параметри као што су *vigor*, *endurance*, *strength*, и *intelligence* могу бити укључени, као и основни атрибути *health*, *mana*, *damage*, и *defence*.
- Оружја и оклопи: Параметар *canEquip* наводи листу типова оружја и оклопа које играч може да носи или користи.

```
Player:
  'player' name = GWID '{'
    (
      'portrayal' portrayal = STRING
      'currentExperience' currentExperience = INT
      'neededExperienceForLevelUp' neededExperienceForLevelUp = INT
      'levelScalingPercentage' levelScalingPercentage = INT
      'level' level = INT
      'position' position = [Region]
      'inventory' '{' inventory *= [Containable][','] '}'
        ('vigor' vigor = INT)?
        ('endurance' endurance = INT)?
        ('strength' strength = INT)?
        ('intelligence' intelligence = INT)?

        'health' health = INT
        ('mana' mana = INT)?
        'damage' damage = INT
        'defence' defence = INT
        ('manaDamage' manaDamage = INT)?
        ('manaDefence' manaDefence = INT)?
        'canEquip' '{' canEquip *= GWID[','] '}'
    )#
  '}'
;
```

Слика 5.1.4.a Дефинисање играча

5.1.4 Непријатељ (*Enemy*)

Правило *Enemy* дефинише сва својства која се односе на непријатеља унутар игре. Дефинисано је називом који представља јединствени идентификатор који је смештен након кључне речи *enemy*. Ово правило обухвата следеће компоненте:

- Опис непријатеља: Атрибут *portrayal* садржи текстуални опис непријатеља.
- Позиција: Атрибут *position* одређује локацију непријатеља унутар игре.
- Здравље и мана: Основне статистике као што су *health* (здравље) и опционални атрибут *mana* (мана) дефинишу виталност непријатеља.
- Плен: Опционални параметар *drops* дефинише предмете које непријатељ може оставити након што је поражен. Ови предмети су наведени унутар витичастих заграда {}. Предмети су типа *Containable* што значи да плен од пораженог непријатеља може бити *item*, *weapon* или *armor*.
- Напади: Атрибут *attacks* садржи листу свих врста напада које непријатељ може изводити. Сваки напад је дефинисан преко правила *AttackType*.
- Искуство: Атрибут *xp* одређује количину искуства коју играч добија након што порази непријатеља.
- Опоравак: Опционални блок *healing* описује могућност самоисцељења непријатеља. Овај блок садржи:
 - *healingChance*: Вероватноћа да ће непријатељ покушати да се излечи.
 - *healingAmount*: Количину здравља коју непријатељ поврати.
 - *healingAmountVariance*: Варијацију у количини здравља коју непријатељ поврати.

Свака од ових компоненти је укључена у тело правила које је смештено унутар витичастих заграда {}, што се може видети на (Слика 5.1.5.a).

```
Enemy:
  'enemy' name = GWID '{'
    (
      'portrayal' portrayal = STRING
      'position' position = [Region]
      'health' health = INT
      ('mana' mana = INT)?
      ('drops' '{}' inventory *= [Containable][' ',' '] ')')?
      'attacks' '{}' attackTypes += AttackType '{}'
      'xp' xp = INT
      ('healing' '{}'
        (
          'chance' healingChance = FLOAT
          'amount' healingAmount = INT
          'amountVariance' healingAmountVariance = FLOAT
        )#
      '}')?
    )#
  '}'
;
```

Слика 5.1.5.a Дефинисање непријатеља

Правило *AttackType* описује појединачне нападе које непријатељ може користити. Дефинисано је називом који представља јединствени идентификатор који је смештен након кључне речи *attack* (Слика 5.1.5.6). *AttackType* правило се састоји од следећих компоненти:

- Штета по здравље: Атрибут *healthDamage* дефинише основну штету по здравље коју напад наноси, док *healthDamageVariance* одређује варијацију у штети.
- Штета по ману: Опционални атрибути *manaDamage* и *manaDamageVariance* дефинишу штету по ману и њену варијацију.
- Трошкови напада: Опционални атрибути *healthCost* и *manaCost* одређују трошкове напада у смислу здравља и мана непријатеља који користи напад.
- Учесталост: Атрибут *frequency* одређује колико често непријатељ користи овај напад.

```
AttackType:  
    'attack' name = GWID '{'  
        ('healthDamage' healthDamage = INT  
         'healthDamageVariance' healthDamageVariance = FLOAT  
         ('manaDamage' manaDamage = INT)?  
         ('manaDamageVariance' manaDamageVariance = FLOAT)?  
         ('healthCost' healthCost = INT)?  
         ('manaCost' manaCost = INT)?  
         'frequency' frequency = FLOAT  
     )#  
    }'  
;
```

Слика 5.1.5.а Дефинисање *AttackType* правила

5.1.6 Оружја и оклоп

Правило *Weapon* дефинише сва својства која се односе на оружје унутар игре. Дефинисано је називом који представља јединствени идентификатор и који је смештен након кључне речи '*weapon*' Слика 5.1.6.а . Ово правило обухвата следеће компоненте:

- Опис оружја: Атрибут *portrayal* садржи текстуални опис оружја.
- Тип оружја: Атрибут *type* дефинише тип оружја, коришћењем јединственог идентификатора.
- Штета по здравље: Атрибут *healthDamage* одређује основну количину штете по здравље коју оружје наноси.
- Штета по ману: Опционални атрибут *manaDamage* одређује количину штете коју оружје наноси противнику мани.
- Трошкови употребе: Опционални атрибуты *healthCost* и *manaCost* дефинишу трошкове употребе оружја у смислу здравља и мана.
- Захтевани ниво: Опционални атрибут *requiredLevel* одређује минимални ниво који играч мора да достигне да би могао да користи ово оружје.
- Модификатори: Опционални блок *modifiers* садржи листу модификатора који могу променити својства оружја. Модификатори су наведени унутар витичастих заграда {}.

```
Weapon:
  'weapon' name = GWID '{'
    (
      'portrayal' portrayal = STRING
      'type' type = GWID
      'healthDamage' healthDamage = INT
      ('manaDamage' manaDamage = INT)?
      ('healthCost' healthCost = INT)?
      ('manaCost' manaCost = INT)?
      ('requiredLevel' requiredLevel = INT)?
      ('modifiers' '{' modifiers *= Modifier '}')?
    )#
  '}'
;
```

Слика 5.1.6.a Дефинисање оружја

Правило *Armor* дефинише сва својства која се односе на оклоп. Дефинисано је називом који представља јединствени идентификатор који је смештен након кључне речи '*armor*' Слика 5.1.6.б. Правило обухвата следеће компоненте:

- Опис оклопа: Атрибут *portrayal* садржи текстуални опис оклопа.
- Тип оклопа: Атрибут *type* дефинише тип оклопа, коришћењем јединственог идентификатора.
- Одбрана: Атрибут *defense* одређује основну количину одбране коју оклоп пружа.
- Одбрана мане: Опционални атрибут *manaDefense* дефинише количину одбране од напада, који циљају ману, коју оклоп пружа.

- Захтевани ниво: Опционални атрибут *requiredLevel* одређује минимални ниво који играч мора да достигне да би могао да користи овај окlop.
- Модификатори: Опционални блок *modifiers* садржи листу модификатора који могу променити својства оклопа. Модификатори су наведени унутар витичастих заграда {}.

```
Armor:
  'armor' name = GWID '{'
    (
      'portrayal' portrayal = STRING
      'type' type = GWID
      'defense' defense = INT
      ('manaDefense' manaDefense = INT)?
      ('requiredLevel' requiredLevel = INT)?
      ('modifiers' '{' modifiers *= Modifier '}')?
    )#
  '}'
;
```

Слика 5.1.6.б Дефинисање оклопа

Правило *Modifier* дефинише модификаторе који могу променити одређене атрибуте унутар игре. Дефинисано је као блок који почиње кључном речи '*modifier*', након које следи тело правила смештено унутар витичастих заграда {}. Слика 5.1.6.в. Ово правило обухвата следеће компоненте:

- Модификовани атрибут: Атрибут *modifies* одређује који атрибут се модификује, користећи вредности из скупа *Modifiable*.
- Коефицијенти: Атрибут *coefficients* представља листу вредности који се примењују као коефицијенти за модификацију изабраног атрибута. Вредности у листи су раздвојене запетама.

```
Modifier:
  'modifier {'|
    'modifies' modifiableAttribute = Modifiable
    'coefficients' coefficients += FLOAT[',']
  '}'
;
```

Слика 5.1.6.в Дефинисање *Modifier* правила

Правило *Modifiable* дефинише скуп атрибута који могу бити модификовани помоћу правила *Modifier*. Овај скуп укључује следеће опције: *current_max_health*,

current_max_mana, damage, defence, mana_damage, mana_defence. Правило *Modifiable* се може видети на Слика 5.1.6.г.

```
Modifiable:  
  'current_max_health' | 'current_max_mana' | 'damage' | 'defence' | 'mana_damage' | 'mana_defence'  
;
```

Слика 5.1.6.г Дефинисање *Modifiable* правила

5.1.7 GeneralSettings

Правило *GeneralSettings* дефинише општа подешавања која утичу на одређене аспекте игре. Дефинисано је кључном речју '*settings*', након које следи тело правила смештено унутар витичастих заграда {} Слика 5.1.7.а. Ово правило обухвата следеће компоненте:

- Испуштање других оружја: Опционални атрибут *dropOtherWeapons* одређује да ли ће играч испустити друга оружја када опреми ново оружје. Вредност се задаје као булеан (BOOL), што значи да може бити *True* или *False*.
- Испуштање других оклопа: Опционални атрибут *dropOtherArmors* одређује да ли ће играч испустити друге оклопе када опреми нови оклоп. Вредност је такође булеан (BOOL).
- Додатни потез након употребе: Атрибут *additionalTurnAfterUse* дефинише да ли играч добија додатни потез након коришћења одређеног предмета или способности приликом борбе. Вредност је такође булеан.

```
GeneralSettings:  
  'settings' '{'  
    ('dropOtherWeapons' dropOtherWeapons = BOOL)?  
    ('dropOtherArmors' dropOtherArmors = BOOL)?  
    'additionalTurnAfterUse' additionalTurnAfterUse = BOOL?  
  }#  
'}'
```

Слика 5.1.7.а Дефинисање додатних подешавања

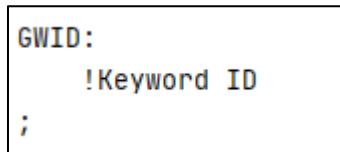
5.1.8 Кључне речи, идентификатори и коментари

Кључне речи (енг. *Keywords*) су резервисане речи које су интегрални део дефинисаног језика. Оне се користе за извођење одређених акција или за дефинисање атрибута у складу са правилима језика, као што је приказано на Слика 5.1.8.a.

```
Keyword:  
'start_position' | 'final_position' | 'drop' | 'take' | 'open' | 'move' | 'N' | 'S' | 'E' | 'W' |  
'portrayal' | 'position' | 'heal' | 'health' | 'contains' | 'inventory' | 'isStatic' |  
'requirements' | 'damage' | 'environmental_dmg' | 'xp' | 'vigor' | 'endurance' | 'strength' | 'currentExperience' |  
'neededExperienceForLevelUp' | 'dropsWeapon' | 'level' |  
'region' | 'item' | 'player' | 'enemy' | 'weapon' | 'use' | 'restoreMana' | 'activation' | 'current_max_health' |  
'current_max_mana' | 'defence' | 'mana_damage' | 'mana_defence' | 'coefficients' | 'modifies' | 'modifier {"'|  
'modifiers' | 'requiredLevel' | 'manaDefense' | 'defense' | 'type' | 'armor' | 'healthDamage' | 'manaDamage' | 'healthCost' |  
'manaCost' | 'attack' | 'healthDamageVariance' | 'manaDamageVariance' | 'frequency' | 'levelScalingPercentage' |  
'intelligence' | 'mana' | 'manaDefence' | 'canEquip' | 'settings' | 'dropOtherWeapons' | 'dropOtherArmors' | 'additionalTurnAfterUse' |  
'drops' | 'attacks' | 'healing' | 'chance' | 'amount' | 'amountVariance'  
;
```

Слика 5.1.8.a Кључне речи

Због присуства кључних речи које корисник може да унесе као име објекта, регије, играча и тако даље, уграђени идентификатор *textX-a*, *ID*, није довољан. Зато уводимо ново правило под називом *GWID* (*Game World ID*). Правило *GWID* функционише слично основном правилу *ID* у *textX-y*, али са додатним проширењем које осигурува да идентификатор не садржи ниједну од кључних речи дефинисаног језика, као што је приказано на Слика 5.1.8.b. Ово правило је кључно јер спречава корисника да користи кључне речи као јединствене идентификаторе, што је од великог значаја за даље успешно парсирање.



