ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА TELEGRAM-БОТА З БАГАТОКОРИСТУВАЦЬКОЮ ГРОЮ У ЖАНРІ КІБЕРПАНК**»

Виконав: студент 5 курсу, групи ППЗ-51

спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

_____Якіменко Д.В._____

(прізвище та ініціали)

Керівник _Негоденко О.В._____

(прізвище та ініціали)

Рецензент_____

(прізвище та ініціали)

Нормоконтроль _____

(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення
О.В. Негоденко

“_____” _____2022 року

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Якіменко Дмитрію Вячеславовичу

(прізвище, ім‘я, по батькові)

1. Тема роботи: «Розробка Telegram-бота з багатокористувацькою грою у жанрі кіберпанк»

Керівник роботи Негоденко О.В., к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “18” лютого 2022 року №

2. Строк подання студентом роботи 03.06.2022.
3. Вихідні дані до роботи:
   3.1. Середовище розробки PyCharm Community Edition 2021.3.3
   3.2. Алгоритм дії бота
   3.3. Telegram Bot API
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   4.1. Аналіз та порівняння існуючих прототипів
   4.2. Дослідження програмних засобів для розробки бота
   4.3. Програмна реалізація бота
   4.4. Приклади використання та тестування системи
   4.5. Висновки

5. Перелік графічного матеріалу.

5.1. Титульний слайд
5.2. Мета, об’єкт та предмет дослідження
5.3. Терміни
5.4. Аналіз існуючих рішень
5.5. Технічне завдання
5.6. Програмні засоби реалізації
5.7. Схема роботи телеграм-бота
5.8. Екранні форми боту
5.9. Висновки

6. Дата видачі завдання: 11.04.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04-15.04	
2	Дослідження існуючих інструментів для автоматизації тестування ПЗ	16.04-18.04	
3	Проектування архітектури системи	19.04 – 24.04	
4	Висновки, оформлення роботи	26.04 – 08.05	
5	Розробка демонстраційних матеріалів	09.05-15.05	
6	Попередній захист роботи	16.05-01.06	
7	Здача роботи	03.06	

Студент _____ Якіменко Д.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Негоденко О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 63с., 61 рис., 5 джерел.

Ключеві слова: Telegram, Python, pyTelegramBotAPI, TinyDB, Pycharm, чат-бот.

Об'єкт дослідження – процес комунікації на платформі Telegram за допомогу чат-бота.

Предмет дослідження – методи та засоби створення чат-бота на платформі Telegram.

Мета роботи – підвищити комунікацію між користувачам за допомогою чат-бота на платформі Telegram.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати технічні засоби, що використовуються для розробки та обрати необхідні для створення ігрового бота.

3. Розробити вимоги до ігрового бота на основі аналізу переваг та недоліків існуючих ботів.

4. Спроектувати та розробити нового ігрового бота на основі аналізу потреб користувачів.

4. Провести тестування гри.

Практичне значення отриманих результатів полягає у написанні функціоналу для роботи з жанром «Кіберпанк» з використанням фреймворку pyTelegramBotAPI та мови програмування Python.

В роботі розглянуто основні етапи створення ігор і досліджено можливості технічних засобів для розробки гри.

Розроблено модель застосунку та функціональні вимоги до гри та підібрані методи реалізації механік.

Галузь використання – бота може використовувати будь-який користувач Telegram у особистих повідомленнях бота та чатах де є бот.

ЗМІСТ

Вступ

Чат-боти революціонізують спосіб взаємодії людей із технологіями. В останні роки їхня простота та низька вартість сприяли поширенню в різних галузях і галузях.

Чат-боти часто рекламують як революцію у тому, як користувачі взаємодіють з технологіями та бізнесом. Вони мають досить простий інтерфейс у порівнянні з традиційними додатками, оскільки вони вимагають від користувачів лише спілкування в чаті, а чат-боти повинні розуміти і робити все, що від них вимагає користувач, принаймні теоретично.

