



Discord bot

ZÁVĚREČNÁ PRÁCE

Prohlašuji, že jsem svou závěrečnou práci napsal samostatně a výhradně s použitím citovaných pramenů.

Ve Velkém Meziříčí dne 11. 4. 2023

Petr Ambrož

Obsah

1	Úvod	4
2	Teoretická část	5
2.1	Historie programovacích jazyků	5
2.1.1	Co to jsou programovací jazyky.....	5
2.1.2	Nejstarší jazyky.....	5
2.1.3	Jazyky symbolických instrukcí	5
2.1.4	Vyšší programovací jazyky.....	5
2.1.5	Jazyk C	6
2.1.6	Java.....	6
2.1.7	Python	6
2.2	Discord, jeho využití a boti	8
2.2.1	Využití Discordu.....	8
2.2.2	Boti	8
3	Praktická část	10
3.1	Registrace na Discordu, vytvoření serveru.....	10
3.1.1	Jak se zaregistrovat	10
3.1.2	Jak vytvořit vlastní server	10
3.2	Vývojářský portál, vytvoření bota, připojení na náš server	11
3.2.1	Vytvoření bota na vývojářském portálu	11
3.2.2	Připojení bota na server	12
3.3	Co bot umí?	13
3.3.1	Seznam příkazů	13
3.3.2	Tvorba memů	13
3.3.3	Emailové oznámení	14
3.3.4	Hra hangman.....	14
3.4	Spuštění Python skriptu	15
3.4.1	Instalace Pythonu.....	15
3.4.2	Vytvoření virtuálního prostředí	15
3.4.3	Instalace potřebných balíčků	16
3.4.4	Spuštění bota	17
3.5	Princip spuštění a připojení	17
3.5.1	Import balíčků	17
3.5.2	Token a připojení	18
3.5.3	.env soubor	18
3.6	Princip fungování jednotlivých příkazů	19
3.6.1	Asynchronní programování.....	19
3.6.2	!seznam_prikazu	19
3.6.3	!list_memes.....	20
3.6.4	!make_meme	21
3.6.5	!subscribe a !unsubscribe	23
3.6.6	!play_hangman	25
3.6.7	!guess	26
3.7	PEP8	28
4	Závěr	29
5	Seznam zdrojů	30
6	Seznam obrázků	31

1 Úvod

Z velkého množství probíraných témat v semináři informatiky pro mě nebylo jednoduché vybrat jedno, které by mi bylo nejbližší. Co mi však pomohlo se rozhodnout bylo absolvování části Korespondenčního semináře z informatiky¹, který každoročně pořádá Fakulta Informatiky Masarykovy Univerzity. Zapojil jsem se do něj v listopadu 2022, prakticky rok poté, co jsme v informatice probírali Python, což je právě jazyk, na který se první část KSI zaměřuje. Musel jsem tedy oprášit všechny znalosti, které jsem měl, a pustil jsem se do řešení úloh.

Nejvíce mě oslovila jedna ze závěrečných úloh, kde bylo zadáno naprogramování bota, který bude umět na discordovém serveru reagovat na různé příkazy. Své řešení jsem se rozhodl použít i jako základ pro svoji maturitní práci, jelikož mě jeho tvorba velmi bavila a posunula mé dovednosti značně dopředu. Velkým bonusem pro mě bylo to, že moje řešení úlohy kontroloval jeden ze studentů Fakulty informatiky a poskytl mi zpětnou vazbu ohledně toho, co by se dalo zlepšit, díky čemuž jsem mohl svůj kód znovu projít a případné nedostatky odstranit.

Teoretická část jedná o historii vývoji programování od nejstarších jazyků až po ty nejčastěji používané v dnešní době.

Cíl praktické části je tvorba kódu bota v jazyce Python, který umí reagovat na různé příkazy v textových kanálech Discordu. Mezi tyto příkazy bude patřit tvorba memů se zadaným textem, oznámení o označení uživatele pomoci emailu a hraní hry *hangman* – něco jako šibenice, avšak s omezením pouze na textové prostředí.

Programovací jazyk, ve kterém je řešení zhotoveno, je **Python**. Ze všech jazyků, jenž jsem měl doposud možnost si vyzkoušet mi je nejbližší – především svou jednoduchostí a nenáročností na pochopení. Zároveň je výhodou jeho univerzálnost a možnost ho bez obtíží spustit na prakticky jakémkoliv PC, na kterém je nainstalován.

¹ <https://ksi.fi.muni.cz>, zkracováno na KSI

2 Teoretická část

2.1 Historie programovacích jazyků

2.1.1 Co to jsou programovací jazyky

Programovací jazyk je prostředek sloužící k zapsání algoritmu neboli přesného sledu instrukcí, které mají být vykonány počítačem. Jde tedy o způsob komunikace mezi programátorem (člověk, jenž algoritmus v daném jazyce zapisuje) a počítačem, který dané úlohy vykoná.

2.1.2 Nejstarší jazyky

U prvních počítačů se pro zápis programů používal strojový kód, jehož sémantika byla závislá na skladbě technických prvků daného počítače. Velká nevýhoda tohoto způsobu zápisu byla však vysoká složitost a nepohodlnost zápisu instrukcí. Z tohoto důvodu byly brzy nahrazeny.

2.1.3 Jazyky symbolických instrukcí

Tyto jazyky již nepoužívaly numericky kódované informace, ale zápisy psané symbolicky, které byly díky překladači (tzv. assembleru) překládány do strojového kódu. To zajistilo mnohem lepší čitelnost a orientaci v kódu, ale stále bylo nutné takový jazyk používat vzhledem k vlastnostem konkrétního počítače.

2.1.4 Vyšší programovací jazyky

Vývoj těchto jazyků měl především dva cíle: odstranění nutnosti znát detailně hardware počítače a možnost spuštění programu na různých, technicky odlišných počítačích. Prvním jazykem, který takové požadavky splňoval byl Fortran. Byl poměrně snadno naučitelný, efektivní, avšak jeho použití bylo hlavně pro vědeckotechnické výpočty, proto neuměl řešit vše.

Prvním uceleným jazykem, který nabízel širokou škálu funkcí byl Algol 60. Sice se prakticky moc neuplatnil, ale právě díky němu vznikla spousta nových jazyků, které z něj vycházely. V roce 1968 byl znovelizován a vznikl tak Algol 68, který byl oproti svému předchůdci vybaven širší škálou efektivnějších nástrojů.

Na Algol 68 navázal jazyk PASCAL, který byl vyvinut především pro výuku programování, nicméně i v dnešní době je hojně používán nejen pro výuku.

2.1.5 Jazyk C

Specifickou roli zastává jazyk C, který nemůžeme zařadit mezi vysoké jazyky jako např. PASCAL, ale ani mezi ty nízké na úrovni assembleru. C-jazyk totiž dokáže pracovat na obou úrovních, tedy jak na systémové, tak i na aplikační úrovni. Jednou z jeho hlavních výhod je jeho rozšiřitelnost díky knihovnám.

I přes své stáří (vyvinut byl v 70. letech 20. století) se v dnešní době těší vysoké popularitě jak při tvorbě systémového softwaru, tak i aplikací.

Zároveň tento jazyk ovlivnil mnoho dalších v dnešní době často užívaných jazyků, jako je např. Java, C#, D, či PHP, které s ním nejsou přímo příbuzné, ale převzali od něj část syntaxe nebo některé funkce. C++ je potom jazyk odvozený od C, který implementuje zároveň objektově orientované i procedurální programování.

2.1.6 Java

Java je velmi používaným objektově orientovaným jazykem. Její vývoj začal již v 90. letech a první verze byla vydána v roce 1995. V dnešní době je Java spravována světoznámou firmou Oracle. Má velmi široké využití – od webových, desktopových a mobilních aplikací až například po hry, především pro platformu Android.