Слика 5.1.8.b Дефинисање GWID правила

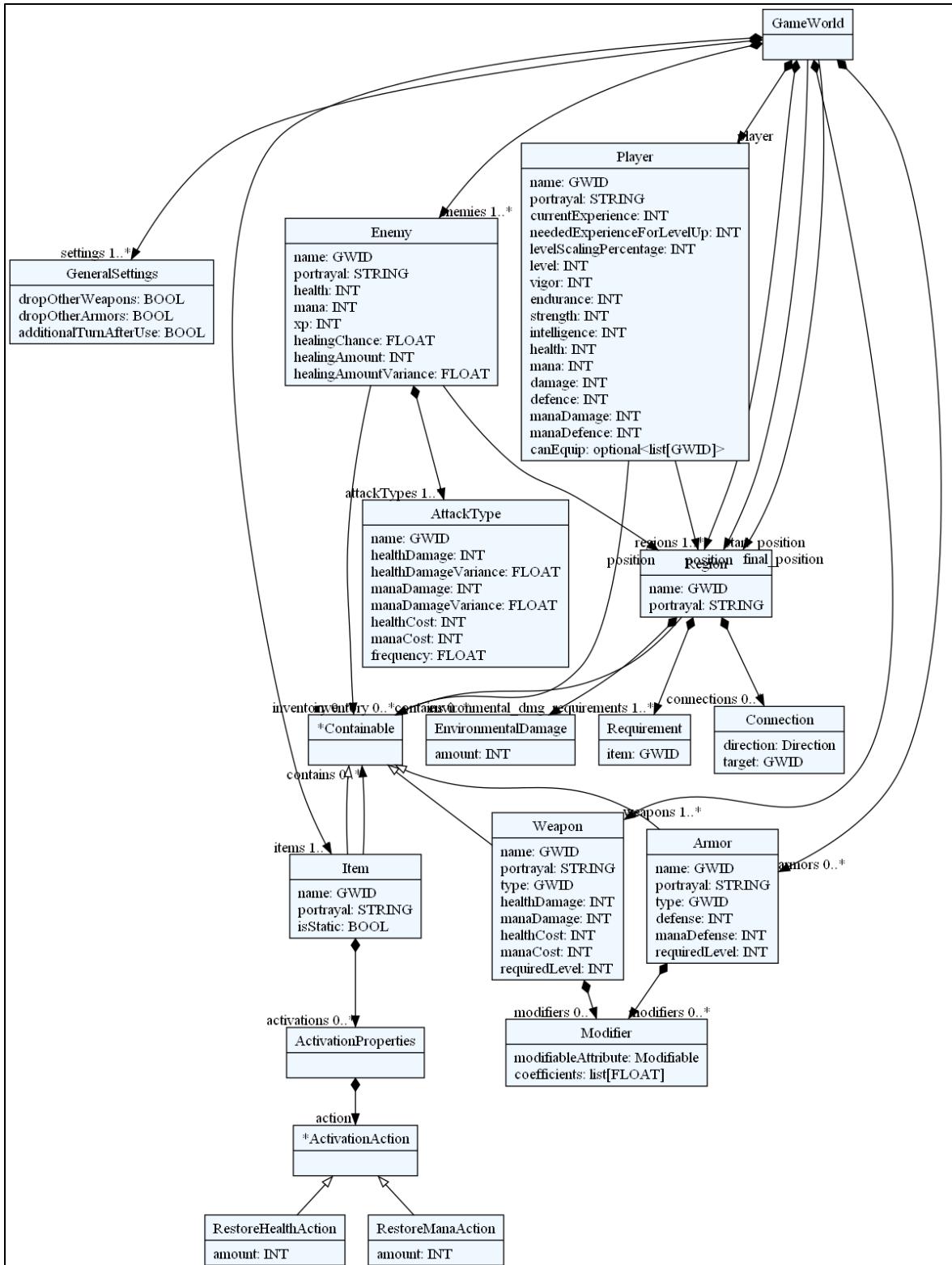
Коментари су омогућени увођењем правила *Comment*. Ово правило омогућава да у коду унесемо коментаре, као што је приказано на Слика 5.1.8.v. Коментар почиње са /*, а завршава се са */, обухватајући све што се налази између ових ознака.

```
Comment:  
/\/*(.|\n)*?\*/  
;
```

Слика 5.1.8.в Дефинисање GWID правила

5.2 Визуализација мета-модела

Приказ визуализације мета-модела одрађено је помоћу *textX* библиотеке која од новодефинисаног језика генерише *.dot* формат који се затим користи за генерисање графичког приказа коришћењем *GraphViz* [25] Слика 5.2.a. Због превелике количине кључних речи у *Keyword* делу који се налази заједно са *Comment*, *Direction*, *GWID* и *Modifiable* на графичком приказу њиховв приказ није могућ, односно није довољно читко и јасно да се може разазнати шта је шта.



Слика 5.2.а Графички приказ мета-модела

6. Интерпретер

Да бисмо могли да манипулишемо интерактивним светом, крећемо се у њему и интерагујемо са њим, неопходно је да направимо интерпретер тог света. За потребе овог мастер рада, интерпретер је написан у програмском језику *Python*. Основно учитавање модела омогућава нам библиотека *textX*.

6.1 Python класе

Новокреиране *Python* класе морају да се подударају са ентитетима из мета-модела. За потребе овог рада на основу већ појашњеног мета-модела креирани су следеће класе: *GameWorld*, *Region*, *Item*, *Player*, *Actions*, *Armor*, *Enemy*, *GeneralSettings*, *Weapon*.

GameWorld класа представља основну структуру интерактивног света у игри. Она обједињује све битне елементе потребне за креирање, управљање и интеракцију са светом игре Слика 6.1.a. У наставку је детаљнији опис сваког атрибута који чини ову класу:

- *regions*: Листа која садржи све регије унутар света игре. Регије могу представљати различите локације или области кроз које се играч креће током игре.
- *items*: Речник који садржи све објекте односно *item*-е који су присутни у свету игре. Кључеви у речнику могу бити имена то јест идентификатори објекта, док су вредности сами објекти.
- *enemies*: Листа непријатеља који постоје у свету игре. Сваки непријатељ може имати различите особине, као што су ниво снаге, здравља, напада и тако даље.
- *weapons*: Речник који садржи оружја доступна у игри. Слично као и код *item*-а, кључеви су идентификатори оружја, а вредности су сами објекти који представљају оружја.
- *armors*: Речник који садржи оклопе доступне у игри. Оклопи служе за заштиту играча од напада непријатеља.
- *player*: Атрибут који представља играча у игри.
- *current_enemy*: Атрибут који представља тренутног непријатеља са којим се играч суочава.
- *start_position*: Типично почетна позиција играча у свету игре. Овај атрибут дефинише где игра почине.
- *final_position*: Финална позиција у игри, односно циљ до којег играч треба да дође да би успешно завршио игру.
- *prev_direction*: Овај атрибут чува информацију о претходном правцу у којем се играч кретао. Омогућава играчу да побегне из борбе без даљег напредовања у игри.
- *opposite_dirs*: Речник који мапира супротне правце кретања. Ова структура помаже у једноставнијем одређивању супротног смера током кретања играча.

- *settings*: Атрибут који чува подешавања игре.

```
class GameWorld:
    def __init__(self):
        self.regions = []
        self.items = {}
        self.enemies = []
        self.weapons = {}
        self.armors = {}
        self.player = None
        self.current_enemy = None
        self.start_position = None
        self.final_position = None
        self.prev_direction = None
        self.opposite_dirs = {"N": "S", "S": "N", "E": "W", "W": "E"}
        self.settings = None
```

Слика 6.1.a GameWorld класа

У овој класи долази до генерисања игре при избору мода играња бесконачне игре. Функције као што су *generate_new_regions*, *generate_new_enemy*, *generate_new_item*, *generate_new_weapon* и тако даље управно служе за генерисање бесконачне игре. Најбитнија функција у овој класи јесте *call_chatgpt* која врши позив ка OpenAI API-ју који заправо генерише суштину новокреираних регија, непријатеља, оружја и тако даље. Детаљнија прича о генерисању бесконачне игре следи у наредним поглављима.

Region класа представља појединачну област или регију у интерактивном свету игре. Ова класа омогућава управљање специфичним карактеристикама и елементима који се односе на одређену регију у игри (Слика 6.1.б). У наставку следи опис сваког атрибута који је део ове класе:

- *name*: Назив регије. Овај атрибут представља јединствено име које идентификује одређену регију у свету игре.
- *portrayal*: Опис регије. Овај атрибут садржи текстуални опис регије који играчу пружа информације о томе како регија изгледа или које карактеристике има.
- *items*: Речник објекта односно *item*-а који се налазе у регији. Кључеви у речнику су идентификатори или називи објекта, док су вредности сами објекти. Овај атрибут омогућава складиштење и управљање предметима који се могу наћи у одређеној регији.
- *connections*: Речник који представља повезаност ове регије са другим регијама. Кључеви у речнику су називи правца, а вредности су друге регије до којих играч може доћи из ове регије. Овај атрибут омогућава дефинисање мреже повезаних регија унутар света игре.

- *requirements*: Листа услова који морају бити испуњени да би играч могао приступити овој регији.
- *environmental_dmg*: Атрибут који представља количину штете коју окружење ове регије може нанети играчу.

```
class Region:
    def __init__(self, name, portrayal):
        self.name = name
        self.portrayal = portrayal
        self.items = {}
        self.connections = {}
        self.requirements = []
        self.environmental_dmg = None
```

Слика 6.1.б *Region* класа

Атрибути *name* и *portrayal* у свакој класи означавају исте ствари везане за класу у којој се налазе, из тог разлога су детаљније појашњени у класи **Region.