Багато галузей переводять обслуговування клієнтів на системи чат-ботів. Це пов’язано з величезним зниженням вартості в порівнянні з реальними людьми, а також через надійність і постійну доступність. Чат-боти забезпечують певну підтримку користувачів без істотних додаткових витрат.

Сьогодні чат-боти використовуються в багатьох сценаріях, починаючи від дрібних завдань, таких як відображення даних про час і погоду, до більш складних операцій, таких як елементарна медична діагностика та спілкування/підтримка клієнтів. Ви можете створити чат-бота, який допоможе вашим клієнтам, коли вони задають певні запитання про ваш продукт, або ви можете створити чат-бота особистого помічника, який зможе виконувати основні завдання і нагадуватиме вам, коли настав час відправитися на зустріч або в спортзал.

Існує багато варіантів, де ви можете розгорнути свого чат-бота, і одним з найпоширеніших варіантів використання є платформи соціальних мереж, оскільки більшість людей використовує їх на регулярній основі. Те ж саме можна сказати і про програми для обміну миттєвими повідомленнями, хоча з деякими застереженнями.

Telegram є однією з найпопулярніших платформ миттєвого обміну повідомленнями сьогодні, оскільки вона дозволяє зберігати повідомлення в хмарі, а не тільки на вашому пристрої, і має гарну підтримку багатьох платформ,

оскільки ви можете мати Telegram на Android, iOS, Windows і майже будь-яка інша платформа, яка підтримує веб-версію. Створення чат-бота в Telegram досить просте і вимагає кількох кроків, виконання яких займає дуже мало часу. Чат-бот можна інтегрувати в групи і канали Telegram, а також він працює самостійно.

Об'єкт дослідження – процес комунікації на платформі Telegram за допомогу чат-бота.

Предмет дослідження – методи та засоби створення чат-бота на платформі Telegram.

Мета роботи – підвищити комунікацію між користувачам за допомогою чат-бота на платформі Telegram.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати технічні засоби, що використовуються для розробки та обрати необхідні для створення ігрового додатку.
3. Розробити вимоги до ігрового додатку на основі аналізу переваг та недоліків існуючих додатків
4. Спроектувати та розробити новий додаток на основі аналізу потреб користувачів.
4. Провести тестування гри.

Практичне значення отриманих результатів полягає у написанні функціоналу для роботи з жанром «Кіберпанк» з використанням фреймворку pyTelegramBotAPI та мови програмування Python.

Розроблено модель застосунку та функціональні вимоги до гри та підібрані методи реалізації механік.

Галузь використання – бота може використовувати будь-який користувач Telegram у особистих повідомленнях бота та чатах де є бот.

1.АНАЛІЗ ТА ВИДІЛЕННЯ ПЕРЕВАГИ ТЕЛЕГРАМ-БОТА З КІБЕРПАНК-ГРОЮ

1.1 Месенджери і Telegram

Тепер, коли використання смартфонів зростає, додатки для обміну миттєвими повідомленнями стали необхідністю для користувачів. Зараз доступно багато безкоштовних програм миттєвого обміну повідомленнями з розвитком технологій, які продемонстрували видатні вдосконалення. Ці послуги дозволяють людям спілкуватися з друзями за допомогою текстових, телефонних дзвінків, відео, спільних файлів, у групах чи особистих повідомленнях, і підтримувати з ними зв’язок навіть на міжнародному рівні. Але дуже мало хто здобув популярність і визнання. Останні дослідження показали, що найпопулярнішими месенджерами є WhatsApp, Viber і Telegram.