Jednou z hlavních předností Javy je její celosvětová rozšířenost – Oracle na svých stránkách uvádí, že ji používá přes 3 miliardy zařízení. Další její nespornou výhodou je fakt, že je prakticky nezávislá na operačním systému, tudíž v ní není nikterak složité vyvíjet aplikace pro různé platformy.²

2.1.7 Python

Python patří mezi vysokoúrovňové jazyky, což znamená, že je značně nezávislý na hardwarovém uspořádání počítače. Dnes je nejčastěji používán jako skriptovací jazyk pro webové aplikace, může být však použit i pro řadu jiných funkcí – uplatnění nalézá např. při

² History of Java Programming Language. *UNext* [online]. [cit. 2023-04-05]. Dostupné z: <https://unext.com/blogs/java/history-of-java/>

vývoji 2D i 3D her, v umělé inteligenci a strojovém učení³. Jeho oblíbenost potvrzuje i fakt, že některé operační systémy (např. Linux) obsahují Python již v základní instalaci.⁴

Nespornou výhodou Pythonu je možnost kombinace jak procedurálního, tak i objektově orientovaného a funkcionálního programování tak, jak je to zrovna potřeba. Mezi další pozitiva můžeme zařadit např. jednoduchost implementace externích knihoven funkcí, dynamické typování dat (byť toto může být v určitých případech považováno za nevýhodu) či ohraničování bloků kódu pomocí tzv. indentace, tedy odsazení tabulátorem nebo čtyřmi mezerami namísto složených závorek, které se využívají např. u Javy nebo C. To přidává kódu na přehlednosti. Ukažme si to na příkladu:

```
int x = 15;
int y = 10;
if (x > y) {
    System.out.println("x je vetsi nez y");
}
```

Obrázek 1: ukázka jednoduchého kódu v Javě⁵

```
x = 15
y = 10
if x>y:
    print("x je vetsi nez y")
```

Obrázek 2: ukázka kódu v Pythonu se stejnou funkcionalitou jako v obrázku 1⁶

Při porovnání obou příkladů si lze povšimnout hned dvou výše zmiňovaných vlastností – ohraničení funkce *if* pomocí závorek, respektive odsazení a odpadající nutnost psát před proměnné datový typ. Navíc Python nevyžaduje zakončení řádku středníkem.

Byť to z takto krátkého příkladu není zcela patrné, celkově je kód psaný v Pythonu velmi přehledný a jednoduchý na orientaci, což je jeden z důvodů jeho oblíbenosti.

³ Python eats away at R: Top Software for Analytics, Data Science, Machine Learning in 2018: Trends and Analysis. *KDnuggets* [online]. [cit. 2023-03-25]. Dostupné z: <https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html/2>

⁴ *Python setup and usage: Using Python on Unix platforms* [online]. [cit. 2023-03-25]. Dostupné z: <https://docs.python.org/3/using/unix.html>

⁵ Vlastní zpracování

⁶ Vlastní zpracování

2.2 Discord, jeho využití a boti

2.2.1 Využití Discordu

Discord je moderní komunikační platforma, oblíbená především mezi hráči počítačových her, nicméně své využití najde i u nehráčů. Je flexibilnější alternativou například pro Skype, ICQ nebo Teamspeak.

Discord nabízí možnost komunikace přes chat, VoIP hovory či videohovory – a to jak přímo mezi dvěma či více uživateli, tak pomocí tzv. serverů. Server je komunita lidí, kteří se k němu pomocí speciálního odkazu připojili a sdílejí společné textové a hlasové kanály. Přístupnost těchto kanálů může být dále vyhrazena pro užší skupinu uživatelů díky širokým možnostem nastavení oprávnění a rolí. Toto umožňuje např. velmi efektivní správu daného serveru v případě, že je k němu připojeno větší množství lidí. Uvedu příklad: uživatelé s administrátorskými právy smí měnit role uživatelů, vylučovat je, vytvářet nové kanály atd., moderátoři pak mohou mít oprávnění psát zprávy do informačních kanálů (kam běžní uživatelé psát nesmí), odstraňovat nevhodné zprávy, či jiným způsobem komunikaci moderovat. Uživatelé bez speciální role pak mají možnost psát zprávy a používat hlasové kanály, ale již nesmí měnit nastavení serveru.

2.2.2 Boti

2.2.2.1 Využití

Boti na Discordu vystupují na stejné úrovni jako běžní uživatelé, s aplikací je však lze snadno poznat podle nápisu „BOT“, který mají vedle jména. Rozdíl oproti uživatelům je však v tom, že za jejich „vystupování“ nezodpovídá člověk, ale počítač, na kterém je předem naprogramováno, co má bot dělat a např. na jaký typ zpráv reagovat.

Jejich využití může být všelijaké – záleží na programátorovi, co bota „naučí“ a k čemu ho chce využít. Může to být např. pro jednodušší správu serveru, např. přiřazování oprávnění uživatelům, nebo psaní automatizovaných zpráv pro přivítání nových členů, testování pingů serveru, nebo dokonce tvorbu memů, vyhledávání písniček podle části textu či cokoliv jiného.

2.2.2.2 Princip fungování

Aby bot fungoval, je potřeba se nejprve zaregistrovat na vývojářský portál Discordu. Ten je dostupný všem uživatelům zdarma. Na portálu lze poté vytvořit nového bota či spravovat

existujícího. Při tvorbě mu můžeme vybrat jméno a profilovou ikonu podle libosti – pod těmi bot bude vystupovat na všech serverech, kde bude přidán. Důležitou součástí jeho fungování je tzv. TOKEN, identifikační řetězec znaků, přes který se počítač, na kterém budeme spouštět program bota, připojí k Discordu a konkrétně k našemu botovi. Je velmi důležité, abychom tento token s nikým nesdíleli, protože by potom mohl kdokoliv naši funkcionalitu bota nahradit svou vlastní, což nechceme. Proto je také vhodné nemít token v samotném kódu, ale v jiném souboru, pro případ, že bychom s někým chtěli kód sdílet.⁷

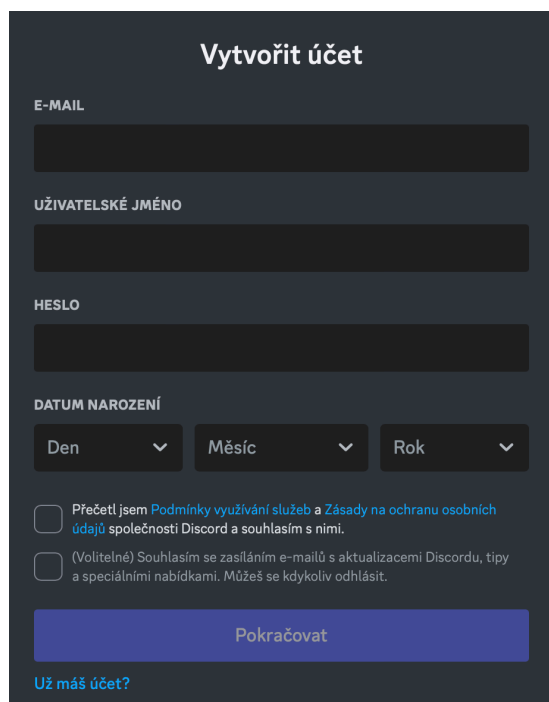
⁷ LOPATKA, Adam. Discord bot příprava [online]. [cit. 2023-03-25]. Dostupné z: <https://ksi.fi.muni.cz/ulohy/512>

3 Praktická část

3.1 Registrace na Discordu, vytvoření serveru

3.1.1 Jak se zaregistrovat

Abychom mohli vůbec Discord používat, musíme si na něm vytvořit uživatelský účet. Na stránce <https://discord.com/login> vybereme položku *Regisruj se*. Pro zdárné dokončení registrace musíme zadat svoji emailovou adresu, datum narození a vytvořit si uživatelské jméno a heslo.