Item класа представља објекат или предмет у свету игре. Ови објекти могу имати различите функције и карактеристике (Слика 6.1.в). У наставку следи опис атрибута ове класе:

- *contains*: Листа предмета који се налазе унутар овог предмета. Ово омогућава да један предмет садржи друге предмете. Листа може бити празна ако предмет не садржи друге предмете.
- *activations*: Листа акција које овај предмет може да окине.
- *isStatic*: Атрибут која означава да ли је предмет статичан. Статични предмети су они који су фиксирани на једном месту и не могу се померати током игре. Ако је ова вредност *True*, предмет је статичан, а ако је *False*, предмет може бити покупљен. Служи да ствари као што су кутија са стварима не могу да се покупе и ставе у играчев посед.

```
class Item:
    def __init__(self, name, portrayal, is_static):
        self.name = name
        self.portrayal = portrayal
        self.contains = []
        self.activations = []
        self.isStatic = is_static
```

Слика 6.1.в *Item* класа

Player класа представља играча у свету игре, укључујући све његове атрибуте, статистике и опрему. Ова класа омогућава управљање играчевим карактеристикама, праћење његовог напретка и интеракцију са светом око њега Слика 6.1.г. У наставку је опис сваког атрибута који чини ову класу:

- *position*: Тренутна позиција играча у свету игре. Иницијално је постављена на *start_position*, што представља почетну локацију на којој играч почиње своје путовање.
- *inventory*: Листа предмета које играч носи са собом. Ова листа омогућава играчу да сакупља и користи предмете током игре.
- *health*: Тренутни ниво здравља играча. Ако здравље падне на нулу, играч умире и у зависности од мода игре или се игра завршава или играч испушта све ствари из свог инвентоара и поново се рађа у стартној регији игре.
- *initial_health*: Почетно здравље играча, постављено на почетну вредност здравља. Након умирања играча његово здравље се поставља на ову вредност.
- *current_max_health*: Тренутни максимални ниво здравља који играч може имати. Ова вредност може варирати током игре услед разних фактора.
- *unmodified_current_max_health*: Немодификовани максимални ниво здравља без икаквих модификација.
- *current_experience*: Тренутни ниво искуства који играч има. Овај атрибут прати напредак играча и повећава се током игре.
- *needed_experience_for_level_up*: Количина искуства потребна за напредовање на следећи ниво. Играчу је потребно да сакупи довољно искуства како би прешао на виши ниво.
- *levelScalingPercentage*: Проценат скалирања који одређује колико је тешко напредовати на вишим нивоима.
- *level*: Тренутни ниво играча. Почетни ниво је 1, али играч може напредовати кроз нивое сакупљањем искуства.
- *level_points*: Поени који се добијају при напредовању на нови ниво и могу се користити за побољшање играчевих атрибута.
- *base_health*: Основни ниво здравља играча, односно здравље без модifikatora.
- *mana*: Тренутни ниво мане играча, која се користи за при нападима који користе ману.
- *current_max_mana*: Тренутни максимални ниво мане који играч може имати.
- *unmodified_current_max_mana*: Нефмодификовани максимални ниво мане без икаквих модификација.
- *base_mana*: Основни ниво мане играча.

- *damage*: Тренутни ниво штете коју играч може нанети непријатељима. Ова вредност може бити модификована оружјем.
- *unmodified_damage*: Немодификовани ниво штете без икаквих модификација.
- *defence*: Тренутни ниво одбране играча, који одређује колико штете играч може апсорбовати од напада.
- *unmodified_defence*: Немодификовани ниво одбране без икаквих модификација.
- *mana_damage*: Штета коју играч може нанети користећи мана нападе.
- *unmodified_mana_damage*: Немодификовани ниво мана штете без модификација.
- *mana_defence*: Ниво одбране играча од мана напада.
- *unmodified_mana_defence*: Немодификовани ниво одбране од мана напада без модификација.
- *weapon*: Оружје које играч тренутно користи. Иницијално је постављено на *None*, што значи да играч не носи никакво оружје на почетку.
- *armor*: Оклоп који играч тренутно носи. Иницијално је постављен на *None*.
- *can_equip*: Листа врста опреме коју играч може опремити. Овај атрибут омогућава играчу да користи различите врсте оружја и оклопа.
- *vigor*: Атрибут који представља виталност играча, утиче на његову количину здравља.
- *endurance*: Атрибут који утиче на издржљивост играча, омогућавајући му да дуже траје у борби или активностима.
- *strength*: Атрибут који одређује физичку снагу играча и утиче на количину штете коју наноси.
- *intelligence*: Атрибут који утиче на играчеву способност коришћења магије односно на ефикасност играчевих мана напада.

Акције као што су покупи, искористи, отвори, помери се, излечи се, повећај неки од атрибута, увид у инвентоар, увид о стању тренутних атрибута, коришћење оружја, стављање оклопа су имплементиране преко функција класе *Player*.

```

class Player:
    def __init__(self, name, start_position, vigor, endurance,
                 strength, intelligence, health, mana, damage, defence,
                 mana_damage, mana_defence):
        self.name = name
        self.position = start_position
        self.inventory = []
        self.health = health
        self.initial_health = health
        self.current_max_health = health
        self.unmodified_current_max_health = health
        self.current_experience = 0
        self.needed_experience_for_level_up = 100
        self.levelScalingPercentage = 10
        self.level = 1
        self.level_points = 0
        self.base_health = health
        self.mana = mana
        self.current_max_mana = mana
        self.unmodified_current_max_mana = mana
        self.base_mana = mana
        self.damage = damage
        self.unmodified_damage = damage
        self.defence = defence
        self.unmodified_defence = defence
        self.mana_damage = mana_damage
        self.unmodified_mana_damage = mana_damage
        self.mana_defence = mana_defence
        self.unmodified_mana_defence = mana_defence
        self.weapon = None
        self.armor = None
        self.can_equip = []
        self.vigor = vigor
        self.endurance = endurance
        self.strength = strength
        self.intelligence = intelligence

```

Слика 6.1.2 Player класа

Enemy класа представља непријатеља у свету игре. Ова класа дефинише све битне карактеристике и способности које непријатељ може имати Слика 6.1.đ. У наставку је описан сваког атрибута који чини ову класу:

- *position*: Тренутна позиција непријатеља у свету игре. Овај атрибут одређује где се непријатељ налази у односу на играча и друге елементе у свету.
- *health*: Тренутни ниво здравља непријатеља. Ако здравље падне на нулу, непријатељ је поражен.

- *initial_health*: Почетни ниво здравља непријатеља, постављен на вредност здравља коју непријатељ има на почетку.
- *mana*: Тренутни ниво мане непријатеља, која се може користити за магичне нападе.
- *initial_mana*: Почетни ниво мане непријатеља, постављен на вредност мане коју непријатељ има на почетку.
- *xp*: Искуство које играч добија након што порази непријатеља. Ова вредност доприноси напредовању играча у игри.
- *items_to_drop*: Речник предмета који непријатељ може испустити након што је поражен. Кључеви у речнику су идентификатори односно називи предмета, а вредности су ти предмети.
- *attacks*: Листа напада које непријатељ може извршити током борбе. Ови напади могу бити физички или магични који наносе штету играчу или изазивају друге ефекте.
- *healing_chance*: Вероватноћа да ће се непријатељ излечити током борбе.
- *healing_amount*: Количина здравља коју непријатељ може вратити себи када се лечи.
- *healing_amount_variance*: Варијација у количини лечења, која се додаје или одузима од основне вредности лечења.

```
class Enemy:
    def __init__(self, name, portrayal, position, health, mana, xp):
        self.name = name
        self.portrayal = portrayal
        self.position = position
        self.health = health
        self.initial_health = health

        self.mana = mana
        self.initial_mana = mana
        self.xp = xp

        self.items_to_drop = {}
        self.attacks = []

        self.healing_chance = 0
        self.healing_amount = 0
        self.healing_amount_variance = 0
```

Слика 6.1.đ Enemу класа

Armor калса представља оклоп који играч може носити у свету игре. Ова класа дефинише различите карактеристике оклопа, укључујући његову заштиту, тип, и захтевани ниво за коришћење Слика 6.1.ђ. У наставку је описан сваког атрибута који чини ову класу:

- *type*: Тип оклопа.
- *defense*: Вредност физичке одбране коју оклоп пружа играчу. Овај атрибут одређује колико штете оклоп може апсорбовати од физичких напада, чиме смањује штету коју играч прима.
- *mana_defense*: Вредност мана одбране коју оклоп пружа играчу. Овај атрибут одређује колико штете оклоп може апсорбовати од мана напада, чиме смањује штету коју играч прима од њих.
- *required_level*: Захтевани ниво који играч мора да достигне да би могао да користи овај оклоп.
- *modifiers*: Речник модификатора који могу променити или побољшати карактеристике оклопа.

```
class Armor:
    def __init__(self, name, portrayal, armorType, defense, mana_defense, required_level):
        self.name = name
        self.portrayal = portrayal
        self.type = armorType
        self.defense = defense
        self.mana_defense = mana_defense
        self.required_level = required_level
        self.modifiers = {}
```

Слика 6.1.ј Armor класа

Weapon класа представља оружје које играч може користити у свету игре. Ова класа дефинише различите карактеристике оружја, укључујући штету коју наноси, тип, и захтеве за коришћење Слика 6.1.e. У наставку је опис сваког атрибута који чини ову класу:

- *health_damage*: Количина физичке штете коју оружје наноси противнику. Овај атрибут одређује колико ће противник изгубити здравља када је погодјен овим оружјем.
- *mana_damage*: Количина магичне штете коју оружје наноси манама противника.
- *health_cost*: Трошак здравља који играч плаћа када користи ово оружје. Овај атрибут указује на то да неко оружје може исцрпети део играчевог здравља када се користи, што је често случај код моћних или проклетих оружја.
- *mana_cost*: Трошак мане који играч плаћа када користи ово оружје. Ово је количина мане која се троши при сваком нападу или коришћењу оружја, што је карактеристично за магична оружја.

```

class Weapon:
    def __init__(self, name, portrayal, weaponType, health_damage, mana_damage, health_cost, mana_cost, required_level):
        self.name = name
        self.portrayal = portrayal
        self.type = weaponType
        self.health_damage = health_damage
        self.mana_damage = mana_damage
        self.health_cost = health_cost
        self.mana_cost = mana_cost
        self.required_level = required_level
        self.modifiers = {}

```

Слика 6.1.е Weapon класа

HealAction и **RestoreManaAction** класе представљају акције лечења у виду здравља и мана ресурса коју играч може да активира. Ове класе дефинишу количину здравља или мане коју ће играч повратити и садржи метод који примењује ту акцију на играча Слика 6.1.ж.

- *amount*: Количина здравља или мане коју ће играч повратити када се акција активира.
- *activate*: Метод који активира акцију лечења. Овај метод позива функцију *heal* или *restore_mana* на објекту играча (*player*), повећавајући здравље или ману играча за количину дефинисану у атрибуту *amount*.
 - *player.heal(self.amount)*: Овај позив повећава тренутно здравље играча за вредност *amount*. Метод *heal* треба бити дефинисан у класи *Player* и одговоран је за додавање здравља играчу.
 - *player.restore_mana(self.amount)*: Овај позив повећава тренутни ниво мане играча за вредност *amount*. Метод *restore_mana* треба бити дефинисан у класи *Player* и одговоран је за додавање мане играчу.

<pre> class RestoreManaAction: def __init__(self, amount): self.amount = amount def activate(self, player): player.restore_mana(self.amount) </pre>	<pre> class HealAction: def __init__(self, amount): self.amount = amount def activate(self, player): player.heal(self.amount) </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------

Слика 6.1.ж Класе акција

GeneralSettings класа служи за дефинисање општих подешавања која утичу на одређене аспекте игре, као што су понашање приликом узимања накнадног оружја у руке или оклопа на себе, као и додатни потези након коришћења одређених акција. Ова подешавања могу бити конфигурисана приликом креирања објекта ове класе и користе се за контролу специфичних механика игре Слика 6.1.з.

drop_other_weapons: Овај атрибут представља подешавање које одређује да ли ће се оружје у рукама играча испустити када играч покупи ново оружје.

drop_other_armors: Слично као и претходни атрибут, овај одређује да ли ће се оклоп, који играч носи, испустити када играч узме нови оклоп или не.

additional_turn_after_use: Овај атрибут одређује да ли играч добија додатни потез након коришћења предмета односно акције приликом борбе или не.

```
class GeneralSettings:  
    def __init__(self, drop_other_weapons, drop_other_armors, additional_turn_after_use):  
        self.drop_other_weapons = drop_other_weapons  
        self.drop_other_armors = drop_other_armors  
        self.additional_turn_after_use = additional_turn_after_use
```

Слика 6.1.3 *GeneralSettings* класа

6.2 *interpreter.py*

У фајлу *interpreter.py* написана је функција *parse_dsl*, која је задужена за парсирање кода игрице на основу нашег прилагођеног језика, односно мета-модела. Функција користи библиотеку *textX* за учитавање и парсирање мета-модела, омогућавајући претварање текста из специфичног доменског језика у структуре које наш програм може да користи. Учитавање мета-модела и његово парсирање се обавља помоћу функције *metamodel_from_file* из *textX* библиотеке. Ова функција учитава дефиницију мета-модела из датотеке и користи га за парсирање *DSL* кода, креирајући структуре података које се могу користити у програму Слика 6.2.a.

```
def parse_dsl(dsl_path, game_path):  
    this_folder = dirname(__file__)  
    dsl_mm = metamodel_from_file(join(this_folder, dsl_path))  
    model = dsl_mm.model_from_file(join("games\\\" + game_path, game_path))  
    game_world = GameWorld()
```

Слика 6.2.a Почетни део *parse_dsl* функције

Остатак *parse_dsl* функције се дели на пар делова. Постоји део за креирање регија, део за креирање *Item*-а, део за креирање оружја, део за креирање оклопа, део за креирање играча, део за креирање непријатеља, део за постављање стартне и финалне регије фиктивне игрице као и део за генерална подешавања.