Telegram — популярний сервіс обміну повідомленнями, який базується на платформі з відкритим кодом. На додаток до повністю безкоштовного сервісу без будь-яких платежів, він також пропонує середовище без реклами з чистим і швидким інтерфейсом. Telegram був створений у серпні 2013 року підприємцем російського походження Павлом Дуровим, але «російський WhatsApp» (хоча сам Дуров заперечує такий епітет, бо сам втік з росії) становить серйозну конкуренцію гігантам індустрії WhatsApp і Viber. Telegram стає настільки популярним, що він стає найбільш завантажуваним додатком для обміну повідомленнями в магазині Google Play. Telegram також здобув статус найпопулярнішої соціальної мережі серед безкоштовних завантажуваних програм у більш ніж сорока країнах, включаючи Німеччину та Сполучені Штати, обігнавши інші соціальні мережі, такі як Facebook, WhatsApp, WeChat та Kik. У Telegram легко зареєструватися та використовувати. Він має багато подібності з WhatsApp з точки зору роботи з ідентифікатором користувача та контактом. Номер телефону використовується для первинної ідентифікації користувача.

Щойно користувач встановлює клієнт, він/вона може спілкуватися з будь-яким номером у своєму списку контактів, який вже встановив клієнт.

Однак Telegram може запропонувати більше. Будь-який користувач може створити ім'я користувача як унікальний ідентифікатор у Telegram, таким чином дозволяючи іншим, хто знає це ім'я користувача, зв'язуватися безпосередньо без необхідності знати номер телефону відповідної сторони. Крім того, додавання чужого імені користувача до списку контактів Telegram не відкриє автоматично номер телефону. Враховуючи, що номер телефону є такою конфіденційністю для кількох людей, ця функція є чудовим способом покращити захист конфіденційності. Крім того, архів спілкування зберігається в хмарі, тому користувачам не доведеться переживати, що вони втратили свої чати після зміни телефону. Серед переваг телеграму також стікери. Це колекції зображень, які можна використовувати в чатах, окрім тексту. Телеграм дозволяє обмінюватися документами в більш різноманітних типах файлів, ніж WhatsApp, а також обмінюватися зображеннями та відео без обмеження розміру. Telegram — це багатоплатформенна програма, яка може працювати на ОС Android, iOS, Windows Phone, Mac і Windows. Крім того, до облікового запису Telegram можна отримати доступ з кількох пристроїв.

1.2 Чат-боти

Нині в месенджерах популярна така річ як чат-боти. Сам термін «чат-бот» придумав Майкл Молдін в 1994р. для опису програм для розмов.

Чат-бот - це програма, яка, отримуючи інформацію від користувача, формує коректні, логічно обґрунтовані відповіді. Нині чат-боти можуть використовуватися не тільки для балачок з компьютером, як це задумувалося початково, але і для багатьох інших функцій. Серед можливих застосувань:

1. Чат-боти для консультації та підтримки клієнтів.
2. новинні чат-боти;
3. ігрові чат-боти;

4. Чат-боти для магазинів, сервісів, доставок;
5. Чат-боти для модерування чатів.
6. Та багато іншого

Чат-бот як віртуальний співрозмовник має базу даних, яка представляє собою набори можливих питань користувача та відповідей для них. Найбільш поширеними варіантами для отримання потрібної відповіді є ключові слова, збіг фрази, або команди.

1.3 Аналіз існуючих ботів для ігор на платформі Telegram

Недалекоглядно було б припустити, що оскільки основним завданням месенджера є спілкування, то в даній ніші не знайдеться місця і для ігор в Телеграм.

Кількість ігрових ботів вже давно перевалило за сотні, і, відповідно, тематики та складність розробки суттєво різняться.

І з усього цього розмаїття варто створити своєрідний топ з чотирма ігровими ботами, які заслужили найбільше розташування користувачів.

1.3.1 "Village Game"

"Село" - це найпопулярніший gamebot на просторах "Telegram", що зібрав вже не одну сотню тисяч установок.

Як і у всіх іграх тематики ферма чи село, основним завданням є збирання врожаю та розвитку свого поселення, але у «Village Game» реалізована можливість боротися і з іншими гравцями.