The image shows the Discord registration form on a dark background. At the top, the title 'Vytvořit účet' is centered. Below it, there are several input fields: 'E-MAIL', 'UŽIVATELSKÉ JMÉNO', and 'HESLO'. Under the 'HESLO' field, there is a 'DATUM NAROZENÍ' section with three dropdown menus for 'Den', 'Měsíc', and 'Rok'. Below these are two checkboxes with text: the first checkbox is for agreeing to the Terms of Service and Privacy Policy, and the second checkbox is for agreeing to receive emails from Discord. At the bottom of the form is a blue button labeled 'Pokračovat'. Below the button, there is a link that says 'Už máš účet?'.

Obrázek 3: registrační formulář

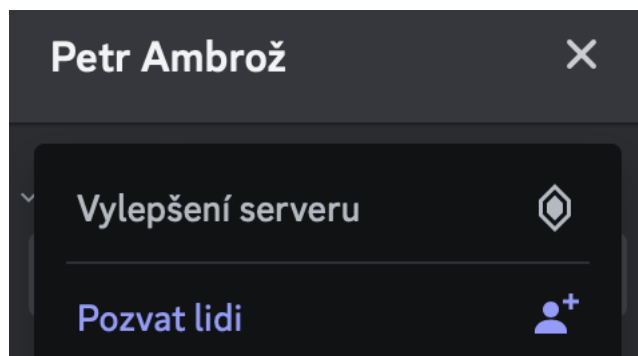
3.1.2 Jak vytvořit vlastní server

Dalším krokem bude vytvoření našeho vlastního serveru. K tomu nám poslouží na levé liště tlačítko s ikonou „plus“ a popiskem *Přidat server*. Po kliknutí vybereme horní možnost *Vytvořit vlastní* a poté možnost *Pro mě a moje přátele*. Poté již stačí server pojmenovat a kliknout na *Vytvořit*. Tímto nám vznikne prázdný server s jedním textovým a jedním hlasovým kanálem. V případě potřeby lze samozřejmě vytvořit kanály další, avšak pro naše potřeby si vystačíme s těmito přednastavenými.⁸

⁸ Postup platný k dubnu 2023. V případě novějších aktualizací se může lišit.

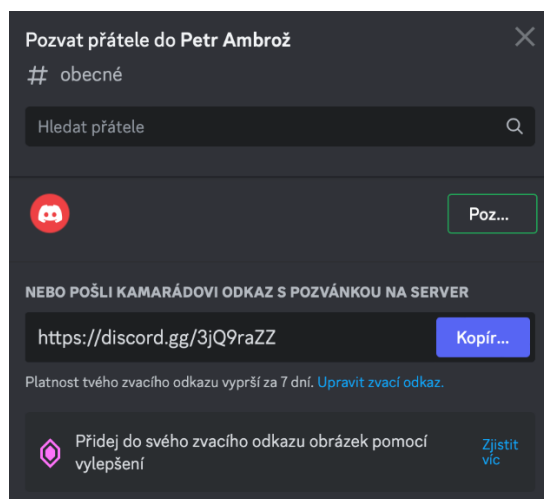
Přidání dalších uživatelů na server

Aby Discord plnil svou funkci – komunikace, viz 2.2.1 [Využití Discordu](#), je potřeba vytvořit odkaz, díky kterému se budou moci připojit do našeho serveru ostatní uživatelé. Klikneme tedy vlevo nahoře na rozbalovací nabídku s názvem serveru a vybereme možnost *Pozvat lidi*.



Obrázek 4: rozbalovací nabídka, tlačítko pro pozvání

Ve spodní části zobrazeného okna se nám objeví odkaz, který nyní můžeme nasdílet ostatním uživatelům. Tento odkaz je platný pouze po 7 dní, avšak toto nastavení lze v nabídce *Upravit zvací odkaz* změnit.



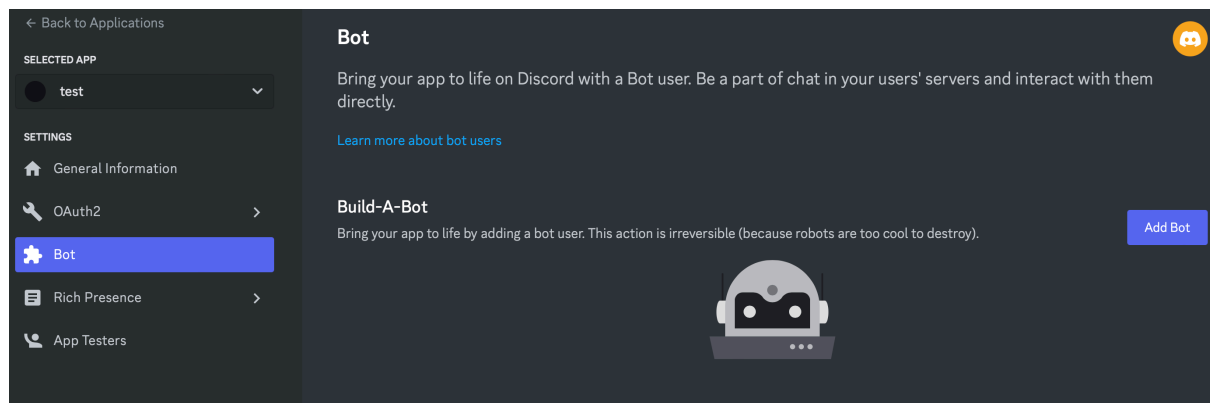
Obrázek 5: okno pro tvorbu zvacího odkazu

3.2 Vývojářský portál, vytvoření bota, připojení na náš server

3.2.1 Vytvoření bota na vývojářském portálu

Protože programování vlastního bota není zcela běžná věc, kterou by většina uživatelů Discordu dělala, jsou veškeré nástroje na samostatné stránce, tzv. vývojářském portálu, který je dostupný na adrese <https://discord.com/developers/>. Pro vytvoření bota musíme

nejprve vytvořit novou aplikaci kliknutím na tlačítko *New application* v pravém horním rohu. Budeme vyzváni k pojmenování aplikace a stisknutím *create* vytvoření dokončíme. Nyní musíme k této aplikaci přiřadit bota – vybereme v levé nabídce záložku *Bot* a poté vpravo tlačítko *Add bot*, viz *Obrázek 6*.



Obrázek 6: vytvoření bota

Zobrazí se nám nabídka, kde můžeme botovi dát vlastní jméno a případně mu nahrát profilový obrázek. Důležité je také tlačítko *Reset token*, které slouží k vytvoření připojovacího tokenu, o tom však podrobněji v kapitole 3.5.2 *Token a připojení*.

Aby nám bot správně fungoval, je ještě nutné zapnout možnost *Message Content Intent*, která mu umožní číst zprávy.

3.2.2 Připojení bota na server

Nyní můžeme připojit bota na sever, který jsme si vytvořili (viz kapitola 3.1.2 *Jak vytvořit vlastní server*). Otevřeme si záložku *OAuth2* a její podzáložku *URL generator*. Zde v nabídce *SCOPES* zaškrtneme, že se má jednat o bota. V *BOT PERMISSIONS* nastavíme, jaké má mít bot oprávnění – v našem případě vybereme *Administrator*, což automaticky udělí oprávnění všechna. Samozřejmě je možné omezit oprávnění pouze na ta, která jsou pro správnou funkci nezbytná, což je vhodné zejména tehdy, pokud bude mít k botovi přístup více uživatelů.

