Organizacja oraz prowadzenie gier fabularnych zdalnie z pomocą autorskiego bota na serwerze Discord

(Organizing and management of role-playing games remotly using custom made bot on Discord server)

Mateusz Zając

Praca inżynierska

Promotor: dr Marcin Młotkowski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

17 czerwca 2021

Streszczenie

Celem niniejszej pracy inżynierskiej była implementacja aplikacji *RPGamer* potocznie zwanej *botem*, mającej znacznie uprościć organizację i prowadzenie wydarzeń (w szczególności *Sesji RPG*), zrzeszających określoną grupę osób. Projekt działa na serwerze komunikatora Discord uczestników, a jego funkcje wywoływane są przy pomocy komend na czacie tekstowym. Bot jest aplikacją napisaną w sposób asynchroniczny, z użyciem biblioteki *discord*, korzystającą z biblioteki *SQLAlchemy*, która stosuje podejście ORM dla bazy danych *SQLite*. Projekt wykorzystuje istniejące mechanizmy komunikatora Discord, w celu implementacji nowych narzędzi przystępnych dla użytkowników. Zaimplementowano m.in rzucanie kostką przez gracza, podsumowywanie wyników rzutów po wydarzeniu, tworzenie ankiet dla graczy czy obsługę odtwarzania plików dźwiękowych z serwisu YouTube.

Abstract in english - It's going to be written after the whole paper is finished.

Spis treści

Wstęp

Społeczności nie zawsze są w stanie spotkać się w świecie rzeczywistym, co stawia je przed wyborem sposobu zdalnej konwersacji oraz wymiany informacji. Podczas gdy proste spotkania można szybko i łatwo zorganizować na dowolnym komunikatorze, tak rozgrywanie *Sesji RPG*[18] na odległość nie jest już do końca trywialne. Szybko okazuje się, że prosty komunikator głosowy łączący uczestników bardzo ogranicza możliwości oraz jest w stanie istotnie obniżyć immersję świata przedstawionego względem rzeczywistych rozgrywek. Brakuje programowych mechanizmów, które wspomagałyby podobne wydarzenia i wyręczały uczestników w powtarzalnych i żmudnych operacjach. Dostępne na rynku gotowe produkty często nie łączą wszystkich pożądanych funkcjonalności, a korzystanie z wielu rozwiązań jednocześnie w celu realizacji zamierzonych celów jest dla użytkowników uciążliwe.

Cel pracy

Celem pracy jest uproszczenie zarządzania użytkownikami podczas wydarzeń organizowanych zdalnie (w szczególności Sesji RPG). Niniejsza praca inżynierska jest rozwinięciem projektu realizowanego samodzielnie poza zajęciami, na użytek własny. Organizując takie wydarzenia z grupą osób, które były rozmieszczone w różnych miejscach Polski zauważyć można było liczne problemy, które trudno rozwiązać gotowymi aplikacjami. W naturalny sposób powstała myśl o stworzeniu własnego rozwiązania, znacznie upraszczającego zarządzanie oraz rozgrywanie sesji zdalnie. Wcześniejsza wersja projektu była używana w praktyce podczas gry przez użytkowników i była stale rozwijana w wyniku głosów i pomysłów graczy.

Zakres pracy

Projekt w swoich założeniach jest rozwiązaniem upraszczającym organizację i rozgrywanie Gier Fabularnych oraz im pokrewnych na serwerze Discord. Praca skupia się głównie na implementacji najbardziej istotnych z punktu widzenia organizacji rozwiązań:

1. Symulacja rzutu kostką przez gracza.

2. Zbieranie informacji na temat rzutów podczas pojedynczej sesji, w celu późniejszego podsumowania uśrednionych wyników.
3. Odtwarzanie muzyki i/lub dźwięków otoczenia, które zechce mistrz gry.
4. Zmiana pseudonimów określonych użytkowników w taki sposób, aby odpowiadały imionom postaci przez nich odgrywanych.
5. Mechanizm „drużyn", czyli grupowe zarządzanie użytkownikami będącymi członkami drużyny (np. wyciszanie całej drużyny lub przenoszenie na inny kanał na serwerze).
6. Rzuty kostką, będące tajemnicą dla wszystkich osób poza mistrzem gry.
7. Dźwiękowa reakcja bota na wyrzucenie bardzo korzystnego dla gracza wyniku na kostce lub bardzo niekorzystnego.
8. Prowadzenie kolejki odtwarzania na wzór typowych botów muzycznych.
9. Mechanizm sond/ankiet, które znacząco ułatwiają umówienie się na sesję RPG (wraz z podsumowaniem wyników).

Praca składa się ze wstępu, 5 głównych rozdziałów, spisu rysunków oraz bibliografii. W pierwszym rozdziale zostały zawarte informacje teoretyczne, wprowadzające w tematykę pracy. Dowiedzieć można się czym jest komunikator Discord, co rozumiemy przez pojęcie Sesji RPG, jakie biblioteki zostały użyte w celu realizacji projektu i co umożliwiają. Rozdział kolejny to opis instalacji, konfiguracji oraz uruchomienia. Poza zainstalowaniem odpowiednich zależności, należy dokonać m.in utworzenia profilu bota w bazie danych Discorda. Rozdział trzeci jest opisem implementacji komend, modułów oraz modelu bazy danych. Zawiera również przykłady użycia bardziej charakterystycznych komend bota oraz ich wyniki na czacie serwera. Przedostatni rozdział przedstawia i porównuje inne aplikacje i rozwiązania dostępne na rynku. Ukazane są zróżnicowane projekty, które w większym bądź mniejszym stopniu realizują założone przez niniejszą pracę inżynierską cele. Rozdział ostatni to podsumowanie całości pracy. Zamieszczone są w nim wnioski wyciągnięte po ukończeniu i w trakcie tworzenia projektu, analiza zrealizowanych celów oraz dalsze plany na rozwój. Opisane jest czego się nauczyłem i co było dla mnie zupełnie nowe i rozwijające podczas pracy nad aplikacją. Pozostałe rozdziały to wpis rysunków, listingi oraz bibliografia pracy. Jako załącznik dołączono wygenerowaną przy pomocy *doxygen*[6] dokumentację projektu, gdzie opisano sposób korzystania z dostępnych funkcji i modułów.