Креирање регија је засновано на проласку кроз листу регија из испарсираног модела попуњавајући атрибуте ново-креiranог објекта *Python* класе *Region* Слика 6.2.6. Након проласка кроз све атрибуте и комплетног попуњавања новог објекта класе *Region* тај исти објекат смештамо у листу регија *regions* објекта класе *GameWorld*.

```

# Create regions
for region_def in model.regions:
    region = Region(region_def.name, region_def.portrayal)
    for connection in region_def.connections:
        region.add_connection(connection.direction, connection.target)
    for requirement in region_def.requirements:
        region.add_requirements(requirement)
    if region_def.environmental_dmg:
        region.add_environmental_dmg(region_def.environmental_dmg)
    for item in region_def.contains:
        region.add_item(item)
game_world.regions.append(region)

```

Слика 6.2.б Креирање регија

Креирање *Item*-а је засновано на проласку кроз листу *Item*-а из испарсираног модела попуњавајући атрибуте ново-креираног објекта Python класе *Item* Слика 6.2.в. Након проласка кроз све атрибуте и комплетног попуњавања односно креирања новог објекта класе *Item* тај исти објекат смештамо у листу *Item*-а *items* објекта класе *GameWorld*.

```

# Create items
for item_def in model.items:
    item = Item(item_def.name, item_def.portrayal, item_def.isStatic)
    item.contains = [item.name for item in item_def.contains]
    for activation in item_def.activations:
        action_name = activation.action.__class__.__name__
        if action_name == "RestoreHealthAction":
            item.activations.append(HealAction(activation.action.amount))
        elif action_name == "RestoreManaAction":
            item.activations.append(RestoreManaAction(activation.action.amount))
    game_world.items[item.name] = item

```

Слика 6.2.в Креирање *Item*-а

Креирање оружја је засновано на проласку кроз листу оружја из испарсираног модела попуњавајући атрибуте ново-креираног објекта Python класе *Weapon* Слика 6.2.g. Након проласка кроз све атрибуте и комплетног попуњавања односно креирања новог објекта класе *Weapon* тај исти објекат смештамо у листу *Weapon*-а *weapons* објекта класе *GameWorld*.

```

# Create weapons
for weapon_def in model.weapons:
    weapon = Weapon(
        weapon_def.name,
        weapon_def.portrayal,
        weapon_def.type,
        weapon_def.healthDamage,
        weapon_def.manaDamage,
        weapon_def.healthCost,
        weapon_def.manaCost,
        weapon_def.requiredLevel
    )
    modifiers = weapon_def.modifiers
    for modifier in modifiers:
        weapon.add_modifier(modifier.modifiableAttribute, modifier.coefficients)
    game_world.weapons[weapon_def.name] = weapon

```

Слика 6.2.g Креирање оружја

Креирање оклопа је засновано на проласку кроз листу оклопа из испарсираног модела попуњавајући атрибуте ново-креiranог објекта *Python* класе *Armor* Слика 6.2.đ. Након проласка кроз све атрибуте и комплетног попуњавања односно креирања новог објекта класе *Armor* тај исти објекат смештамо у листу *Armora-a armors* објекта класе *GameWorld*.

```

# Create weapons
for weapon_def in model.weapons:
    weapon = Weapon(
        weapon_def.name,
        weapon_def.portrayal,
        weapon_def.type,
        weapon_def.healthDamage,
        weapon_def.manaDamage,
        weapon_def.healthCost,
        weapon_def.manaCost,
        weapon_def.requiredLevel
    )
    modifiers = weapon_def.modifiers
    for modifier in modifiers:
        weapon.add_modifier(modifier.modifiableAttribute, modifier.coefficients)
    game_world.weapons[weapon_def.name] = weapon

```

Слика 6.2.đ Креирање оклопа

Креирање играча је засновано на проналаску играча у испарсираном моделу и преузимању његових вредности у ново-креiran објекат *Python* класе *Player* Слика 6.2.ђ. Након успешног копирања података ново-креiranог играча смештамо у *player* атрибут објекта класе *GameWorld*.

```

# Create player
player_def = model.player

initial_position = None
for region in game_world.regions:
    if region.name == player_def.position.name:
        initial_position = region
        break
player = Player(player_def.name, initial_position, player_def.vigor, player_def.endurance, player_def.strength,
               player_def.intelligence,
               player_def.health, player_def.mana, player_def.damage, player_def.defence, player_def.manaDamage,
               player_def.manaDefence)

player.current_experience = player_def.currentExperience
player.needed_experience_for_level_up = player_def.neededExperienceForLevelUp
player.levelScalingPercentage = player_def.levelScalingPercentage
player.level = player_def.level
player.inventory = [item.name for item in player_def.inventory]
player.can_equip = player_def.canEquip
game_world.player = player

```

Слика 6.2.ђ Креирање играча

Креирање непријатеља је засновано на проласку кроз листу непријатеља из испарсираног модела попуњавајући атрибуте ново-креiranог објекта *Python* класе *Enemy*. Слика 6.2.e. Након проласка кроз све атрибуте и комплетног попуњавања односно креирања новог објекта класе *Enemy* тај исти објекат смештамо у листу *Enemy*-а *enemies* објекта класе *GameWorld*.

```

# Create enemies
for enemy_def in model.enemies:
    enemy = Enemy(enemy_def.name.replace("_", " "), enemy_def.portrayal, enemy_def.position, enemy_def.health,
                  enemy_def.mana, enemy_def.xp)
    for attack in enemy_def.attackTypes:
        enemy.attacks.append({
            'name': attack.name,
            'health_damage': attack.healthDamage,
            'health_damage_variance': attack.healthDamageVariance,
            'mana_damage': attack.manaDamage,
            'mana_damage_variance': attack.manaDamageVariance,
            'health_cost': attack.healthCost,
            'mana_cost': attack.manaCost,
            'frequency': attack.frequency
        })
    enemy.healing_chance = enemy_def.healingChance
    enemy.healing_amount = enemy_def.healingAmount
    enemy.healing_amount_variance = enemy_def.healingAmountVariance
    for item in enemy_def.inventory:
        enemy.items_to_drop[item.name] = item
    game_world.enemies.append(enemy)

```

Слика 6.2.e Креирање непријатеља

На самом крају функције *parse_dsl* имамо део са постављањем стартне и финалне регије игре у објекат класе *GameWorld* као и парсирање генералних подешавања *GameWorld*-а Слика 6.2.ж. Повратна вредност саме функције представља објекат *game_world* класе *GameWorld* који смо попунили са свим подацима из нашег испарсираног модела.

```
# Set start and final positions
for player_region in game_world.regions:
    if player_region.name == model.start_position.name:
        game_world.set_start_position(player_region)
    elif player_region.name == model.final_position.name:
        game_world.set_final_position(player_region)

# Set settings
for settings_def in model.settings:
    settings = GeneralSettings(settings_def.dropOtherWeapons, settings_def.dropOtherArmors,
                                settings_def.additionalTurnAfterUse)
    game_world.settings = settings
```

Слика 6.2.ж Крај функције *parse_dsl*

7. GPT генерисање

У овом поглављу ће се детаљније објаснити приступ и коришћење *OpenAI API*-а, односно њиховог *GPT* модела. За потребе оног мастер рада коришћење *GPT* модела се појављује у два сегмента. Део за креирање готових односно игара интерактивне фикције које имају крај. Финални производ овог дела јесте заправо код те игре. Такође *GPT* модел се користи при избору мода за бесконачно играње где је финални производ не готова игра односно код игре већ новогенерисани објекти регије, предмета, непријатеља, оружја и оклопа.

7.1. Генерисање коначних игара интерактивне фикције

Генерисање игара са коначним трајањем користи се када желимо да игра има јасно дефинисан почетак, средину и крај. Ово је обично случај код прича са фиксираним заплетом, где играч на крају досеже одређену циљну тачку или постигне одређену мисију, након чега игра завршава. Изглед промпта који се шаље *GPT* моделу се може видети на Слика 7.1.a.

Овај процес се може поделити на више корака:

- Учитавање граматике и примера игре: Прво, учитава се *grammar* из *textX DSL* датотеке и пример игре. Ово служи као водич *GPT* моделу за креирање нове игре.
- Генерисање игре: Коришћењем *OpenAI GPT* модела, на основу унапред дефинисаних правила и примера, *GPT* генерише код за нову игру која има крај. Важно је напоменути да је посебна пажња посвећена додатку завршне регије, која служи као крај игре.
- Резултат: Функција враћа генерисани код игре, који затим може бити коришћен при игрању.

```
f"Create a game using the following "
f"TextX grammar file -> {grammar} and here is the example of how the "
f"generated game could look like-> {exampleGame}. Just give me the game code "
f"itself and nothing more. Here is the interpreter for the game you are "
f"about to generate -> {interpreter}. Be careful not to mix up"
f" weapons and items. Weapons are entity for themselves! Also add a region"
f"that will be used to end the game, so that when the player steps into it"
f"the game ends. I mean the final region should be region after the final region"
```

Слика 7.1.a GPT промпт

7.2. Мод бесконачног играња

Бесконачни мод играња подразумева да игра нема крајњи циљ или завршну тачку, већ да играч може бесконачно истраживати нове регије, суочавати се са новим изазовима, непријатељима и проналазити нове предмете, оружја и оклопе. Овај мод је креиран тако да увек нуди нешто ново, омогућавајући играчу да истражује свет игре без ограничења.

Неки од кључних елемената за овај мод су следећи:

- *call_chatgpt*: Функција која служи за комуникацију са *GPT* моделом. Користи се за генерисање нових регија, непријатеља, предмета и других елемената у игри на основу унапред дефинисаних промптова.
- *generate_new_regions*: Ова функција је одговорна за креирање нових регија у игри. Генерише не само имена и описе нових регија, већ и све интерактивне елементе у њима, укључујући предмете, оружја, оклопе, непријатеље и повезане области. Кроз ову функцију, игра стално добија нове изазове и могућности истраживања. Идеја о потенцијалној повезаности новокреираних регија са старијима инспирисана је играма попут *"Dark Souls"*, где играч, након проласка кроз свет и савладавања свих изазова, може неочекивано стићи до почетних зона. Овде постаје очигледно колико је напредовао током своје авантуре.
- *explore_new_area*: Ова функција се активира када играч достигне тренутно дефинисану финалну позицију у игри. Она започиње процес генерирања нових регија и наставља игру без краја, омогућавајући играчу да открије нове области и да се суочава са новим изазовима.

Поред горе наведених и појашњених функција постоје и функције које се позивају унутар *generate_new_regions* функције. То су функције та генерирање предмета, оружја, оклопа, непријатеља. У свим наведеним функцијама *GPT* модел се користи за генерирање јединственог назива генерисаног објекта, као и за генерирање описа.