SCOPES

<input type="checkbox"/> identify	<input type="checkbox"/> email	<input type="checkbox"/> connections
<input type="checkbox"/> guilds	<input type="checkbox"/> guilds.join	<input type="checkbox"/> guilds.members.read
<input type="checkbox"/> gdm.join	<input type="checkbox"/> rpc	<input type="checkbox"/> rpc.notifications.read
<input type="checkbox"/> rpc.voice.read	<input type="checkbox"/> rpc.voice.write	<input type="checkbox"/> rpc.activities.write
<input checked="" type="checkbox"/> bot	<input type="checkbox"/> webhook.incoming	<input type="checkbox"/> messages.read
<input type="checkbox"/> applications.builds.upload	<input type="checkbox"/> applications.builds.read	<input type="checkbox"/> applications.commands
<input type="checkbox"/> applications.store.update	<input type="checkbox"/> applications.entitlements	<input type="checkbox"/> activities.read
<input type="checkbox"/> activities.write	<input type="checkbox"/> relationships.read	<input type="checkbox"/> voice
<input type="checkbox"/> dm_channels.read	<input type="checkbox"/> role_connections.write	
<input type="checkbox"/> applications.commands.permissions.update		

BOT PERMISSIONS

GENERAL PERMISSIONS	TEXT PERMISSIONS	VOICE PERMISSIONS
<input checked="" type="checkbox"/> Administrator	<input type="checkbox"/> Send Messages	<input type="checkbox"/> Connect
<input type="checkbox"/> View Audit Log	<input type="checkbox"/> Create Public Threads	<input type="checkbox"/> Speak

Obrázek 7: výběr oprávnění

Nyní již můžeme zcela dole na stránce zkopírovat odkaz pro připojení, otevřít ho v novém okně, vybrat ke kterému serveru chceme bota připojit a kliknout na *Autorizovat*. Pokud se vrátíme do okna běžného Discordu, uvidíme, že se bot připojil. Také si můžeme povšimnout, že je bot offline – to proto, že zatím neběží program, který by ho obsluhoval.

3.3 Co bot umí?

Hlavním úkolem bota je reagovat na příkazy, které pozná tak, že zpráva v kanálu začíná předem definovaným znakem, tzv. prefixem. V případě tohoto bota to je znak vykřičníku. Na serveru samozřejmě může být připojeno vícero botů, pak je ale vhodné, aby měl každý svůj vlastní prefix, aby se nestalo, že na jeden příkaz budou reagovat dva boti.

3.3.1 Seznam příkazů

Po napsání *!seznam_prikazu* nebo *!prikazy* do kteréhokoliv textového kanálu nám bot odpoví zprávou, která obsahuje všechny aktuálně použitelné příkazy včetně nápovědy, co máme do příkazu napsat jako argumenty, pokud to příkaz vyžaduje.

3.3.2 Tvorba memů

První funkcí je tvorba memů. K té slouží dva příkazy – *!list_memes* a *!make_meme*. První příkaz nevyžaduje argumenty a bot pošle do zprávu s aktuálně populárními šablonami pro memy

na stránce <https://imgflip.com/>. Druhý příkaz potom zajistí tvorbu memu, který bot poté pošle do textového kanálu. Je třeba uvést jako argument id šablony (ze seznamu populárních vzorů) a texty, které budou zobrazeny na horní a dolní části. Aby je bot správně pochopil, musí být ohraničeny uvozovkami (`!make_meme 123 "text1" "text2"`).

3.3.3 Emailové oznámení

Další funkcí je oznámení, že nás jiný uživatel označil v nějakém discordovém kanálu (napíše do zprávy `@naše_uživatelské_jméno`), které dostaneme na email. To se hodí v případě, že zrovna nejsme u PC nebo nemáme Discord otevřený, ale někdo se přímo nás snaží kontaktovat. K tomuto slouží příkaz `!subscribe <email>` pro přihlášení k dostávání emailů a `!unsubscribe` v případě, že již oznámení dostávat nechceme.

3.3.4 Hra hangman

Účelem této funkce je možnost si s botem zahrát hru hangman, což je anglický název pro naši šibenici. Princip je jednoduchý – bot má uložený seznam anglických slov, ze kterého náhodně jedno vybere, vytvoří „grafické rozhraní“, ve kterém zobrazí jméno hráče, již hádaná písmena, zbývající počet životů a zobrazení uhodnutých a neuhodnutých písmen ve slově.



Obrázek 8: ukázka ze hry hangman

Pomocí příkazu `!play_hangman` hru spustíme. Během probíhající hry ji tímto příkazem ukončíme a vytvoříme novou.

Příkazem `!guess <písmeno>` hádáme jednotlivá písmena, bot automaticky pro větší přehlednost tuto zprávu maže. Na posledním řádku napíše, zda jsme uhodli správně, špatně, zda nám došly životy (v tomto případě odhalí slovo) nebo jsme vyhráli.

3.4 Spuštění Python skriptu

Poznámka: postup v této kapitole je platný k dubnu 2023, používá verzi Pythonu 3.10 a operační systém Windows 10. Při použití jiných verzí či jiného operačního systému je možné, že nebude fungovat. To platí především pro systém Linus a MacOS, kde jsou jiné příkazy pro spuštění Pythonu a podložky virtuálního prostředí mají jinou strukturu.

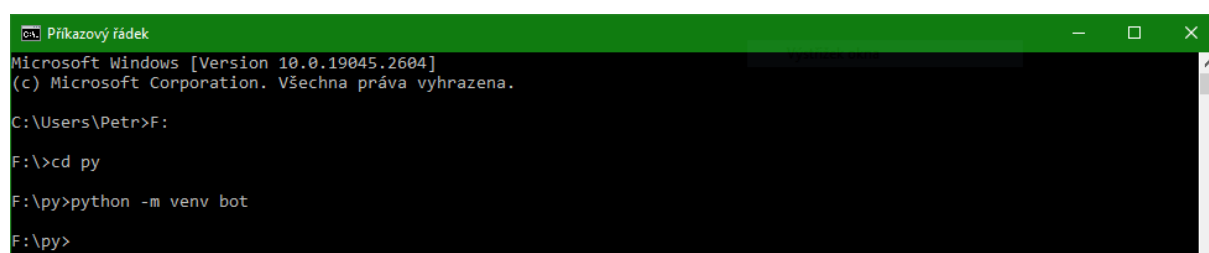
3.4.1 Instalace Pythonu

Jelikož bot umí pracovat bez instalace speciálních programů, postačí nám k němu příkazový řádek a nainstalovaný Python. Ten si můžeme stáhnout na adrese <https://www.python.org/downloads/>. Je vhodné mít nejnovější verzi, minimálně však Python 3.10, na které je skript aktuálně sestaven a otestován.⁹

3.4.2 Vytvoření virtuálního prostředí

Je dobrým zvykem skripty v Pythonu spouštět ve virtuálním prostředí, protože to umožňuje instalaci balíčků jen pro daný program a zároveň je jednoduché je pak odstranit. Pokud bychom instalovali balíčky bez virtuálního prostředí, hrozilo by, že se přestaneme orientovat v tom, které jsme použili a které ne. Zjednodušeně řečeno, balíčky jsou soubory, které obsahují moduly rozšiřující běžnou knihovnu funkcí Pythonu. Tyto balíčky můžeme (za dodržení licenčních podmínek) využít pro své programování.¹⁰

Začneme otevřením příkazového řádku. Pomocí příkazu `cd` přejdeme do složky, kde chceme mít bota umístěného. Nyní stačí zadat příkaz `python -m venv nazev_prostredi`, který nám vytvoří nové prázdné prostředí. Pojmenovat ho můžeme, jakkoliv chceme.