Rozdział 1.

Rys teoretyczny

1.1. Sesja RPG

Podstawą niniejszej pracy dyplomowej jest koncepcja Sesji RPG. Z języka angielskiego *Role-Playing Game*, czyli gra z odgrywaniem roli[1]. To wydarzenie, w którym uczestniczą gracze odgrywający postaci w świecie gry, Mistrz Gry (nazywany potocznie DM, GM lub MG[2]) będący narratorem historii i w szczególnych przypadkach obserwatorzy. Historia najczęściej jest prowadzona według wcześniej wybranego przez Mistrza Gry scenariusza, w określonym systemie RPG[3]. Odgrywanie postaci można porównać z powodzeniem do odgrywania roli przez aktora na scenie. To od uczestników w dużej mierze zależy jakość i sposób prowadzenia historii, dlatego bardzo istotnym jest, aby gracze „wczuli się" w swoją postać. Głównym ograniczeniem jest tylko wyobraźnia i kreatywność zarówno uczestników, jak i Mistrza Gry. Zapisany skrypt, system RPG oraz realia mają tylko naprowadzać graczy i stanowić wskazówkę w prowadzonej przygodzie. Sposób gry jest dowolny i nie ma narzuconych z góry scen do odegrania, co często powoduje wychodzenie poza przewidziane ramy scenariusza i wymusza na prowadzącym improwizację dalszych wydarzeń. Rozgrywka dąży do zrealizowania zaplanowanego na początku przygody celu, jednak to od uczestników zależy droga, którą podążą i sposób realizacji tego celu.

Przykładem systemu RPG może być ten oparty o twórczość pisarza H.P. Lovecrafta - *Call of Cthulhu*. Czas i miejsce akcji to zwykle lata 20. XX wieku w Ameryce, niedługo po wprowadzeniu prohibicji. Gracze wcielają się w role badaczy, zgłębiających tajemnice świata Wielkich Przedwiecznych. W trakcie swoich przygód rozwiązują zadania paranormalne oraz nadnaturalne i często tylko od ich działań zależą dalsze wydarzenia oraz losy postaci[4].

[1] Także *Gra Fabularna*, *Gra Wyobraźni* lub *Gra Narracyjna*

[2] ang. *Dungeon Master/Game Master* lub w polskim przekładzie *Mistrz Gry*

[3] System RPG określa zasady rozgrywki, atrybuty postaci i generalny sposób rozgrywania historii.

[4] Takie sesje dobrze oddają nagrania w serwisie YouTube, np. na kanale *Baniak Baniaka*[1]

1.2. Komunikator Discord

W celu lepszego zrozumienia sposobu działania aplikacji oraz problemów przez nią rozwiązywanych należy przyjrzeć się lepiej komunikatorowi *Discord*. Jest to jedno z najbardziej popularnych rozwiązań komunikacji przez internet, łączące w sobie zalety zarówno *Skype*[12] jak i *TeamSpeak*[5], poprawiając wady tych rozwiązań. Umożliwia porozumiewanie się za pomocą tekstu, głosu oraz wideo. Chętnie wybierany przez użytkowników ze względu na dostępność użycia, nieduże użycie pamięci oraz oferowane możliwości. Dowolny użytkownik może stworzyć własny serwer o rozbudowanej strukturze oraz systemie rang użytkowników wraz z uprawnieniami. Komunikator oferuje szereg rozwiązań dostępnych także w *MS Teams*[11] (jak np. reakcje na wiadomości użytkowników czy wklejanie grafik ze schowka). Administrator może ponadto dodać do serwera boty, które znacząco rozszerzają możliwości oraz scenariusze użytkowania. Istniejące na rynku rozwiązania oferują nawet wyszukiwanie informacji bezpośrednio przez czat komunikatora czy rozgrywanie nieskomplikowanych gier za pomocą interfejsów dostępnych w Discordzie. Dzięki wykorzystaniu prostych funkcjonalności wbudowanych w komunikator możemy stworzyć wiele nowych narzędzi dla użytkowników i dostosować serwer w taki sposób, aby spełniał nawet bardziej zaawansowane wymagania.

1.3. Biblioteka discord

Biblioteka *discord* to w rzeczywistości *API Wrapper* dla Discorda napisany w języku *Python*. Udostępnia szereg interfejsów dla programisty, dzięki czemu główny nacisk projektu jest położony na tworzeniu funkcji serwerowych i komend, zamiast na niskopoziomowym zarządzaniu sprzętem oraz połączeniami z serwerem. Biblioteka jest łatwa w użyciu, napisana w sposób asynchroniczny oraz posiada bogatą dokumentację[15] wraz z przykładami użycia metod[14]. Używana jest do tworzenia tzw. *botów*[17], automatyzujących dzięki jej użyciu funkcje udostępnione przez Discorda.

Bot działając na serwerze obserwuje zdarzenia, które mają na nim miejsce oraz swoją własną skrzynkę odbiorczą wiadomości. Jeśli wiadomość zaczyna się od określonego prefiksu lub na serwerze wystąpiło określone zdarzenie, program wykonuje kod odpowiedzialny za przetworzenie takiego zdarzenia lub komendy. Każda instancja bota biblioteki discord posiada własną pętlę zdarzeń (event loop). Jest ona oparta o pętlę zdarzeń biblioteki *asyncio* języka Python. To swojego rodzaju lista zadań do wykonania przez aplikację, po której program przełącza się, wykonując kod poszczególnych zadań. W momencie wywołania przez użytkownika komendy, tworzone jest nowe zadanie w pętli zdarzeń bota. W odpowiednim czasie zostanie ono przetworzone, a wynik zwrócony.

1.4. Programowanie asynchroniczne

Większość prostych aplikacji, które są napisane w celu wykonania konkretnego zadania lub zadań mogą być z powodzeniem napisane w sposób synchroniczny. Kod wykonuje się linijka po linijce, od początku do końca, krok po kroku. Problem może wystąpić, gdy jakaś część programu korzysta z czasochłonnych operacji, które znacznie opóźniają wykonanie innych części kodu. Przykładem mogą być tutaj operacje wejścia/wyjścia czy długie wyliczenia w algorytmach, z których korzysta program. W podejściu synchronicznym blokujemy działanie do czasu zakończenia obliczeń, potem możemy wznowić dalsze wykonanie. Pierwszym pomysłem na zaradzenie sobie z takimi problemami jest programowanie wielowątkowe. Operacje czasochłonne zlecamy oddzielnym wątkom, które system operacyjny budzi co jakiś czas, przerywając wykonanie aktualnego programu. Zapanowanie jednak nad takim kodem jest stosunkowo trudne, nie mamy także kontroli nad tym w którym miejscu kod zostanie przerwany.