Потребно је нагласити да су ствари као што су број просторија који се генерише, постојање непријатеља у просторији, количина предмета, оружја или оклопа у просторијама, повезаност просторија са другим просторијама генерисани помоћу *Python* библиотеке *random*. На пример, шанса да се појави непријатељ у просторији је 30%.

Приликом генерирања објекта игре кораци су били следећи:

1. Генерирати назив регије имајући у виду називе свих претходних регија
2. Приликом генерирања сваког објекта унутар те регије генерирати назив тог објекта имајући у виду претходне називе те врсте објекта, као и назива

регије у којој се тај објекат налази. У овом случају остали објекти су предмети, оружја и оклоп. Доделити вредности тим новогенерисаним објектима помоћу *Python random* библиотеке.

3. Генерисати непријатеља и његове нападе имајући у виду у којој регији се налази. Овде је битно напоменути да због количине података односно величине описа непријатеља се морао задати лимит од 600 карактера за одговор описа. Доделити вредности новогенерисаним непријатељима и њиховим нападима.
4. На основу пређашње генерисаног назива регије, објеката у регији, као и непријатеља у регији се може генерисати финални опис регије са свим поменутим стварима у виду. Овде је такође битно напоменути да је постављен лимит описа на 600 карактера.

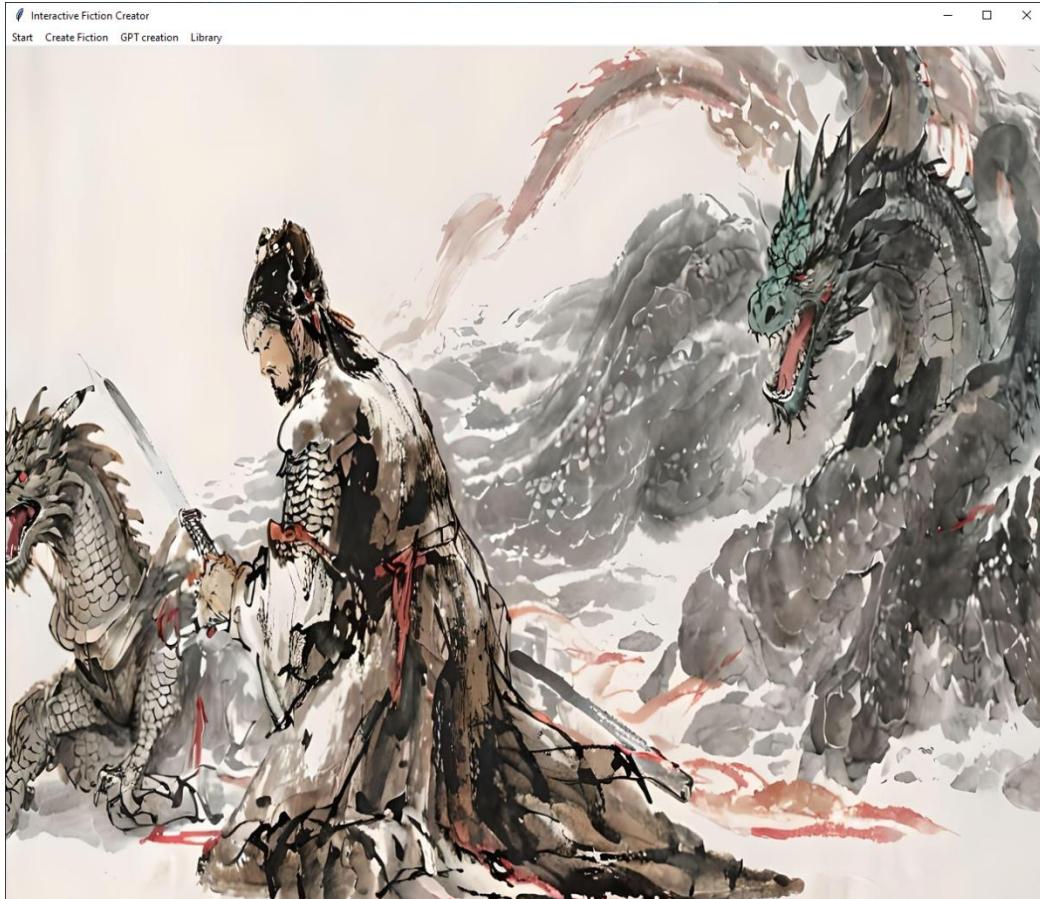
8. Графички интерфејс

Графички кориснички интерфејс (*GUI*) апликације имплементиран је помоћу уgraђене *Python* библиотеке *Tkinter*. Интерфејс је подељен у пет целина, од којих свака има специфичну функцију у оквиру апликације:

- Основни прозор апликације: Овај прозор садржи само навигациону траку која служи као централна тачка за управљање апликацијом.
- *Code Editor*: Овај део интерфејса омогућава корисницима да креирају игре интерактивне функције путем уређивања кода. *Code Editor* пружа алате за писање и уређивање кода, омогућавајући корисницима да директно управљају логиком и садржајем игара.
- Део за креирање слика регија: Ова секција интерфејса омогућава корисницима да креирају визуелне приказе регија у игри. Корисници могу додавати и уређивати слике које представљају различите делове света игре, чиме се визуелно обогаћује искуство играња.
- Део за генерисање игара помоћу промпта: Овај део интерфејса омогућава корисницима да генеришу игре интерактивне функције коришћењем њихових промптова. Генерација игара се врши уз помоћ *GPT* модела, који креира сценарије и садржаје игре на основу унетих промптова.
- Део за играње игара интерактивне функције: Овај део интерфејса служи за играње креираних игара. Корисници могу тестирати односно играти игре интерактивне функције које су сами креирали или које су генерисане путем промпта, директно унутар апликације.

8.1 *gui.py*

Основа графичког корисничког интерфејса се налази баш у овом фајлу. У класи *App* се отвара *TkInter* прозор и задају му се димензије. Поставља се навигациона линија на врху прозора. У тој навигационој траци су заправо смештена четири прозора. Један почетни који је празан и на коме се налази само навигациона трака односно *Start*, један који служи за креирање игара интерактивне функције односно *Create Fiction*, један за креирање игара интерактивне функције путем корисничког промпта односно *GPT creation* и један који приказује листу свих креираних игара, дугме за учитавање кода селектоване игре, дугме за играње игре, дугме за креирање слика регија селектоване игре као и *checkbox* за избор играња игре са или без слика, *checkbox* за избор играња игре у бесконачном моду, *checkbox* за избор поновног учитавања стања пређашње игране игре односно *Library* (Слика 8.1.a). Значи *Start* прозор је исти као и прозор приказан на слици испод, само без икаквих графичких елемената на њему осим навигационе линије.



Слика 8.1.а Старт прозор

У овом фајлу односно *App* класи је обезбеђена логика приказивања и сакривања различитих садржаја главног прозора. Пример једне од тих логика се може видети у функцији *show_start_frame* Слика 8.1.б и функцији *_hide_all_frames* Слика 8.1.в.

```
def show_start_frame(self):
    self._hide_all_frames()
    self.start_frame.pack()
```

Слика 8.1.б Функција за приказ Start садржаја

```
def _hide_all_frames(self):
    for frame in [self.start_frame, self.fiction_frame, self.library_frame,
                  self.gpt_creation_frame, self.play_frame, self.picture_creator_frame]:
        frame.pack_forget()
```

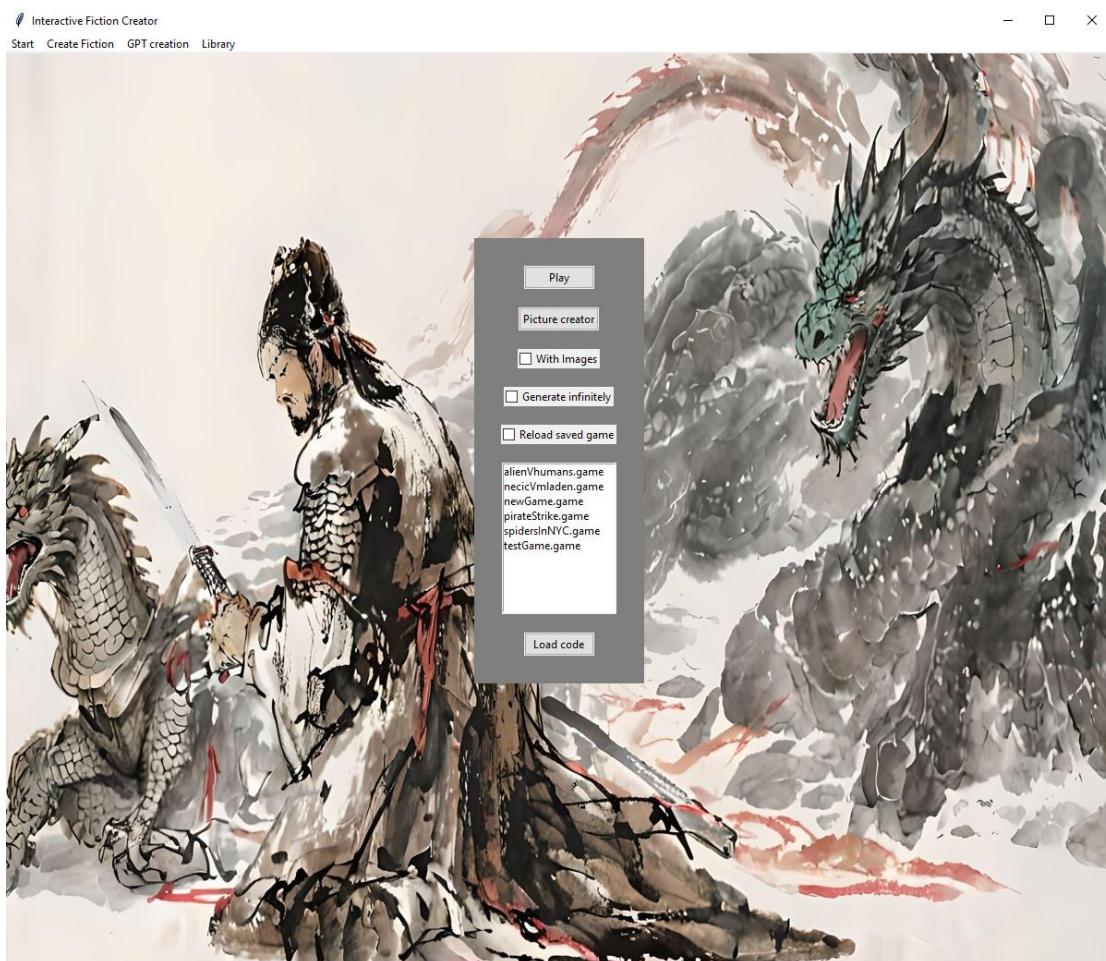
Слика 8.1.в Функција за приказ сакривање садржаја

Главни прозор ове апликације је *Library* прозор Слика 8.1.г. Као што је горе наведено у тексту у овом прозору постоји преглед свих игара, игре могу да се покрену, до креатора слика за игре се долази из овог прозора, код игре се може погледати из овог прозора.

Кликом на дугме *Play* главног прозора покреће се селектована игра за играње. Детаљније о овоме ћемо причати у поглављу *gameFrame*.

Кликом на дугме *Picture creator* главног прозора приказује се прозор креатора за слике изабране игре. Детаљније о овоме ћемо причати у *pictureCreatorFrame* поглављу.

Кликом на дугме *Load code* главног прозора отвара се прозор *Code editor*-а са кодом слектоване игре. Детаљније о овоме ћемо причати у следећем поглављу.