Obrázek 9: vytvoření virtuálního prostředí

⁹ Knihovna `discord.py` by měla být kompatibilní až po Python 3.8, ale nemám to vyzkoušené, raději bych se držel verze 3.10

¹⁰ PROCHÁZKA, Karel. *PIP správa balíčkov* [online]. [cit. 2023-04-04]. Dostupné z: <https://ksi.fi.muni.cz/ulohy/504>

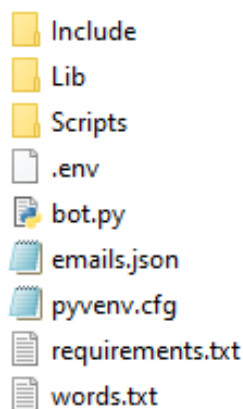
3.4.3 Instalace potřebných balíčků

Python umožňuje jednoduchou správu a instalaci balíčků pomocí předinstalovaného nástroje *pip*. Jak bylo zmíněno výše, my je budeme instalovat pouze do virtuálního prostředí, proto ho musíme nejprve aktivovat. Pomocí příkazu *cd* vstoupíme do složky prostředí (má stejný název jako prostředí) a zadáme příkaz *Scripts\activate*. Úspěšnou aktivaci značí název prostředí v závorkách, viz Obrázek 10.

```
F:\py>cd bot
F:\py\bot>Scripts\activate.bat
(bot) F:\py\bot>
```

Obrázek 10: aktivace virtuálního prostředí

Nyní můžeme začít s instalací. Do složky virtuálního prostředí vložíme soubory *bot.py*, *words.txt*, *.env*, *emails.json* a *requirements.txt*. Do příkazového řádku (pořád otevřeném ve složce a aktivovaným virtuálním prostředím) zadáme příkaz *pip install -r requirements.txt*, který automaticky nainstaluje všechny potřebné balíčky.



Obrázek 11: složka virtuálního prostředí se všemi soubory v OS Windows 10

Samozřejmě lze instalovat jednotlivé balíčky ručně (nebo je mazat), což je vhodné při vývoji nové aplikace, kdy začneme s čistým Pythonem a instalujeme pouze ty balíčky, které potřebujeme. Pokud však přebíráme řešení někoho jiného a nevíme, jaké balíčky použil, je nejlepší se spolehnout právě na *requirements.txt*, který obsahuje všechny balíčky nutné pro spuštění (pokud ho máme vůbec k dispozici). Správný programátor by vždy měl tento soubor ke svému kódu přikládat a udržovat ho aktuální.

3.4.4 Spuštění bota

Nyní je již vše připravené pro spuštění samotného bota. Pokud máme stále aktivované virtuální prostředí, stačí do příkazového řádku zadat `python ./bot.py`. Mělo by se zobrazit několik informačních hlášení, na discordovém serveru bychom měli vidět, že bot již není offline, a v příkazovém řádku se zobrazí hláška „bot se připojil“.

Před samotným spuštěním bota je navíc nutné doplnit všechny údaje do `.env` souboru. Viz kapitola 3.5.3 *.env soubor*.

Přiložený .env soubor z bezpečnostních důvodů obsahuje pouze názvy proměnných bez přihlašovacích údajů. Kompletní soubor je za účelem kontroly poskytnut pouze zadávajícímu učiteli.

3.5 Princip spuštění a připojení

3.5.1 Import balíčků

Prvním krokem k funkčnosti kódu je import balíčků umožňující funkce, které by nebylo snadné naprogramovat „od nuly“. U tohoto bota je využít především balíček `discord.py`, dále potom např. `requests`, `notifiers`, `dotenv`, `json`, a `random`. Pokud tyto balíčky nejsou součástí základní instalace Pythonu, je třeba je nejprve nainstalovat pomocí správce balíčků `pip` – viz kapitola 3.4.3 *Instalace potřebných balíčků*.

```
1  from typing import List
2  from os import getenv
3  import json
4  import random
5  import re
6  import requests
7  from discord import Intents, Message
8  from discord.ext import commands
9  from discord.ext.commands import Context
10 from dotenv import load_dotenv
11 from notifiers import get_notifier
```

Obrázek 12: import balíčků

Pro přehlednost a správnou funkci importujeme vždy na začátku kódu.

Nejdůležitější balíček, bez kterého by bylo velmi složité bota sestavit, je `discord.py`, který je přímo navržen tak, aby obsahoval všechny funkce, které by mohl takový bot potřebovat –

operace se zprávami, uživateli... Existuje k němu obsáhlá dokumentace, která popisuje, jak kterou funkci použít.¹¹ Během vývoje jsem s ní aktivně pracoval.

3.5.2 Token a připojení

Klíčem k připojení bota k serveru je již zmiňovaný token, viz kapitola 2.2.2.2 *Princip fungování*. Token je jedinečná kombinace znaků a číslic, která umožňuje Discordu identifikovat, kterého bota se námi spuštěný skript snaží ovládat. Kvůli bezpečnosti je uložen mimo samotný hlavní kód, a to především proto, aby bylo možné kód sdílet, ale nehrozila možnost, že někdo jiný bude našeho bota ovládat. Token je načten z `.env` souboru, který je uložen ve stejné složce, jako skript. Je uložen do proměnné `TOKEN`, kterou pak využije metoda `bot.run(TOKEN)` na posledním řádku kódu, která bota spustí a připojí k Discordu.

3.5.3 `.env` soubor

Je speciální soubor, do kterého se ukládají citlivé informace, které nechceme, aby byly uloženy v kódu. Mezi ně může patřit právě již zmíněný token, různé přihlašovací údaje, přístupové údaje či aj. Důvody, proč je dobré tyto informace takto ukládat jsou hned dva. Zaprvé je to bezpečnost, nemůže se stát, že při sdílení kódu se někdo dostane k našim informacím, protože `.env` soubor zásadně nikomu neposíláme, a za druhé, snadná změna údajů – pokud např. změníme heslo, nemusíme ho složitě hledat v kódu (to pomůže zvláště pokud má několik set řádků a již si nepamatujeme, pod kterou proměnnou jsme ho uložili) a pokud by bylo využité vícekrát, nemusíme ho přepisovat na všech místech. Jednoduše v `.env` souboru heslo aktualizujeme a v kódu se automaticky použije to nové.

```
1  DISCORD_TOKEN=  
2  MEME_USERNAME=  
3  MEME_PASSWORD=  
4  MAIL_USERNAME=  
5  MAIL_PASSWORD=  
6  SMTP_HOST=
```

Obrázek 13: ukázka obsahu `.env` souboru, za = je umístěn napsán konkrétní údaj

¹¹ Dokumentace je dostupná na adrese <https://discordpy.readthedocs.io/en/latest/>

DISCORD_TOKEN je token vygenerovaný na vývojářském portálu, viz kapitola 3.2.1 *Vytvoření bota na vývojářském portálu*

MEME_USERNAME a **MEME_PASSWORD** jsou přihlašovací údaje ke stránce imgflip.com, na kterou je potřeba se zaregistrovat

MAIL_USERNAME a **MAIL_PASSWORD** jsou přihlašovací údaje k SMTP serveru, přes který odesíláme emaily

SMTP_HOST je odkaz na SMTP server

Všechny tyto údaje je nutné do `.env` souboru uložit ještě před spuštěním bota. Údaje se zadávají jako prostý text bez uvozovek.

3.6 Princip fungování jednotlivých příkazů

3.6.1 Asynchronní programování

Celá knihovna `discord.py` (ze které čerpáme funkce) je stavěná tak, že funguje asynchronně, což znamená, že se nemusí čekat na dokončení předchozího příkazu či procesu, než může začít nový. To klasické synchronní programování neumožňuje. Z tohoto důvodu jsou všechny procedury a funkce definovány klíčovým slovem `async` a pokud je chceme zavolat, musíme použít `await`.