Jeszcze innym pomysłem jest programowanie asynchroniczne. To programista decyduje kiedy dane zadanie może zostać zatrzymane (w Pythonie służy do tego słowo kluczowe `await`), a na jego miejsce może wejść inne. Zyskujemy wtedy dużą responsywność aplikacji, prostą koncepcję działania oraz dużo mniej potencjalnych błędów. Należy pamiętać, aby w metodach asynchronicznych nie używać blokujących funkcji[5], ponieważ odbiera to kontrolę pętli zdarzeń (kod staje się synchroniczny).

1.5. Biblioteka SQLAlchemy

Biblioteka programistyczna *SQLAlchemy*[2] służy do pracy z bazami danych typu SQL. Wspiera m.in. SQLite, MySQL, Microsoft SQL Server. Głównymi zaletami jest spamiętywanie wyników zapytań oraz śledzenie stanu utworzonych (lub pobranych do pamięci podręcznej) obiektów zmapowanych na tabele bazy danych. Dzięki niej skupiamy się bardziej na obiektach określonych klas, niż wierszach tabel bazy danych. SQLAlchemy chroni także program przed atakami typu *SQLInjection*[3], ze względu na stosowanie mechanizmu *Escape Characters*[19]. W większości przypadków korzystanie z takiej abstrakcji jest wystarczające do poprawnego i wydajnego działania. Możemy również pominąć warstwę *ORM*[20] i pisać „surowe" zapytania bezpośrednio do bazy danych. Wtedy niestety nie korzystamy z niektórych zalet biblioteki (np. spamiętywanie stanu obiektów), ponieważ omijamy całą warstwę *Object-relational mapping*. SQL Alchemy sam tworzy model bazy na podstawie wskazanych modeli oraz połączeń pomiędzy nimi. Nie musimy (a nawet nie powinniśmy) tworzyć modelu sami, ponieważ może to skutkować błędami w działaniu biblioteki. Do testów aplikacji możemy używać bazy danych zapisywanej lokalnie

[5] zatrzymujących program do czasu wykonania zadania

w pliku na dysku (*sqlite*[7]). W dokumentach biblioteki oprócz opisu logiki poszczególnych obiektów i funkcjonalności zostały zamieszczone również stosowne do omawianego zagadnienia przykłady. Podążając za tymi wskazówkami można nauczyć się korzystania z niej od podstaw i zaimplementować ją bez większych przeszkód w danym projekcie.

1.6. Biblioteka youtube_dl

W celu komunikacji za pomocą API YouTube RPGamer używa biblioteki *youtube_dl*[4], która jest jednocześnie podstawą działania modułu odtwarzacza muzycznego bota. Przy jej udziale możemy pobierać filmy, dźwięk oraz dane zamieszczonego materiału w formie typowego słownika z Pythona. Odtwarzanie transmisji na żywo również nie stanowi problemu. Umożliwia pobieranie wyników wyszukiwania pojedynczych filmów czy całych list odtwarzania. Otrzymując dzięki bibliotece bezpośredni link do materiału możemy go dalej przekazać do *FFMpeg*[16] w celu strumieniowania dźwięku na kanale aplikacji Discord.

Rozdział 2.

Instalacja projektu

Aby przystąpić do uruchomienia projektu, musimy najpierw utworzyć konto użytkownika (lub zalogować się na już istniejące), do którego będzie przypisana aplikacja oraz serwer, na którym bot ma działać. Z uwagi na charakter aplikacji, oprócz zainstalowania niezbędnych zależności musimy także odpowiednio skonfigurować pliki bota, jak również utworzyć instancję aplikacji w bazie Discorda.

2.1. Przygotowanie aplikacji

Tworzymy aplikację w bazie Discorda przy użyciu załączonego poradnika[8]. Ponadto, do poprawnego działania niektórych funkcji należy włączyć *Privileged Gateway Intents*. W tym celu należy przejść do zakładki *Bot* i utworzyć nowego bota. Na nowo wygenerowanej stronie wystarczy włączyć *Presence Intent* oraz *Server Members Intent*, a następnie zapisać zmiany. Tak przygotowaną aplikację możemy dodać do własnego serwera Discord[9][8]. Po wykonaniu tych działań bot powinien pojawić się na liście użytkowników, ze statusem *Offline*.

2.2. Środowisko uruchomieniowe

Wszystkie zależności niezbędne do uruchomienia aplikacji znajdują się w folderze venv (łącznie z interpreterem języka python). Otwierając projekt przy pomocy *JetBrains PyCharm*[10] możemy skorzystać z wcześniej skonfigurowanego środowiska wirtualnego, bez potrzeby instalacji dodatkowych bibliotek. Jeśli natomiast uruchamiamy z linii poleceń, za pomocą czystego interpretera języka Python, potrzebujemy zależności z Tabeli 2.1.

Przed uruchomieniem kodu należy również dostosować ustawienia w pliku *config.json*, w folderze projektu. W miejscu *botToken* należy wpisać swój token z poradnika[8]. Pole *ownerId* powinno zostać uzupełnione o ID użytkownika, który ma

Nazwa	**Wersja**	**Nazwa**	**Wersja**
PyNaCl	1.4.0	idna	3.1
SQLAlchemy	1.4.15	multidict	5.1.0
aiohttp	3.7.4.post0	mutagen	1.45.1
async-timeout	3.0.1	pip	21.1.2
attrs	21.2.0	pycparser	2.20
cffi	1.14.5	setuptools	57.0.0
chardet	4.0.0	six	1.16.0
discord	1.0.1	typing-extensions	3.10.0.0
discord.py[voice]	1.7.2	yarl	1.6.3
ffmpeg	1.4	youtube-dl	2021.4.7
greenlet	1.1.0		

Tablica 2.1: Zależności niezbędne do uruchomienia projektu

być administratorem bota. W celu włączenia wyświetlania ID użytkownika należy:

1. Wejść w ustawienia konta w aplikacji Discord
2. Przejść do sekcji *Zaawansowane*
3. Włączyć *Tryb developera* oraz zapisać zmiany

Teraz wystarczy nacisnąć prawym przyciskiem na swój profil na liście użytkowników serwera i wybrać *Kopiuj ID*. Tak uzupełniony plik konfiguracyjny można zapisać i zamknąć, a następnie włączyć aplikację. Jeśli projekt załaduje się poprawnie, na konsoli wyświetli się ciąg znaków: *Application has started properly*. W celu wyświetlenia panelu pomocy, wystarczy wpisać na czacie komendę `‘help`.