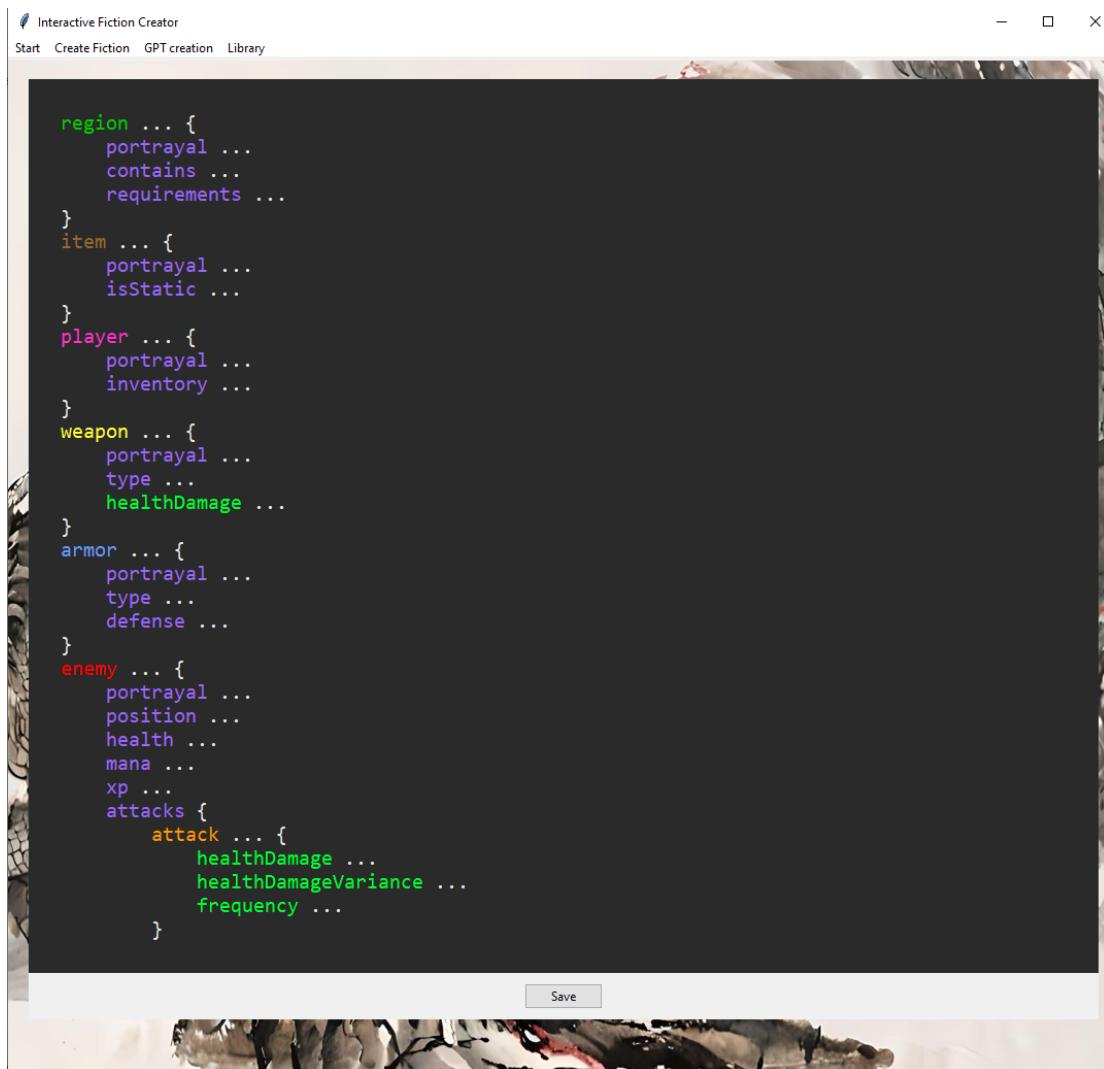


Слика 8.1.г *Library* прозор

Поред наведених дугмића постоје и *checkbox*-ови за играње у моду са сликама, у моду бесконачног играња и *checkbox* за учитавање пређашње сачуваног стања игре.

8.2 codeEditorFrame.py

У овом фајлу је креирана *Python* класа *CodeEditorFrame* која је задужена за писање кода игре интерактивне фикције као и за учитавање кодова већ креираних игара Слика 8.2.a.



Слика 8.2.а Прозор CodeEditorFrame-a

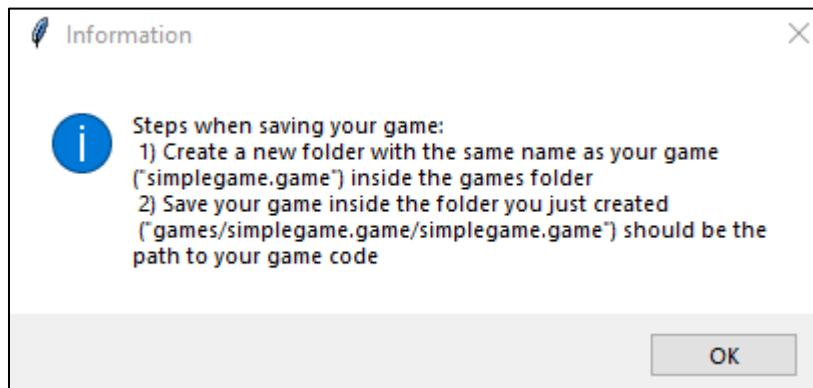
У апликацији је направљен једноставан едитор за писање кода који пружа кориснику неколико корисних функција за рад са кодом. Овај едитор није само обично текстуално поље за унос кода, већ садржи и неколико напреднијих функција које побољшавају корисничко искуство:

- **Маркер синтаксе (енг. *Syntax Highlighter*):** Едитор укључује функцију која препознаје кључне речи језика и боји их различитим бојама. Ово олакшава кориснику да се снађе у коду и брже уочи важне делове кода, чиме се побољшава читљивост и уређивање.

- Осветљавање тренутне линије: Линија на којој се тренутно налази корисников курсор је осветљена, што помаже у праћењу кода који се пише. Осветљење линије се аутоматски помера када корисник користи стрелице на тастатури за навигацију кроз текст, чиме се олакшава рад на специфичним деловима кода.

Поред едитора за код, апликација садржи и дугме "Save" које омогућава чување ново-креираних игара интерактивне функције. Ово дугме је кључно за обезбеђивање да корисников код буде сачуван и доступан за даљи рад или играње.

Када корисник први пут уђе у секцију *Create Fiction* у апликацији, појављује се нотификација која га упућује на то како да сачува нове игре. Ова нотификација служи као водич за нове кориснике, помажући им да разумеју процес креирања и чувања игара Слика 8.2.6.



Слика 8.2.6 Нотификација при уласку у едитор за код

8.3 *pictureCreatorFrame.py*

Овај фајл се може поделити у две целине. Прва целина је посвећена *Dall-e*-у односно креирању слика регија интерактивног света. Друга целина је посвећена самом баратању креирањем слика, односно, у питању је класа *PictureCreatorFrame*. За долазак на станицу за креирање слика користимо дугме *Picture creator* које се налази на прозору *Library* Слика 8.1.2.

У првој целини имамо генерисање слика помоћу *OpenAI API*-ја, тачније помоћу *DALL-E* модела.

```

def generate_image(self):
    prompt = self.text_area.get("1.0", "end-1c").lower()

    response = client.images.generate(
        model="dall-e-2",
        prompt=prompt,
        size="256x256",
        n=4,
    )
    self.img0_fromPipe = Image.open(BytesIO(requests.get(response.data[0].url).content))
    self.img1_fromPipe = Image.open(BytesIO(requests.get(response.data[1].url).content))
    self.img2_fromPipe = Image.open(BytesIO(requests.get(response.data[2].url).content))
    self.img3_fromPipe = Image.open(BytesIO(requests.get(response.data[3].url).content))

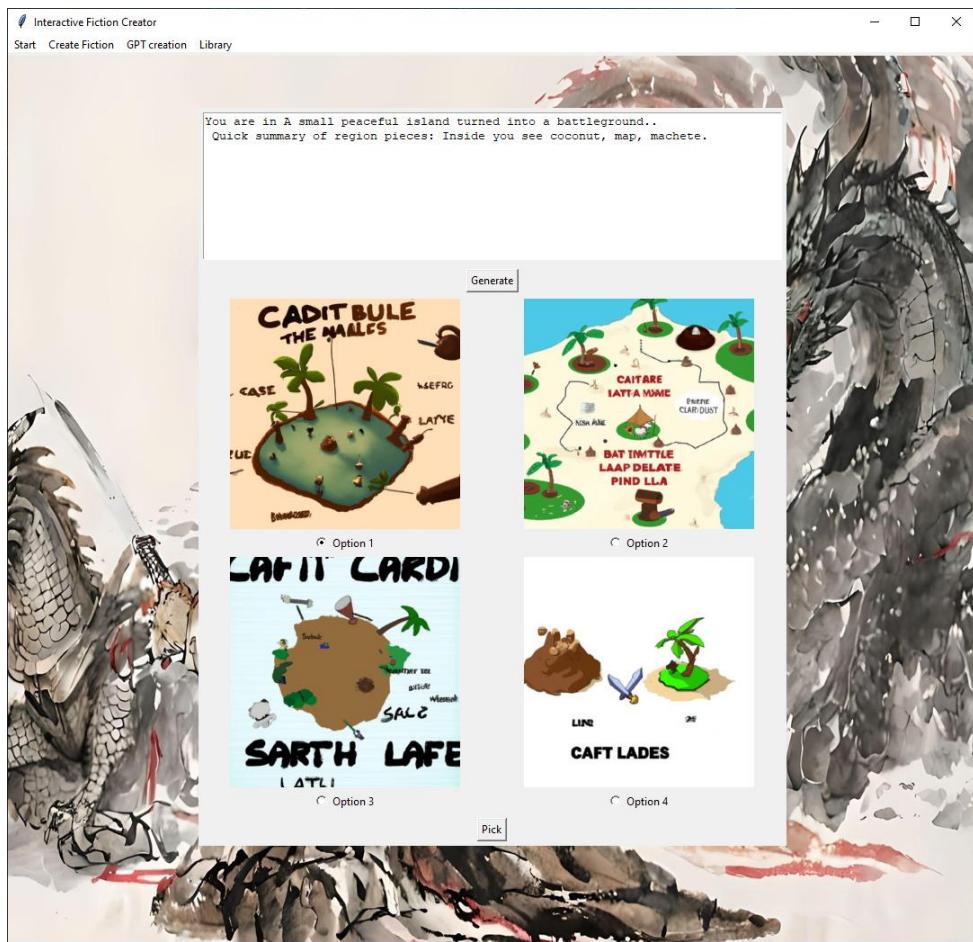
    self.update_image_labels()

```

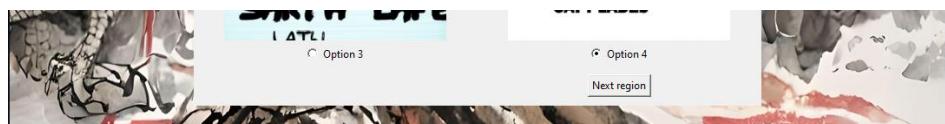
Слика 8.3.a *StableDiffusion* могући приступи

Друга целина апликације се састоји од логике за креирање слика за регије у игри и њиховог приказа у прозору. Приступањем прозору *pictureCreatorFrame* за изабрану игрицу, корисник добија једно дугме под називом *Next region*. Кликом на ово дугме започиње процес креирања слика за свет интерактивне фикције. Процес креирања слика функционише на следећи начин:

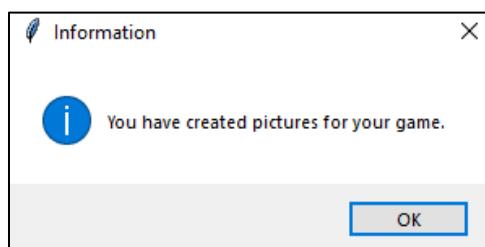
- Пролазак кроз регије: За сваку регију у свету игре, апликација приказује поље за унос текста које представља промпт за креирање слика. Ово поље за унос текста је иницијално попуњено основним описом регије и елементима који се налазе у њој. Овај опис служи као основа за креирање слика, али корисник може да га изменi по својој жељи ако жели да прилагоди слике специфичним захтевима.
- Генерирање слика: Испод поља за унос текста налази се дугме *Generate*. Кликом на ово дугме покреће се функција која генерише четири слике на основу унетог промпта. Генерисане слике се затим приказују у прозору, поред четири *Radio button*-a који омогућавају кориснику да изабере жељену слику Слика 8.3.б.
- Прескацање креирања слика: Уколико корисник не жели да креира слику за одређену регију, може прескочити овај корак кликом на дугме *Pick* на дну прозора. У том случају, за ту регију ће бити постављена предефинисана слика са натписом *No Image Available*.
- Финални избор слике: Након што корисник изабере жељену слику или одлучи да прескочи креирање слике, кликом на дугме *Pick* његов избор постаје финалан. Изабрана слика се чува у фолдеру игре. Дугме *Pick* се уклања са прозора, а уместо њега се појављује дугме *Next region*, које корисника пребације на следећу регију Слика 8.3.в.
- Завршетак процеса: Овај процес се понавља за све регије у игри. Након избора слике за последњу регију, кориснику се приказује обавештење да је успешно завршио процес креирања слика Слика 8.3.г.