3.6.2 `!seznam_prikazu`

Tento příkaz je ze všech nejjednodušší. Je definován jako asynchronní procedura s názvem `prikazy`, před kterou stojí dekorátor `@bot.command` s určením, jak má příkaz znít, v tomto případě to je `seznam_prikazu`. V těle procedury je volána metoda `seznam_prikazu.create_message()`, které vygeneruje zprávu, která je příkazem `ctx.send()` poslána do textového kanálu. Kontext zprávy neboli informace o tom, kam byla poslána, její obsah a další podrobnosti jsou automaticky ukládány do objektu `ctx` knihovnou `discord.py`, tudíž ho nemusíme zjišťovat sami.

```
149 # --- seznam prikazu ---
150 seznam_prikazu = SeznamPrikazu
151
152
153 @bot.command(name="seznam_prikazu")
154 async def prikazy1(ctx: Context) -> None:
155     await ctx.send(seznam_prikazu.create_message(ctx))
```

Obrázek 14: příkaz `seznam_prikazu`

```

22 class SeznamPrikazu:
23     def __init__(self) -> None:
24         pass
25
26     def create_message(self):
27         message = (
28             "```\n"
29             "Přikazy:\n"
30             "!list_memes\n"
31             '!make_meme <id> "<text 1>" "<text 2>"\n'
32             "!subscribe <email>\n"
33             "!unsubscribe\n"
34             "!play_hangman\n"
35             "!guess <písmeno>\n"
36             "```\n"
37         )
38         return message

```

Obrázek 15: třída *SeznamPrikazu* a metoda *create_message*

3.6.3 !list_memes

V tomto příkazu je již využita předem definovaná třída, ze které je vytvořen objekt *meme_generator*. Opět je dekorátorem definován příkaz a v těle procedury je zavolána metoda *list.memes()* třídy *meme_generator*.

```

162 # --- tvorba memu ---
163 meme_generator = MemeGenerator()
164
165
166 @bot.command(name="list_memes")
167 async def list_memes(ctx: Context) -> None:
168     meme_list = meme_generator.list_memes()
169     await ctx.send(meme_list)
170     await ctx.send("https://imgflip.com/memetemplates")

```

Obrázek 16: příkazu *!list_memes*

Tato metoda si vyžádá seznam populárních memů z imgflipu, které převede do formátu *.json*. Z něj je potom vytažen seznam 25 populárních memů – jejich název a id. Ten je vrácen ve formátu *string* jako zpráva, která se pošle do textového kanálu opět metodou *ctx.send()*. Poté je poslána druhá zpráva, která obsahuje odkaz na imgflip, kde si můžeme zobrazit tyto populární šablony a zjistit, jak vypadají.

```

41 class MemeGenerator:
42     def __init__(self) -> None:
43         pass
44
45     def list_memes(self) -> str:
46         response = requests.get("https://api.imgflip.com/get_memes", timeout=10)
47         memes = response.json()
48         meme_id = []
49         meme_name = []
50         for i in range(25):
51             meme_id.append(memes["data"]["memes"][i]["id"])
52             meme_name.append(memes["data"]["memes"][i]["name"])
53         message = ""
54         for i in range(25):
55             message += "\n" + meme_id[i] + str(" ") + meme_name[i]
56         message += ""
57         return message

```

Obrázek 17: vytvoření zprávy s populárními šablonami

3.6.4 !make_meme

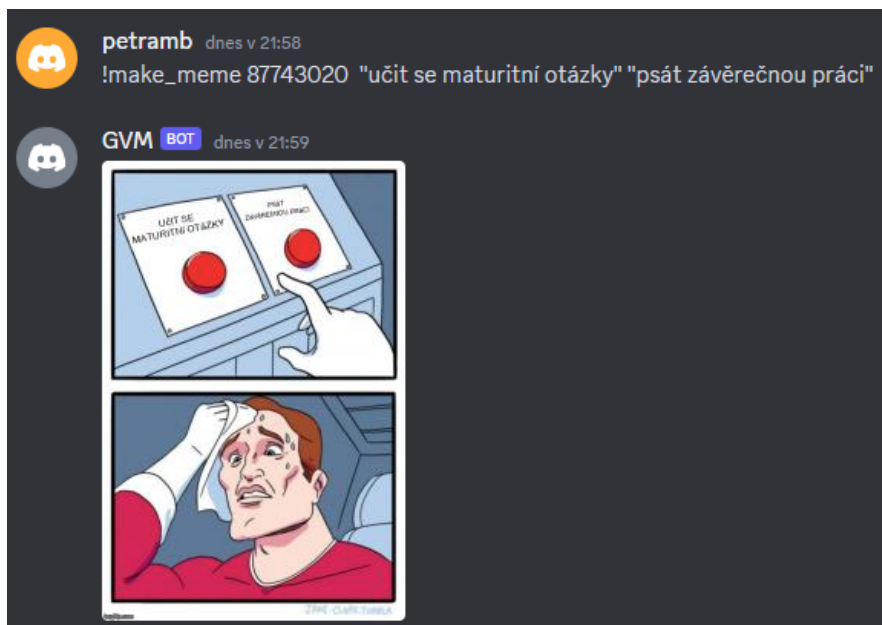
Pomocí tohoto příkazu lze vytvořit meme dle zadání. Princip je poměrně jednoduchý – příkaz přijme argumenty id šablony, horní text a dolní text (to je univerzální označení, některé memy dávají text např. i vedle sebe). Tyto argumenty (spolu s přihlašovacími údaji do imgflipu, které jsou načtené z `.env` souboru) jsou potom předány pomocí modulu `requests` metodou `.post()` do API imgflipu, které vygeneruje příslušný meme a vrátí ho jako odpověď. Ta je poté poslána do textového kanálu ve formě odkazu, který Discord automaticky načte a zobrazí obrázek.

```

59 def make_meme(
60     self, template_id: int, top_text: str, bottom_text: str
61 ) -> str:
62     response = requests.post("https://api.imgflip.com/caption_image", timeout=10,
63                             data={"template_id": template_id,
64                                   "username": getenv("MEME_USERNAME"),
65                                   "password": getenv("MEME_PASSWORD"),
66                                   "text0": top_text,
67                                   "text1": bottom_text})
68     meme_data = response.json()
69     return meme_data["data"]["url"]

```

Obrázek 18: požadavek na API a vrácení url odkazu na meme



Obrázek 19: zadání příkazu včetně argumentů, odpověď bota s memem



Obrázek 20: výsledek tvorby memu

Takto nějak pak může vypadat vygenerovaný meme. Nevýhodou je, že mnou použité API je stavěné především pro anglické prostředí a neumí dobře pracovat s češtinou.

3.6.5 !subscribe a !unsubscribe

Oba tyto příkazy slouží k oznamování zpráv, ve kterých nás někdo označí, viz kapitola 3.3.3 *Emailové oznámení*.

K tomuto účelu slouží třída *MentionsNotifier*. Ta ve svém konstruktoru vytvoří prázdný slovník (typ proměnné, funguje na principu klíč:záznam), který je nazván *emails* a využije se na ukládání emailových adres uživatelů. Při příkazu *!subscribe* se uloží email zadaný uživatelem spolu s jeho id, které je pak využito při zjišťování, který uživatel byl označen a je podle něj vyhledán jeho email.