Rozdział 3.

Implementacja

Aplikacja jest podzielona na kilka różnych modułów, które zawierają w sobie nawiązujące do nich funkcjonalności. Dzięki użyciu mechanizmu *Cogs*[13] biblioteki discord możemy tymi modułami dowolnie zarządzać w trakcie wykonania programu (podłączać oraz odłączać je od bota podczas działania). Wszystkie dostępne w projekcie moduły są składowane w folderze *cogs* projektu. Każdą z dostępnych komend poprzedzić należy znakiem „`" (klawisz tyldy) jeśli prefix nie został zmieniony w pliku *config.cfg*.

3.1. Struktura modułu

Każdy oddzielny moduł to nowa klasa, dziedzicząca po klasie `Cog` z pakietu `discord.ext.commands`. Aby zdefiniować nowy moduł (np. o nazwie `Example`) wystarczy utworzyć klasę taką jak na Listingu 3.1.

```
1 #exampleCog.py
2 from discord.ext import commands
3
4 class Example(commands.Cog):
5     def __init__(self, client):
6         self.client = client
7
8 def setup(client):
9     client.add_cog(Example(client))
```

Listing 3.1: Przykładowy kod modułu `Example`

Tak utworzony moduł wystarczy załadować za pomocą metody `load_extension` obiektu `Bot`[1] lub komendą `‘load <nameOfModule>` (użytkownik musi być właścicielem[2] bota).

[1] Przykład można znaleźć w pliku `main.py`, gdzie w pętli są ładowane wszystkie moduły projektu z katalogu *cogs*

[2] Z ang. *owner*.

3.2. Struktura komendy

Definiowanie komend bota nie różni się znacznie od definiowania zwykłych metod asynchronicznych. Należy jedynie dodać parametr klasy `discord.ext.commands.context` zaraz po parametrze `self` oraz dodać dekorator `@commands.command()` nad nagłówkiem funkcji. Przykładową komendę wypisującą na czacie wiadomość „Hello world!" zamieszczono w Listingu 3.2

```
1  # exampleCommand.py
2  @commands.command()
3  async def helloworld(self, ctx: commands.context):
4      await ctx.send("Hello, world!")
```

Listing 3.2: Przykładowa komenda wyświetlająca Hello World!

Możemy także użyć innych dekoratorów, np. aby umożliwić wykonanie komendy tylko określonym użytkownikom[3] czy zablokować możliwość zbyt częstego wykonania danej akcji przez pojedynczego użytkownika[4].

3.3. Baza danych

Dzięki SQL Alchemy aplikacja korzysta z bazy danych działającej na silniku SQLite przy użyciu abstrakcji Object-relational mapping. W celach implementacyjnych powstały klasy, modelujące poszczególne tabele oraz relacje, które pomiędzy nimi zachodzą. Każdy z tych modeli jest zawarty w katalogu *Models* i ma prefiks *Model*.

Mimo, iż korzystamy z abstrakcji obiektów i klas, nadal musimy pamiętać o relacjach, które zachodzą między obiektami. Przyjrzyjmy się relacji wiele-do-wielu tabel `UserNameBackup` oraz `ServerSession`, której tabelą pośrednią jest `UserName`. Taką relację możemy modelować łącząc dwie relacje (*one-to-many* oraz *many-to-one*) (przykład na Listingu 3.3).

```
1  #serverSession.py
2  from sqlalchemy import Column, Integer, String, Boolean
3  from sqlalchemy.orm import relationship
4  from DBManager import dbmanager
5
6  class ServerSessionModel(dbmanager.Base):
7      __tablename__ = "ServerSession"
8
9      ID_ServerSession = Column(Integer,
10                               primary_key=True,
11                               autoincrement=True)
12     ID_Server = Column(Integer)
13     SessionShort = Column(String)
```

[3]Taki dekorator jest użyty np. nad funkcją `load` w pliku main.py

[4]Przykład użycia takich dekoratorów zamieszczony w module `Session`

```
14      ID_GM = Column(Integer)
15      SoundBoardSwitch = Column(Boolean)
16
17      UsersNames = relationship("UserNameModel",
18                                cascade="all,delete,delete-orphan")
```

Listing 3.3: Wycinek implementacji serverSession.py (tabela po stronie *one*)

Jak widzimy powyżej, modele dziedziczą po `dbmanager.Base` (musi to być ten sam obiekt, który „budujemy" w pliku `main.py`[5]). Większość pól klasy to obiekty `Column`, z odpowiednim typem podanym w argumencie. Po stronie relacji *one* należy utworzyć atrybut (obiekt klasy `relationship`), będący listą obiektów w relacji. Po stronie *many* dodajemy natomiast pole, będące obiektem klasy `Column`, jednak posiadające w argumencie (oprócz typu) obiekt klasy `ForeignKey`. Biblioteka zbuduje cały model bazy, jeśli ten nie istnieje.

```
1  #username.py
2  from sqlalchemy import Column, Integer, String, ForeignKey
3  from DBManager import dbmanager
4
5  class UserNameModel(dbmanager.Base):
6      __tablename__ = "UserName"
7
8      ID_UserName =
9              Column(Integer, primary_key=True)
10     ID_ServerSession =
11             Column(Integer,
12                     ForeignKey('ServerSession.ID_ServerSession'))
13     HeroName = Column(String)
14     ID_User =
15             Column(Integer,
16                     ForeignKey('UserNameBackup.ID_User'))
```