Слика 8.3.б Прозор за избор слика са Pick дугметодом

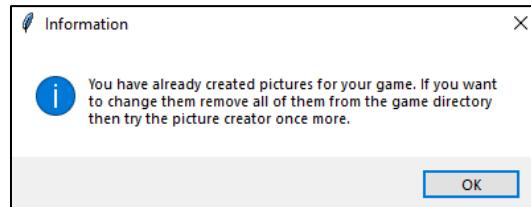


Слика 8.3.в Прозор за избор слика са Next region дугметодом



Слика 8.3.г Нотификација која означава успешан крај креирања слика

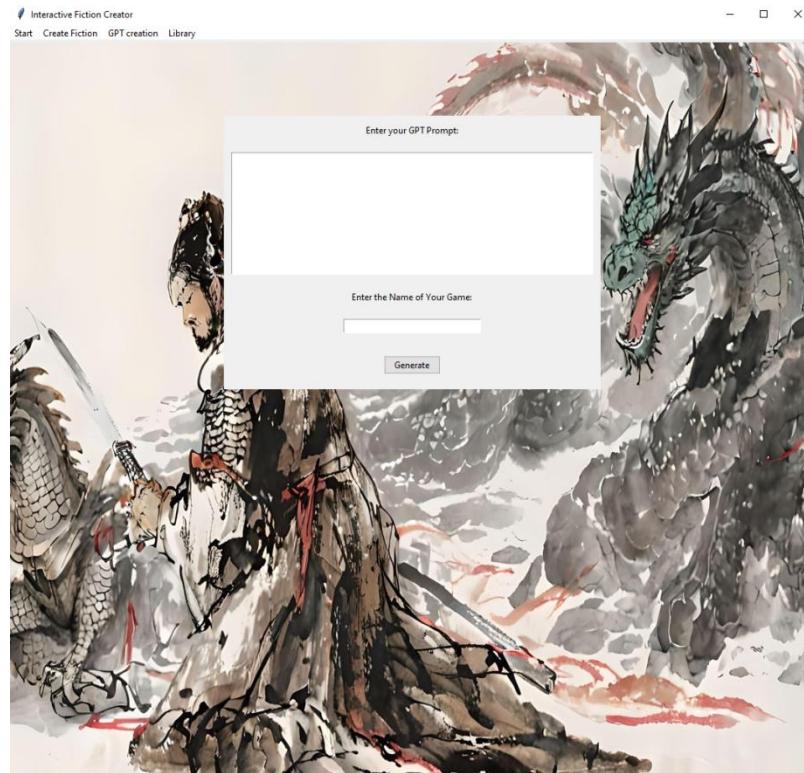
Уколико корисник жели да креира слике за игру за коју су слике већ креиране, апликација приказује нотификацију са обавештењем. Ова нотификација информише корисника о томе да су слике за ову игру већ креиране и да има могућност да их поновно креира ако жели. У обавештењу су наведени кораци које корисник треба да предузме уколико жели да замени постојеће слике новим. Ова функција омогућава кориснику да свесно донесе одлуку да ли жели да задржи постојеће слике или да креира нове, чиме се избегава случајно преклапање или губитак претходно креираних слика Слика 8.3.д.



Слика 8.3.д Нотификација која обавештава корисника да су слике већ креиране

8.4 *gptCreationFrame.py*.

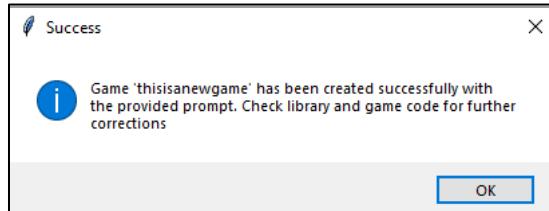
У овом фајлу долази до креирања игара интерактивне фикције помоћу GPT модела. Прозор се састоји из два текстуална дела Слика 8.4.а. Један велики за писање промпта за GPT модела, помоћу којег ће он да креира игру и једног мањег текстуалног поља за жељени назив креiranе игре.



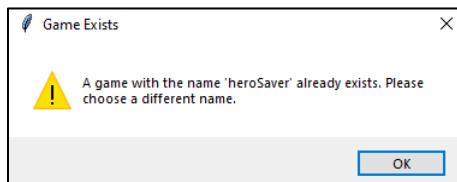
Слика 8.4.а Прозор за креирање игара помоћу GPT-а

Позадинска логика овог дела апликације јесте заправо само генерисање помоћу *GPT* модела које је објашњено у поглављу седам Слика 7.1.a.

Након успешног генерисаног кода игрице кориснику искаче нотификација о успешном креирању и о могућим даљим корацима Слика 8.4.b. Уколико већ постоји игра са истим називом кориснику искаче нотификација која га обавештава о томе Слика 8.4.v.



Слика 8.4.b Нотификација која обавештава корисника да је игрица изгениерисана

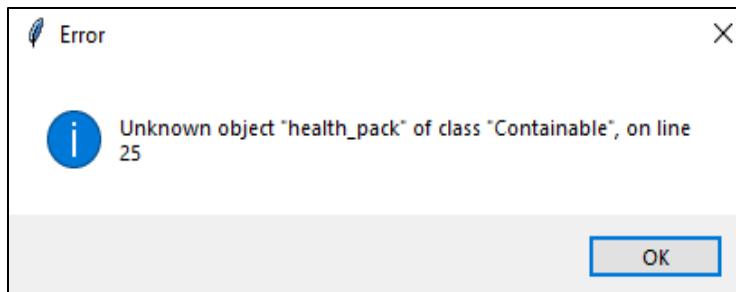


Слика 8.4.b Нотификација која обавештава корисника да већ постоји игра са тим именом

8.5 *gameFrame.py*

У овом фајлу се налази класа *GamePlayFrame* која је задужена за логику играња игре као и за њен сам приказ. За сам долазак на прозор *GamePlayFrame*-а користимо дугме *Play* које се налази на *Library* прозору Слика 8.1.a. Класи *GamePlayFrame* се такође прослеђују и вредности *checkbox*-ова *With Images*, *Generate infinitely* и *Reload saved game*. Прослеђене вредности служе да се одреди у ком моду играња ће корисник играти жељено изабрану игру. *With Images* представља мод играња са сликама, *Generate infinitely* представља играње у бесконачном моду и *Reload saved game* представља мод настављања играња пређашње сачуване игре, овај мод је направљен помоћу *Python*-ове библиотеке *Pickle* која чува цело објекат *GameWorld* у *.pickle* фајли као *byte stream*.. Ови модови се могу комбиновати тако да је искуство које играч може да осети разноврсно.

Приликом покретања игре врши се провера валидности кода, односно проверава се да ли жељена игра поштује све прописе граматике овог рада. Уколико граматика језика није испоштована корисник добија нотификацију са обавештењем који је проблем и где се у коду налази Слика 8.5.a.



Слика 8.5.а Нотификација која обавештава корисника да код игре није валидан

Када је код игрице валидан, корисник се премешта у прозор за играње саме игре. Овај прозор представља један кључни део апликације и омогућава интерактивно искуство играња. Састоји се од два основна текстуална поља и, у зависности од избора корисника, од приказа слике тренутне регије, као и једног дугмета “Save” на дну апликације које служи да корисник сачува тренутно стање игре.

- Главно текстуално поље: Ово велико текстуално поље представља главни извор информација за играча. У њему се приказују сви важни исписи као на пример: тренутна локација играча у игри, покушаје узимања објеката, интеракција са предметима у свету игре, напади непријатеља, помоћ и друге команде које играч уноси. Све информације које играч добија током игре приказују се у овом пољу, чинећи га централним местом за праћење напретка и дешавања у игри.
- Поље за унос команди: Испод главног текстуалног поља налази се мање текстуално поље које служи за унос команди од стране играча. Ово поље представља основни и једини начин интеракције играча са игром. Играч уноси команде као што су open, take, move и друге, чиме контролише свој напредак кроз игру и доноси одлуке.
- Приказ слике тренутне регије: Уколико је играч изабрао да игра игру са slikama, испод текстуалног поља за унос команди приказује се слика тренутне регије у којој се играч налази. Ова слика пружа визуелни контекст и помаже играчу да боље разуме окружење у којем се налази.
- Дугме “Save”: Ово дугме омогућава кориснику да сачува стање игре у било ком тренутку. Ово дугме покреће функцију `save_game_state` која прво сакупља све податке из тренутног играња, а затим те податке смешта у .pickle фајл Слика 8.5.б. Након успешног чувања кориснику искоче нотификација која га о томе и обавештава Слика 8.5.в.

```

def save_game_state(self, game_title, game_world):
    game_state = {
        'regions': game_world.regions,
        'items': game_world.items,
        'enemies': game_world.enemies,
        'weapons': game_world.weapons,
        'armors': game_world.armors,
        'player': game_world.player,
        'start_position': game_world.start_position,
        'final_position': game_world.final_position,
        'current_enemy': game_world.current_enemy,
        'prev_direction': game_world.prev_direction,
        'opposite_dirs': game_world.opposite_dirs,
        'settings': game_world.settings,
    }
    with open(game_title + ".pickle", 'wb') as file:
        pickle.dump(game_state, file)

    messagebox.showinfo("Information", "Game state saved!!")