Příkaz *!unsubscribe* zkontroluje, zda má uživatel registrovaný nějaký email a pokud ano, odstraní ho.

```
53 class MentionsNotifier:
54     def __init__(self) -> None:
55         self.emails = {}
56
57     def subscribe(self, user_id: int, email: str) -> None:
58         self.emails[user_id] = email
59
60     def unsubscribe(self, user_id: int) -> None:
61         if user_id in self.emails:
62             del self.emails[user_id]
```

Obrázek 21: třída *MentionsNotifier*

Zajímavější je dále zpracování zprávy, která obsahuje označení. Vytvoří se seznam obsahující id jednotlivých označených uživatelů a for cyklem se podle počtu označení vykoná metoda *notify_about mention*, která rozešle emaily.

```
176 @bot.event
177 async def on_message(message: Message) -> None:
178     users = message.mentions
179     mentions = []
180     if users != []:
181         for i in range(len(users)):
182             mentions.append(users[i].id)
183     print(mentions_notifier.emails)
184     for user in mentions:
185         if str(mentions[i]) in mentions_notifier.emails:
186             mentions_notifier.notify_about_mention(mentions[i],
187                                                         message.content,
188                                                         message.jump_url)
189     await bot.process_commands(message)
```

Obrázek 22: vytvoření seznamu označených uživatelů

Tato metoda funguje díky knihovně *notifier*, která se umí připojit na SMTP server, předat příslušné informace a odeslat email. Podle id je vyhledána emailová adresa uživatele a na ni je odeslána zpráva obsahující odkaz na server, ve kterém byl uživatel označen.

```
69     def notify_about_mention(self, user_id: List, msg_content: str,
70                               msg_url: str) -> None:
71         user_email = self.emails[str(user_id)]
72         email = get_notifier("email")
73         settings = {'host': 'ksi2022smtp.iamroot.eu',
74                    'port': 465,
75                    'ssl': True,
76                    'username': getenv("MAIL_USERNAME"),
77                    'password': getenv("MAIL_PASSWORD"),
78                    'to': user_email,
79                    'from': getenv("MAIL_USERNAME"),
80                    'subject': "Discord mention notification",
81                    'message': "Someone mentioned you in channel " + msg_url}
82         res = email.notify(**settings)
```

Obrázek 23: odeslání emailu

Výhodou je, že uživateli stačí provést příkaz *!subscribe* na kterémkoli ze serverů, kde je bot připojen a oznámení mu potom budou chodit i při označení na jiných serverech.

Za značnou nevýhodu pak považuji, že vytvořený seznam emailových adres není ukládán do externího souboru (byl by vhodný např. *.json*), ale do slovníku, který se při ukončení bota neuloží, takže při restartu bota jsou adresy ztraceny. Uživatel samozřejmě nemůže vědět, kdy se bot restartoval, aby se mohl k odběru oznámení znovu přihlásit.

Na poslední chvíli jsem se tento nedostatek rozhodl odstranit a ukládání do *.json* souboru skutečně implementovat. Soubor pro ukládání se jmenuje *emails.json* a jeho načtení do proměnné má automaticky formát slovníku, což znamená, že ho pouze stačí uložit do již existující proměnné *emails*, se kterou lze pracovat úplně stejně jako předtím.

Soubor je načten v konstruktoru `__init__`, který je zavolán vždy při spuštění bota (respektive automaticky při vytvoření nového objektu ze třídy *MentionsNotifier*, který je však vždy vytvořen pouze jeden, a to při spuštění). Toto zajistí, že je seznam vždy aktuální.

Pro zápis je volána metoda *json.dump()*, která při každé změně zapíše celý slovník *emails* do souboru *emails.json*. Zároveň po celou dobu zůstává slovník uložený v proměnné, tudíž ho stačí během celého běhu programu načíst ze souboru pouze jednou.

Nevýhodou v tomto případě může být, že pokud by byl soubor *emails.json* jakkoli poškozený nebo obsahoval nesprávně zapsané či formátované hodnoty, celý skript se odmítne spustit, protože daný soubor nedokáže správným způsobem přečíst.

```
53 class MentionsNotifier:
54     def __init__(self) -> None:
55         with open("emails.json", "r") as file:
56             self.emails = json.load(file)
57
58     def subscribe(self, user_id: int, email: str) -> None:
59         with open("emails.json", "w") as file:
60             self.emails[str(user_id)] = email
61             json.dump(self.emails, file)
62
63     def unsubscribe(self, user_id: int) -> None:
64         if user_id in self.emails:
65             with open("emails.json", "w") as file:
66                 del self.emails[user_id]
67                 json.dump(self.emails, file)
```

Obrázek 24: implementace ukládání do souboru .json

3.6.6 !play_hangman

Tento příkaz slouží k zahájení hry hangman. Nejprve je vytvořen nový objekt třídy *Hangman* a jeho metodou *start_game* je vytvořena zpráva,

```
205 # --- hra hangman ---
206 hangman = Hangman()
207
208
209 @bot.command(name="play_hangman")
210 async def play_hangman(ctx: Context) -> None:
211     hangman.start_game(ctx.author.name)
212     displayed_word = ""
213     for i in range(len(hangman.word)):
214         displayed_word += hangman.word_letters[i] + " "
215     global MSG_ID
216     MSG_ID = await ctx.send("**hangman**\n"
217                             + "Hráč: " + str(hangman.player) + "\n"
218                             + "Hádaná písmena: " + "\n"
219                             + "Životy: " + str(hangman.lives) + "\n"
220                             + "Slovo: " + displayed_word)
221
```

Obrázek 25: vytvoření nové hry hangman

Metoda *start_game* načte slova ze souboru *words.txt* a uloží je do seznamu. Metodou knihovny *random* je poté vybráno náhodné slovo, které je naformátováno do správného tvaru.

Na výchozí hodnoty je nastaven počet životů a hádaná písmena. Podle počtu znaků je potom vygenerován řetězec ze znaků „–“, které se zobrazují místo zatím neuhádnutých písmen.

```
103 class Hangman:
104     def start_game(self, player) -> None:
105         with open("words.txt", "r", encoding="utf8") as file: # nacteni slov ze souboru
106             all_words = file.readlines()
107             self.word_unformatted = random.choice(all_words) # vyber slova
108             # odebrani \n pokud ho slovo obsahuje
109             if "\n" in self.word_unformatted:
110                 self.word = self.word_unformatted[:-1]
111             else:
112                 self.word = self.word_unformatted
113             print(self.word) # vypsani slova do konzole pro kontrolu
114             self.player = player
115             self.lives = 7
116             self.guesses = []
117             self.word_letters = []
118             for char in self.word:
119                 self.word_letters.append("- ")
```

Obrázek 26: metoda `start_game`

Pro účely debugování a kontroly se vždy při vytvoření nové hry do konzole vypíše slovo, které se má hádat. Především to slouží k vyzkoušení, zda fungují správně všechny varianty, jak může dopadnout hádání slova (správně, špatně, výhra, prohra).

3.6.7 !guess

Nejzajímavější, ale nejsložitější příkaz je potom tento příkaz. Jeho hlavním účelem je zpracovat uživatelský tip a vyhodnotit, zda se „trefil“, nebo ne. Další důležitým úkolem je zkontrolovat zbývající počet životů a počet neuhodnutých písmen, podle čeho se uživateli sdělí, zda vyhrál, prohrál nebo může pokračovat v hádání.

Úplně prvním krokem je smazání zprávy, ve které uživatel příkaz napsal, aby se zajistila větší přehlednost. Druhým krokem je potom kontrola, zda uživatel zadal pouze jedno písmeno (pomocí `len(letter) != 1`) a zda je zadaný znak písmeno (pomocí `letter.isalpha()`). Obě tyto podmínky jsou dány do logického součtu, což znamená, že pokud je jedna z nich porušena, příkaz dál s vyhodnocováním nepokračuje a do chatu napíše varovné hlášení.

```
223 @bot.command(name="guess")
224 async def guess(ctx: Context, letter: str) -> None:
225     await ctx.message.delete()
226     global MSG_ID
227     if len(letter) != 1 or not letter.isalpha():
228         await ctx.send("Hádejte pouze 1 písmeno.")
229     return
```

Obrázek 27: zahájení příkazu `!guess`, kontrola hádaného znaku

Pokud je znak v pořádku, je zavolána metoda *play()* objektu *hangman*, která vyhodnotí správnost tipu. Podle toho vrátí buď k číslo 0, 1 nebo 2 (0 = špatný tip, 1 = správný tip, 2 = písmeno již bylo hádáno). Poté je vytvořen řetězec hádaného slova a již tipovaných písmen, které jsou pro přehlednost odděleny mezerami.

```
232     return_code = hangman.play(letter)
233     guessed_letters = ""
234     displayed_word = ""
235     for i in range(len(hangman.word)):
236         displayed_word += hangman.word_letters[i] + " "
237     for i in range(len(hangman.guesses)):
238         guessed_letters += hangman.guesses[i] + " "
```

Obrázek 28: vytvoření řetězců a volání metody *hangman.play()*

Metoda *hangman.play()* nejprve zkontroluje, zda již nebylo písmeno hádáno. Pokud ano, vrátí kód 2, pokud ne, pokračuje se dál.