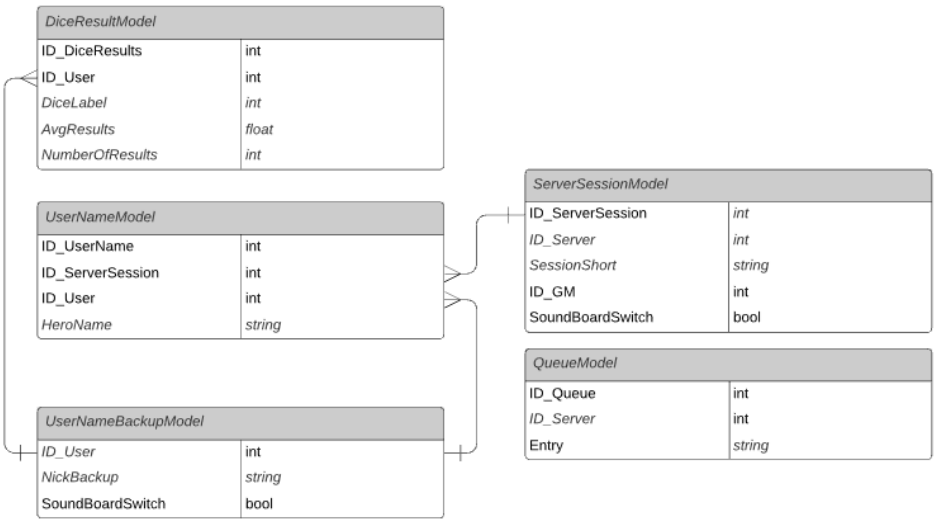
Listing 3.4: Wycinek implementacji username.py (tabela po stronie *many*)

3.4. Przykłady mechanizmów aplikacji

3.4.1. Głosowanie na termin

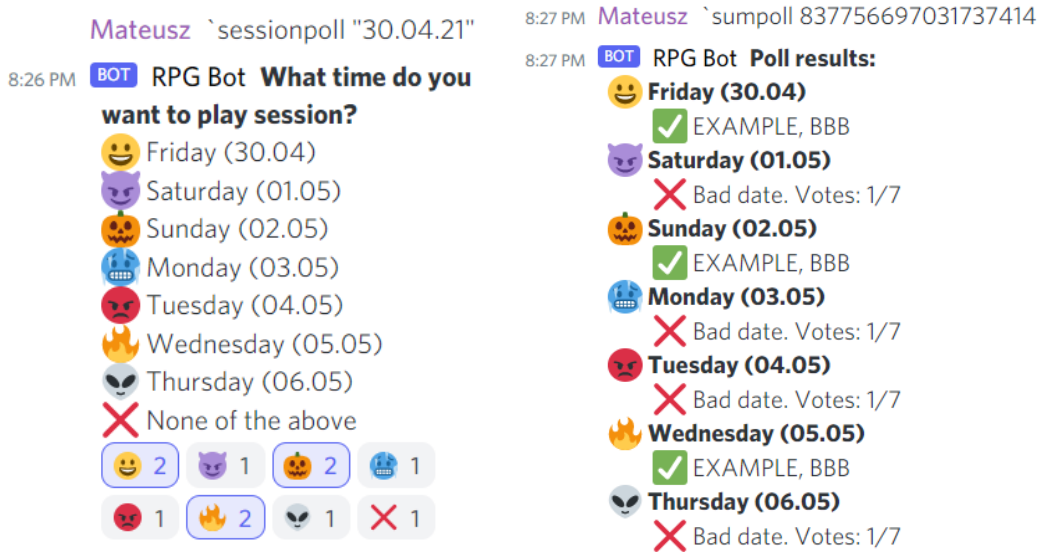
Jednym z większych problemów w organizacji zarówno sesji RPG jak i innych wydarzeń w większej grupie osób jest znalezienie terminu. Jeśli organizujemy spotkanie w małej liczbie osób lub nie spotykamy się za często, ręczne sprawdzenie jest rozsądnym podejściem. Sytuacja staje się tym bardziej skomplikowana, im więcej mamy typów wydarzeń i osób do sprawdzenia. Pomysłem, który jest prosty w swoich założeniach jest utworzenie ankiety, w której uczestnicy będą głosować przy pomocy emotikon Discorda. Poźniej aplikacja użyje wyników, aby podsumować ankietę

[5] Budowanie odbywa się w linijce `DBManager.dbmanager.Base.metadata.create_all(bind=DBManager.dbmanager.engine)`



Rysunek 3.1: Model bazy danych całej aplikacji

i sprawdzić kiedy można zorganizować dane wydarzenie. Przykład użycia mechanizmu został zamieszczony na Rysunku 3.2.



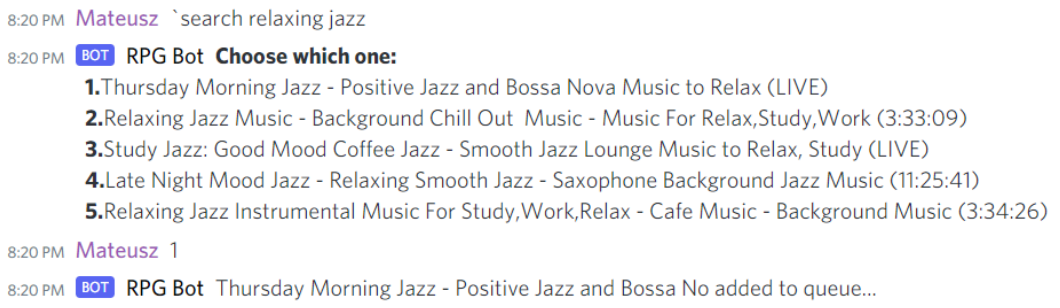
(a) Ankieta oraz osiem reakcji (b) Podsumowanie ankiety

Rysunek 3.2: Przykład użycia mechanizmu ankiet

3.4.2. Strumieniowanie muzyki z serwisu YouTube

Niekiedy podczas spotkania organizator chciałby odtworzyć dźwięk lub utwór muzyczny, który będzie słyszany przez wszystkich uczestników na kanale. Istnieją rozwiązania synchronizujące odtwarzacze użytkowników, jednak wymagają użycia

dodatkowych mechanizmów poza Discordem[6]. Opisywana aplikacja jest w stanie odtwarzać dźwięk z serwisu YouTube na żywo na kanale głosowym (bez uprzedniego pobierania utworu), ale także umożliwia wygodne wyszukiwanie utworów poprzez czat. Przykład zastosowania mechanizmu zamieszczono na Rysunku 3.3.