```

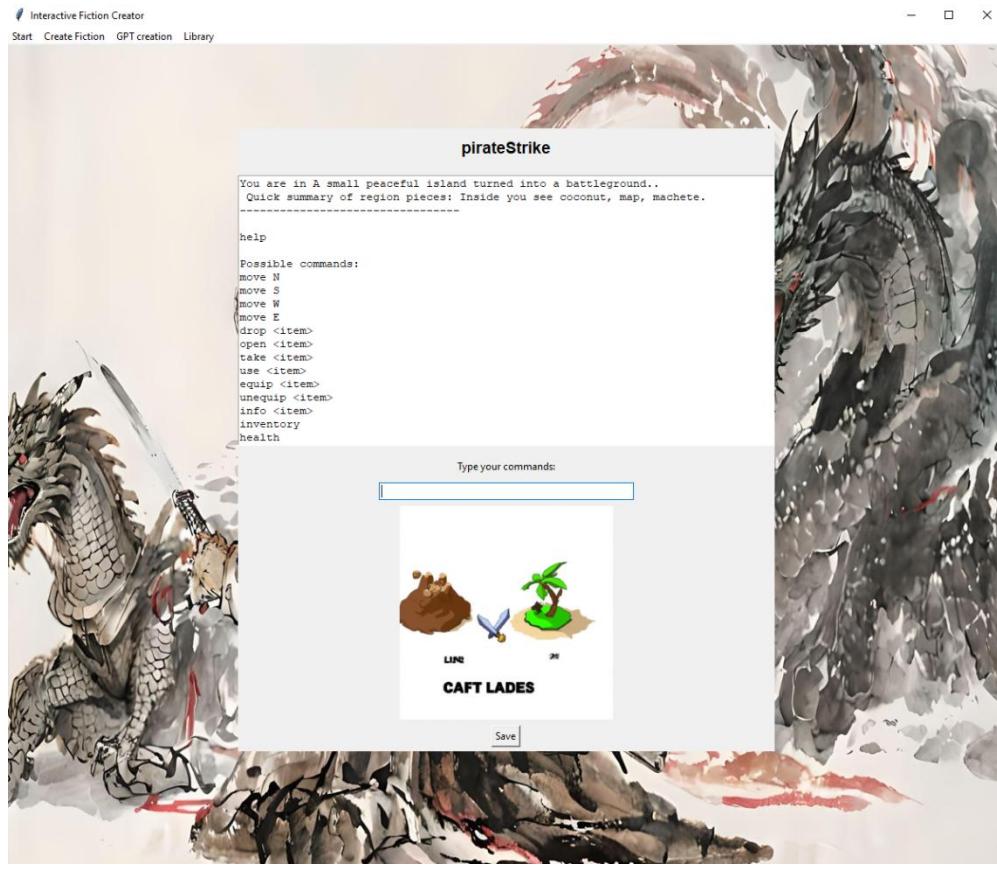
Слика 8.5.б Функција *save_game_state*



Слика 8.5.в Обавештење да је стање игре успешно сачувано

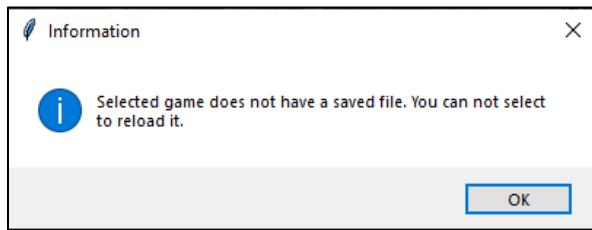
Приказ команде *help*, као и других команда и интеракције са светом игре, могу се видети на слици испод. Овај систем омогућава играчу да на интуитиван и једноставан начин управља игром и ужива у интерактивној прици Слика 8.5.г.

Команда *help* је заснована на листи могућих команда *possible_commands* која се може проширити у случају појављивања нових команда. Логика играња односно разумевања корисникова уноса односно његова интеракција са игром је заснована на читању корисниковог једноставног уноса који представља команду од једне или две речи. У класи *GamePlayFrame* се налази функција *process_user_input* која је задужена баш за то. Она процесира корисникove команде и позива одговарајуће функције на основу њих.



Слика 8.5.г Приказ прозора играња са *help* командом

У класи *GamePlayFrame* је обезбеђена сва логика иза играња игре. На почетку класе постоји део посвећен моду *Reload saved game*. У овом делу долази до провере да ли је тај мод укључем, ако јесте долази до учитавања фајла игрице са екstenзијом *.pickle* која се затим смешта у променљиву *gameWorld* која представља срж игре. Уколико не постоји сачувани фајл игре а корисник покрене игруцу са овим модом укљученим добија следећу нотификацију Слика 8.5.д.



Слика 8.5.д Обавештење о непостојању сачуване игре

Битно за напоменути јесте да приликом избора мода играња са сликама и бесконачног мода играња долази до потребе да се континуално генеришу слике за сваку

нову регију у коју авантуриста уђе. То је обезбеђено функцијом *infinitely_generate_pictures* која врши позиве ка *OpenAI API*-ју, прецисније ка *DALL-E* моделу Слика 8.5.ћ. *DALL-E* модел креира једну слику помоћу описа регије који му је прослеђен. Креирана слика се такође чува у фолдеру игре тако да након поновног улаза авантуристе у исту просторију неће доћи до креирања нове слике већ ће се искористити стара, већ креирана слика.

```
def infinitely_generate_pictures(self, region, game_title):
    prompt = self.gameWorld.region_string_for_image_creation(region)

    response = client.images.generate(
        model="dall-e-2",
        prompt=prompt,
        size="256x256",
        n=1,
    )
    games_directory = "games/" + game_title
    generated_image = Image.open(BytesIO(requests.get(response.data[0].url).content))
    generated_image.save(games_directory + '/' + region.name + '.png')

return generated_image
```

Слика 8.5.ћ Функција *infinitely_generate_pictures*

Након успешно пређене игре кориснику се брише текстуално поље игре и приказује му се *THE END* порука. Уколико је корисник играо са slikama онда му се приказује и црна слика са текстом *THE END*.

9. Закључак

Интерактивна фикција представља софтвер који симулира свет, односно регије у којима играч користи текстуалне команде како би управљао ликовима, кретао се кроз регије и интераговао са стварима које се у њима налазе. Интерактивна фикција је била битан корак на почетку развоја рачунарских игара. Иако је данас њен значај мањи у поређењу са ранијим периодом, ова врста игара наставља да се развија и да привлачи специфичну публику.

У овом раду су приказани алат и језик за развој интерактивне фикције. На почетку рада, представљена је историја интерактивне фикције, праћена постојећим решењима у овој области. Потом су детаљно описаны алати који су коришћени при имплементацији овог пројекта. Прво је представљен *textX* мета-језик, који је коришћен за креирање прилагођеног језика за описивање света интерактивне фикције. Затим је приказана *Python* графичка библиотека *Tkinter*, која је коришћена за креирање графичког корисничког интерфејса, као и *OpenAI API*, тачније његови *GPT* и *Dall-e* модели који су коришћени за генерисање назива и описа ентитета у бесконачном моду играња као и за генерисање слика на основу унетог текста.

Посебан акценат је стављен на мета-модел и интерпретер тог модела, који су кључни за функционисање система. Такође, посебан акценат је потребно ставити и на генерисање слика и описа помоћу великих језичких модела. На крају је детаљно описан графички кориснички интерфејс, који омогућава корисницима да креирају, визуализују и играју интерактивне фикције, праћен приказом процеса кроз које корисник пролази током коришћења апликације.

Закључак овог рада наглашава да језик који је коришћен за истраживање и израду овог пројекта представља напредни степен комплексности, али и да се додатно може проширити и унапредити. На пример, модели који су коришћени ће временом бити све бољи и напреднији, чиме ће се побољшати овај рад. Напреднијим моделима ће се омогућити бржа израда слика, као и маштовитији описи ентитета. Графички интерфејс, иако функционалан, такође пружа простора за даља побољшања, као што су унапређење корисничког искуства и додавање нових функција за лакшу интеракцију са игром.

Ове основне смернице представљају путоказ за буђење потенцијала овог рада. Будуће истраживање и развој могли би донети значајне иновације и унапређења у области коју овај рад обухвата. Креирање све напреднијих алата и метода за развој интерактивне фикције могло би да доведе до оживљавања овог жанра и привлачења нових генерација играча и програмера.

Литература

- [1] https://en.wikipedia.org/wiki/Interactive_fiction, прегледано 20. август 2024.
- [2] <https://www.sciencedirect.com/science/article/abs/pii/S0950705116304178>
Dejanović I, Vaderna R, Milosavljević G, Vuković Ž. Textx: a python tool for domain-specific languages implementation. Knowledge-based systems. 2017 Jan 1;115:1-4., прегледано 20. август 2024.
- [3] <https://www.mdpi.com/1999-5903/15/6/192>
Roumeliotis KI, Tselikas ND. Chatgpt and open-ai models: A preliminary review. Future Internet. 2023 May 26;15(6):192. , прегледано 20. август 2024.
- [4] <https://arxiv.org/pdf/2301.04655>
Gozalo-Brizuela R, Garrido-Merchan EC. ChatGPT is not all you need. A State of the Art Review of large Generative AI models. arXiv preprint arXiv:2301.04655. 2023 Jan 11. , прегледано 20. август 2024.
- [5] <https://dspace.mit.edu/bitstream/handle/1721.1/129076/Montfort-Riddle-Machines.pdf?sequence=2&isAllowed=y>
Montfort N. Riddle machines: The history and nature of interactive fiction. A companion to digital literary studies. 2013 May 17:267-82., прегледано 20. август 2024.
- [6] <https://literative.com/editorials/the-evolution-of-interactive-fiction/>, прегледано 20. август 2024.
- [7] <https://www.ifwiki.org/index.php/TADS>, прегледано 20. август 2024.
- [8] https://www.ifwiki.org/index.php/TADS_3, прегледано 20. август 2024.
- [9] <https://en.wikipedia.org/wiki/TADS>, прегледано 20. август 2024.
- [10] <https://inform-fiction.org/>, прегледано 20. август 2024.
- [11] <https://en.wikipedia.org/wiki/Inform>, прегледано 20. август 2024.
- [12] <https://eclipse.dev/Xtext/>, прегледано 20. август 2024.
- [13] <https://realpython.com/python-gui-tkinter/>, прегледано 20. август 2024.
- [14] <https://en.wikipedia.org/wiki/Tkinter#>, прегледано 20. август 2024.
- [15] <https://openai.com/index/openai-api/>, прегледано 20. август 2024.
- [16] <https://www.ieee-jas.net/en/article/doi/10.1109/JAS.2023.123618>, Wu T, He S, Liu J, Sun S, Liu K, Han QL, Tang Y. A brief overview of ChatGPT: The history, status quo

and potential future development. IEEE/CAA Journal of Automatica Sinica. 2023 May 1;10(5):1122-36., прегледано 20. август 2024.

[17] <https://www.mikecaptain.com/resources/pdf/GPT-1.pdf>, Radford A, Narasimhan K, Salimans T, Sutskever I. Improving language understanding by generative pre-training., прегледано 20. август 2024.

[18] <https://insightcivic.s3.us-east-1.amazonaws.com/language-models.pdf>, Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I. Language models are unsupervised multitask learners. OpenAI blog. 2019 Feb 24;1(8):9., прегледано 20. август 2024.

[19] <https://splab.sdu.edu.cn/GPT3.pdf>, Brown TB. Language models are few-shot learners. arXiv preprint ArXiv:2005.14165. 2020., прегледано 20. август 2024.

[20] <https://arxiv.org/pdf/2303.08774.pdf>, Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, Aleman FL, Almeida D, Altenschmidt J, Altman S, Anadkat S, Avila R. Gpt-4 technical report. arXiv preprint arXiv:2303.08774. 2023 Mar 15., прегледано 20. август 2024.

[21] <https://openai.com/index/dall-e/>, прегледано 20. август 2024.

[22]

https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=The+ethical+implications+of+DALL-E%3A+Opportunities+and+challenges&btnG=, Zhou KQ, Nabus H. The ethical implications of DALL-E: Opportunities and challenges. Mesopotamian Journal of Computer Science. 2023 Feb 5;2023:16-21., прегледано 20. август 2024.

[23] <https://arxiv.org/pdf/2204.13807.pdf>, Marcus G, Davis E, Aaronson S. A very preliminary analysis of DALL-E 2. arXiv preprint arXiv:2204.13807. 2022 Apr 25., прегледано 20. август 2024.

[24] <https://cdn.openai.com/papers/dall-e-3.pdf>, Betker J, Goh G, Jing L, Brooks T, Wang J, Li L, Ouyang L, Zhuang J, Lee J, Guo Y, Manassra W. Improving image generation with better captions. Computer Science. <https://cdn.openai.com/papers/dall-e-3.pdf>. 2023;2(3):8., прегледано 20. август 2024.

[25] <https://www.graphviz.org>, прегледано 20. август 2024.