Poté je zjištěno, zda hádané slovo obsahuje tipované písmeno a pokud ano, vyhodnotí se, zda pouze jednou či víckrát. V prvním případě ho jednoduše nahradí v seznamu uhodnutých písmen („–“ na správné pozici se nahradí písmenem), v druhém případě je potřeba použít *for* cyklus v závislosti na počtu výskytů písmena ve slově. Po dokončení se vrací kód 1.

Pokud není splněna ani jedna z předchozích podmínek, znamená to, že tip byl špatný a ve slově se písmeno nevyskytuje. Hráči je proto odebrán 1 život a vrací se kód 0.

```
122     def play(self, letter) -> int:
123         if letter in self.guesses:
124             return 2 # pripad, kdy je jiz pismeno hadano
125         self.guesses.append(letter)
126         if letter in self.word: # kontrola zda je pismeno ve slove
127             k = self.word.count(letter)
128             if k > 1: # pokud je pismeno ve slove vickrat
129                 indices = [i.start() for i in re.finditer(letter, self.word)]
130                 for i in range(k):
131                     self.word_letters[indices[i]] = letter
132                 return 1
133             if k == 1: # pokud je pismeno ve slove jednou
134                 self.word_letters[self.word.find(letter)] = letter
135                 return 1
136         else: # pismeno ve slove neni
137             self.lives -= 1 # odebrani zivota
138             return 0
```

Obrázek 29: zpracování tipu

3.7 PEP8

Jedním z požadavků pro získání plného počtu bodů za splnění úlohy v KSI bylo dodržení pravidel psaní kódu PEP8. Jedná se o soubor pravidel týkajících se formátování, který by měl každý programátor při psaní svého kódu dodržovat, což zajistí čitelný kód, ve kterém se dá snadno orientovat. Jsou to pravidla, která nemají přímý vliv na funkčnost kódu, nicméně funkční a kvalitní kód je vždy lepší než funkční a nekvalitní kód.¹²

Pravidla jsou dostupná např. zde: <https://peps.python.org/pep-0008/>.

Tato pravidla jsem se snažil již během vývoje co nejvíce dodržovat a velmi mi s tím pomohl nástroj *pycodestyle*, který umí části kódu, kde je něco v rozporu s pravidly, najít a sdělit, co je špatně.

¹² TESLIA, Yuliia. *PEP8* [online]. [cit. 2023-04-07]. Dostupné z: <https://ksi.fi.muni.cz/ulohy/492>

4 Závěr

V této práci byl vytvořen v Pythonu skript, který dokáže obsluhovat bota na Discordu. Umožňuje botovi zpracovávat příkazy a příslušným způsobem na ně reagovat. Mezi funkce, které bot má, patří:

1. Zobrazení všech příkazů, které bot umí zpracovat
2. Zobrazení aktuálně populárních šablon pro tvorbu memů na stránce imgflip.com
3. Podle zadaného kódu šablony a dvou textů vytvořit meme pomocí veřejného API stránky imgflip.com a poslat ho do kanálu na Discordu
4. Možnost nechat si zasílat emailové oznámení o tom, že byl uživatel označen v jakémkoliv textovém kanálu, případně tuto možnost vypnout
5. Hraní hry „Hangman“ (v českém překladu „šibenice“) v rámci textového prostředí Discordu

Určitě stojí za zmínku různá vylepšení, která by mohla funkce bota (či kód samotný) posunout na vyšší úroveň. Jedna z nich je ukládání emailových adres pro zasílání upozornění do externího souboru, tudíž seznam těchto adres zůstane nezměněn i v případě restartu bota. Toto vylepšení mě napadlo v průběhu psaní popisu kódu a zdárně jsem ho implementoval. Za další nedostatek (a tedy příležitost pro zlepšení) pokládám to, že kód je místy zbytečně „ukecaný“ a opakuje se, což mě během původního psaní kódu nenapadlo řešit. Přispět k větší jednoduchosti a čitelnosti kódu by se dalo využitím více procedur a funkcí, které by nahradily opakující se části kódu, které se vyskytují především v kódu obsluhujícím hru hangman.

Odkaz na veřejný repozitář celého kódu:

https://github.com/petrambroz/bot_zaverecna_prace

Odkaz (pozvánka) na testovací server, kde je bot připojený:

<https://discord.gg/YAzy5XFf2u>

5 Seznam zdrojů

1. Historie programovacích jazyků. Fakulta informatiky Masarykovy Univerzity [online]. [cit. 2023-03-22]. Dostupné z: <https://www.fi.muni.cz/usr/jkucera/pv109/2000/xkrubova.htm>
2. History of Java Programming Language. UNext [online]. [cit. 2023-04-05]. Dostupné z: <https://u-next.com/blogs/java/history-of-java/>
3. LOPATKA, Adam. Discord bot příprava [online]. [cit. 2023-03-25]. Dostupné z: <https://ksi.fi.muni.cz/ulohy/512>
4. PROCHÁZKA, Karel. PIP správa balíčkov [online]. [cit. 2023-04-04]. Dostupné z: <https://ksi.fi.muni.cz/ulohy/504>
5. TESLIA, Yuliia. PEP8 [online]. [cit. 2023-04-07]. Dostupné z: <https://ksi.fi.muni.cz/ulohy/492>

6 Seznam obrázků

Obrázek 1: ukázka jednoduchého kódu v Javě	7
Obrázek 2: ukázka kódu v Pythonu se stejnou funkcionalitou jako v obrázku 1	7
Obrázek 3: registrační formulář	10
Obrázek 4: rozbalovací nabídka, tlačítko pro pozvání	11
Obrázek 5: okno pro tvorbu zvacího odkazu	11
Obrázek 6: vytvoření bota	12
Obrázek 7: výběr oprávnění.....	13
Obrázek 8: ukázka ze hry hangman	14
Obrázek 9: vytvoření virtuálního prostředí.....	15
Obrázek 10: aktivace virtuálního prostředí	16
Obrázek 11: složka virtuálního prostředí se všemi soubory v OS Windows 10.....	16
Obrázek 12: import balíčků.....	17
Obrázek 13: ukázka obsahu .env souboru, za = je umístěn napsán konkrétní údaj	18
Obrázek 14: příkaz seznam_prikazu	19
Obrázek 15: třída SeznamPrikazu a metoda create_message	20
Obrázek 16: příkazu !list_memes.....	20
Obrázek 17: vytvoření zprávy s populárními šablonami	21
Obrázek 18: požadavek na API a vrácení url odkazu na meme.....	21
Obrázek 19: zadání příkazu včetně argumentů, odpověď bota s memem	22
Obrázek 20: výsledek tvorby memu	22
Obrázek 21: třída MentionsNotifier.....	23
Obrázek 22: vytvoření seznamu označených uživatelů	23
Obrázek 23: odeslání emailu.....	24
Obrázek 24: implementace ukládání do souboru .json	25
Obrázek 25: vytvoření nové hry hangman.....	25
Obrázek 26: metoda start_game	26
Obrázek 27: zahájení příkazu !guess, kontrola hádaného znaku	26
Obrázek 28: vytvoření řetězců a volání metody hangman.play()	27
Obrázek 29: zpracování tipu	27