Rysunek 3.3: Wyszukiwanie utworów oraz wybór

3.4.3. Podsumowanie rzutów kostką po wydarzeniu

Aplikacja zapisuje wyniki rzutów kostką użytkowników, którzy należą do co najmniej jednej sesji. Sesja to tylko zbiór użytkowników wraz z podstawowymi informacjami o nich. System nie śledzi tego w jakiej sesji padły jakie rzuty (nie ma to sensu, ponieważ dany użytkownik nie gra więcej niż jednej sesji w danej jednostce czasu). Podsumowując rzuty korzystamy z tej struktury, aby wiedzieć jakie rzuty uwzględnić w podsumowaniu. Aby usunąć dane o rzutach, należy albo wywołać odpowiednią komendę (`‘resetrolls <nameOfSession>`), lub skorzystać z komendy przywracania pseudonimów graczy po sesji (`‘changenicks <nameOfSession: str> True`), która także usuwa informacje o rzutach.

Zdefiniujmy sesję o nazwie *Sesja_Waterfall*, która zawiera w sobie gracza *Mateusz* (`‘makesession Sesja_Waterfall 0 @Mateusz`)[7]. Po rzuceniu kilka razy kostką d100 oraz d5 (oznaczenie po literze d to liczba ścian kostki)[8] możemy zobaczyć średnią tych rzutów (Rysunek 3.4).

Aplikacja oblicza nową średnią na podstawie wzoru i zapisuje ją z powrotem do bazy danych. Dzięki temu zużywamy minimalną ilość pamięci do spamiętywania wyników, a ilość możliwych rzutów zanim licznik się „przekręci” to $2^{63} - 1$ (maksymalna wartość pola `INTEGER` w SQLite). W praktyce bardzo trudno będzie uzyskać taką wartość podczas zwykłego użytkowania, więc jesteśmy w stanie śledzić wyniki

[6]Przykładem takiego rozwiązania jest Watch2Gether (`https://w2g.tv`), gdzie możemy w zsynchronizowany sposób odtwarzać filmy np. z YouTube.

[7]Konstrukcja komendy tworzenia sesji to: `‘makesession <nameOfSession: str> <soundboard: bool> <*members: discord.Member>`, gdzie `nameOfSession` to nazwa sesji, `soundboard` to przełącznik reakcji dźwiękowych bota na rzuty, `members` to lista uczestników sesji.

[8]W sesjach rozgrywanych na żywo 100-ścienna kostka d100 jest zastępowana przez dwie kostki. Jedna z cyfrą dziesiątek, druga z cyfrą jedności.

(a) Kilka rzutów kostką d100 oraz d5

(b) Średni wynik po rzutach kostką

Rysunek 3.4: Przykładowa średnia rzutów kostką

naprawdę długich i intensywnych sesji. Przy dużej ilości rzutów coraz wyraźniej będziemy widzieć jednak zbliżanie się obliczeń do wartości oczekiwanej danej kostki i obliczanie średniej nie będzie miało sensu.

$$\text{newAverage} = \frac{\text{oldAverage} * \text{oldNumberOfResults} + \text{sumOfCurrentRolls}}{\text{oldNumberOfResults} + \text{numberOfCurrentRolls}}$$

Rozdział 4.

Inne rozwiązania

Z opisanych tutaj rozwiązań korzysta wiele użytkowników na całym świecie. Są ogólnodostępne, a ich bezpłatna wersja umożliwia podstawową realizację określonych funkcji.

4.1. YAGPDB.xyz - bot

Bot YAGPDB.xyz posiada funkcję rzutu kostką N-ścienną K razy dzięki komendzie `-roll`. Można również zdefiniować ile rzutów ma zostać wykonanych przez bota (-roll KdN). Udostępnia również komendę do tworzenia prostych ankiet. Nie zapisuje niestety średnich wartości rzutów wykonanych przez każdego użytkownika oraz nie udostępnia dodatkowych mechanizmów typowo pod sesję RPG. Ankiety są bardzo prostym mechanizmem, nie ma możliwości sprawdzenia pasującego terminu dla określonej sesji. `https://yagpdb.xyz/`

4.2. Rhytm - bot

Rhytm to najbardziej popularny bot muzyczny, obsługujący zarówno muzykę z URL YouTube, jak i wyszukiwanie utworów poprzez czat Discorda. Prowadzi kolejkę odtwarzania oraz wszelkie mechanizmy niezbędne dla odtwarzacza muzycznego. Służy tylko odtwarzaniu muzyki, nie posiada innych mechanizmów związanych z sesjami RPG. Opcjonalnie może służyć jako drugi bot do odtwarzania dźwięków (boty mogą odtwarzać na raz tylko jedno źródło dźwięku). `https://rythm.fm/`

4.3. Roll20 - serwis internetowy

Serwis Roll20 jest stworzony z myślą o prowadzeniu oraz rozgrywaniu sesji RPG. W podstawowej, bezpłatnej wersji umożliwia rzuty kostką, prowadzenie kart po-

staci oraz dużo więcej. Jest jednak dodatkowym serwisem (jeśli użytkownik używa Discorda do porozumiewania się), który każdy z graczy musi regularnie sprawdzać i odpowiednio ustawiać. Nie ma również możliwości zagłosowania na termin organizowanej sesji. Wiele graczy zgłaszało, że lepiej im prowadzić kartę postaci w pliku pdf na komputerze lub na kartce papieru, niż na powyższym serwisie. `https://app.roll20.net`

4.4. Watch2Gether - serwis internetowy

Mówiąc o wspólnym słuchaniu muzyki czy podkładu dźwiękowego nie można pominąć jednego z najpopularniejszych rozwiązań przystosowanych do tego celu, czyli Watch2Gether. Rozwiązanie umożliwia odtwarzanie filmów z wielu popularnych takich jak YouTube czy Vimeo. Oferuje utworzenie pokoju dla słuchających oraz listy odtwarzania, do której każdy z uczestników może dodawać swoje propozycje. Może zostać użyty do wspólnego słuchania ścieżki dźwiękowej, synchronizując odtwarzacze użytkowników. Niestety na tym kończy się funkcjonalność przydatna w realizacji rozgrywek. Nie uświadczymy żadnego wsparcia dla rzutów kostką czy też zarządzaniem użytkownikami. Poza tym, w wersji mobilnej strona musi być cały czas aktywna, aby odtwarzać dźwięk. Przełączenie się na inną aplikację automatycznie wyłącza odtwarzany film. To uniemożliwia rozgrywkę w jakikolwiek sposób angażując tylko jedno urządzenie.

4.5. Foundry Virtual Tabletop - serwis internetowy

Foundry Virtual Tabletop jest rozwiązaniem z wyglądu bardzo przypominającym serwis Roll20. Wymaga jednak uiszczenia jednorazowej opłaty na dostęp do serwera przez Mistrza Gry (gracze łączą się za darmo) w wysokości około 61.50 USD. Rozwiązanie bardzo zaawansowane, posiadające gotowe grafiki, mapy czy karty postaci nawiązujące do najpopularniejszych systemów RPG. Niestety potrzeba dłuższego czasu na zapoznanie się z interfejsem oraz możliwościami, co dla nowych i mało doświadczonych graczy będzie stanowiło sporą przeszkodę. Wersja demonstracyjna zamieszczona na stronie produktu nie pozwala na dużo, przez co trudno w pełni ocenić praktyczność i jakość tego projektu. `https://foundryvtt.com/`

Rozdział 5.

Podsumowanie

Istotą niniejszej pracy inżynierskiej była implementacja mechanizmów znacznie upraszczających rozgrywanie sesji gier fabularnych, ale też organizację spotkań w dużej grupie osób. Było to możliwe dzięki zastosowaniu popularnego i przystępnego komunikatora Discord, biblioteki discord dla języka Python korzystającej z podejścia asynchronicznego, jak również SQLAlchemy, będącego systemem ORM bazy danych SQLite. W toku trwania prac nad projektem wytworzono sprawnie działający system, którego użycie nie powinno sprawiać problemów po krótkim zapoznaniu. Każda funkcjonalność jest odpowiednio opisana w kodzie źródłowym, ale również w komendzie `‘help`, dostępnej publicznie dla użytkownika. przypadku wystąpienia nieoczekiwanego przez autora programu błędu, bot nie przestaje działać, ale wypisuje stosowną informację na czacie, oraz w logach. Aplikacja może zostać użyta również przez organizatorów wydarzeń niezwiązanych tematycznie z grami RPG. Mechanizm zarządzania użytkownikami na kanale czy tworzenie ankiet służących wyborowi konkretnego terminu spotkania to przykłady funkcjonalności o dużo szerszym zastosowaniu. Zapotrzebowanie na taką aplikację powstało głównie w wyniku głosów użytkowników rozgrywających gry fabularne, ale także braku dostatecznie funkcjonalnych rozwiązań na rynku.

5.1. Analiza realizacji celów pracy

Wytworzone rozwiązanie spełnia założone na początku pracy cele oraz pokrywa przewidziany zakres pracy (wstęp dokumentu):

1. Komenda `‘roll` losuje liczbę z wybranego przez użytkownika przedziału w sposób losowy, wykorzystując funkcję z biblioteki `random` języka Python/

2. Moduł `Rolls` zbiera informacje o rzutach graczy oraz umożliwia podsumowanie wyników.

3. Moduł `Soundboard` służy odtwarzaniu dźwięków oraz muzyki (również na żywo z serwisu YouTube, bez uprzedniego pobierania całości).

4. Komenda `‘changenicks` zmienia pseudonimy graczy na odpowiednie dla danej sesji.

5. Moduł `Session` realizuje funkcjonalności drużyn graczy. Wyciszanie oraz przenoszenie użytkowników na inne kanały jest również możliwe.

6. Rzuty kostką realizowane przez moduł `Rolls` posiada możliwość przypisania użytkownika (dodatkowy argument), który otrzyma informację o rzucie.

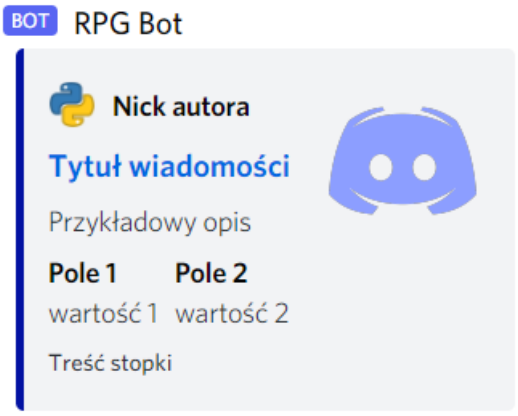
7. Dźwiękowa reakcja bota na korzystny lub niekorzystny wynik na kostce został zaimplementowany w module `Rolls`. Możliwość wyłączenia tej funkcjonalności została odzwierciedlona w odpowiednim argumencie komendy tworzenia sesji.

8. Kolejka odtwarzania dla danego serwera jest przechowywana w bazie danych i aktualizowana na bieżąco przy zmianie utworu oraz modyfikacji kolejki.

9. Komenda `‘sessionpoll` umożliwia tworzenie ankiet dla użytkowników, wraz z komendą `‘sumpoll`, podsumowującą wyniki ankiety.

5.2. Dalsze plany na rozwój aplikacji

Kolejnym celem na rozwój jest przede wszystkim zastosowanie obiektów typu `Embed` zamiast zwykłych wiadomości na czacie. Takie prezentowanie treści jest dużo lepsze oraz czytelniejsze w odbiorze dla użytkownika. Ilość pól oraz ich rozmieszczenie w wiadomości jest dowolne. Przykład takiego obiektu wysłanego na czacie zamieszczono na Rysunku 5.1.



Rysunek 5.1: Przykład wiadomości typu `Embed`

W celu wydajnego działania na większej ilości serwerów oraz oszczędzania zasobów należy także zoptymalizować działanie modułu muzycznego. Odtwarzanie

dźwięków na pustym kanale oraz połączenie z takim kanałem powinny zostać wykluczone, co może mieć duże znaczenie podczas znacznego obciążenia aplikacji. Aby sprawniej zarządzać sesjami oraz użytkownikami, przydatny mógłby być interfejs zaimplementowany w formie graficznej, np. w przeglądarce. Zamiast wpisywania komend, użytkownik mógłby część funkcji wywołać przy pomocy elementów graficznych. To umożliwiłoby również przechowywanie całych kart postaci oraz wygodne zarządzanie nimi (przy obecnym interfejsie prowadzenie karty postaci jest zbyt żmudne). Jednym z ostatnich kroków rozwoju dla aplikacji jest udostępnienie jej dla szerszej publiczności na publicznym hostingu, aby każdy użytkownik Discorda mógł dodać bota na swój własny serwer i korzystać z niego niezależnie od uruchomionego lokalnie skryptu.

5.3. Wnioski

Dzięki przygotowaniu projektu RPGamer poznałem proces tworzenia aplikacji z zupełnie innej perspektywy. Przede wszystkim dzięki braku z góry narzuconych celów czy specyfikacji, musiałem głęboko zastanowić się czy proponowane przeze mnie rozwiązania będą praktyczne i użyteczne dla przeciętnego użytkownika. Należało przeprowadzić analizę najbardziej potrzebnych mechanizmów oraz zaimplementować je, używając gotowych rozwiązań proponowanych przez aplikację Discorda.

Dużą nowością dla mnie było podejście asynchroniczne w tworzeniu kodu. Musiałem zwracać uwagę na to czy dana metoda nie wykonuje się zbyt długo i nie blokuje wykonania innych metod. Także przy korzystaniu z zewnętrznych bibliotek, trzeba było uważać na ich blokujące wywołania (np. przy pobieraniu danych z serwisu YouTube przez youtube_dl).

Z uwagi na wielkość projektu i dużą liczbę metod oraz aspektów, którym należało poświęcić czas nauczyłem się również priorytetyzować zadania do wykonania. Zaadaptowałem nie stosowany przeze mnie wcześniej sposób pracy, jakim jest podział zadań według poziomu istotności z punktu widzenia działania projektu. Stworzyłem tablicę na stronie Monday[1], na której każde zadanie było klasyfikowane do jednej z trzech kategorii: *Crucial*, *Medium crucial* oraz *Feature*. Przed każdą sesją tworzenia kodu podsumowywałem to jakie cele zostały do zrealizowania oraz czym zająć się w następnej kolejności. Pozwoliło to na sprawną i dokładną pracę nad najbardziej istotnymi częściami bota, a części mogące stanowić jedynie dodatek nie wpływający znacznie na funkcjonalność pozostawić do poprawy w ewentualnych kolejnych wersjach aplikacji.

[1] `https://www.monday.com`

Spis rysunków

Listingi

Bibliografia

[1] Baniak Baniaka. *Baniak Baniaka - YouTube*. URL: https://www.youtube.com/channel/UCXnI7wpHJ-x8bafp1BK3DUQ. (dostęp: 20.05.2021).

[2] Michael Bayer. *SQLAlchemy - The Database Toolkit for Python*. URL: https://www.sqlalchemy.org/. (dostęp: 20.05.2021).

[3] Refsnes Data. *SQL Injection*. URL: https://www.w3schools.com/sql/sql_injection.asp. (dostęp: 20.05.2021).

[4] dstftw. *ytdl-org/youtube-dl: Command-line program to download videos from YouTube.com and other video sites*. URL: https://github.com/ytdl-org/youtube-dl. (dostęp: 20.05.2021).

[5] TeamSpeak Systems GmbH. *Home — TeamSpeak*. URL: https://www.teamspeak.com/pl/. (dostęp: 20.05.2021).

[6] Dimitri van Heesch. *Doxygen: Doxygen*. URL: https://www.doxygen.nl/index.html. (dostęp: 20.05.2021).

[7] D. Richard Hipp. *SQLite Home Page*. URL: https://www.sqlite.org/index.html. (dostęp: 20.05.2021).

[8] Discord Inc. *Creating a Bot Account*. URL: https://discordpy.readthedocs.io/en/stable/discord.html. (dostęp: 20.05.2021).

[9] Discord Inc. *Jak stworzyć serwer? – Discord*. URL: https://support.discord.com/hc/pl/articles/204849977-Jak-stworzy%C4%87-serwer-. (dostęp: 20.05.2021).

[10] JetBrains. *PyCharm: the Python IDE for Professional Developers by JetBrains*. URL: https://www.jetbrains.com/pycharm/. (dostęp: 20.05.2021).

[11] Microsoft. *Konferencje wideo, spotkania, połączenia — Microsoft Teams*. URL: https://www.microsoft.com/pl-pl/microsoft-teams/group-chat-software. (dostęp: 20.05.2021).

[12] Microsoft. *Skype — Twój sposób na bezpłatne rozmowy i czat*. URL: https://www.skype.com/pl/. (dostęp: 20.05.2021).

[13] Rapptz. *Cogs*. URL: https://discordpy.readthedocs.io/en/stable/ext/commands/cogs.html. (dostęp: 20.05.2021).

[14] Rapptz. *discord.py/examples at master · Rapptz/discord.py · GitHub*. URL: https://github.com/Rapptz/discord.py/tree/master/examples. (dostęp: 20.05.2021).

[15] Rapptz. *Welcome to discord.py*. URL: https://discordpy.readthedocs.io/en/stable/. (dostęp: 20.05.2021).

[16] FFmpeg team. *FFmpeg*. URL: https://www.ffmpeg.org/. (dostęp: 20.05.2021).

[17] Wikipedia. *Bot (program) – Wikipedia, wolna encyklopedia*. URL: https://pl.wikipedia.org/wiki/Bot_(program). (dostęp: 20.05.2021).

[18] Wikipedia. *Gra fabularna – Wikipedia, wolna encyklopedia*. URL: https://pl.wikipedia.org/wiki/Gra_fabularna. (dostęp: 20.05.2021).

[19] Wikipedia. *Znak modyfikacji – Wikipedia, wolna encyklopedia*. URL: https://pl.wikipedia.org/wiki/Znak_modyfikacji. (dostęp: 20.05.2021).

[20] Wikipedia. *Znak modyfikacji – Wikipedia, wolna encyklopedia*. URL: https://pl.wikipedia.org/wiki/Mapowanie_obiektowo-relacyjne. (dostęp: 20.05.2021).