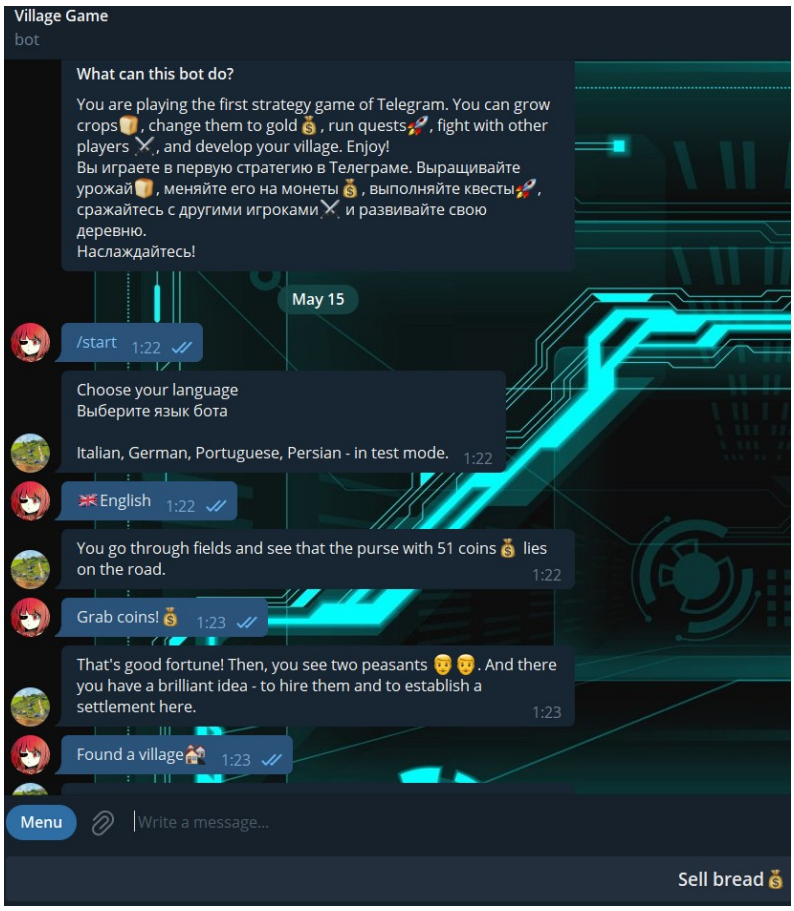


Рисунок 1.3.1 – приклад гри у боті "Village Game".

Після старту перші кроки стандартно здійснюються з підтримкою бота, далі ж гравець обирає самостійно, як він проводитиме політику розвитку.

Основний ресурс у цій грі - це робітники. Отримати їх можна за допомогою запрошення в гру друзів та знайомих, використовуючи реферальне посилання. До речі, це й сприяло такій кількості користувачів у грі.

Також варто відзначити нескладні текстові квести, які дають змогу скрасити пару – трійку хвилин очікування продажу врожаю.

1.3.2 «Quizarium»

Концепція Quizarium проста - бот ставить запитання, а гравці повинні знайти правильну відповідь якомога швидше протягом однієї хвилини. Бот може час від часу видавати підказки, щоб допомогти гравцям. Однак швидкість важлива в цій груповій грі Telegram, тому що чим раніше ви отримаєте правильну

відповідь, тим більше очок отримаєте. Очок коливається від одного до п’яти, і виграє той, хто набере найбільшу кількість очок в кінці гри!

Рисунок 1.3.2 – приклад гри у боті «Quizarium».

1.3.3 «Arena Game RPG»

Ігри Telegram не забов’язані бути лише міні-іграми та картковими іграми. В цій грі можна створювати власних персонажів, керувати їхніми навичками та інвентарем, а також налаштовувати свого персонажа на перемогу у битві гравця проти гравця. У боях проти інших гравців ви по черзі вибираєте місце атаки, наприклад, голова, груди або ноги. Кількість шкоди, яку ви завдаєте, залежить від того, наскільки ваш персонаж налаштований. Воїн, який виграє битву, отримує більше очок навичок, які можна використати для покращення атаки чи захисту.

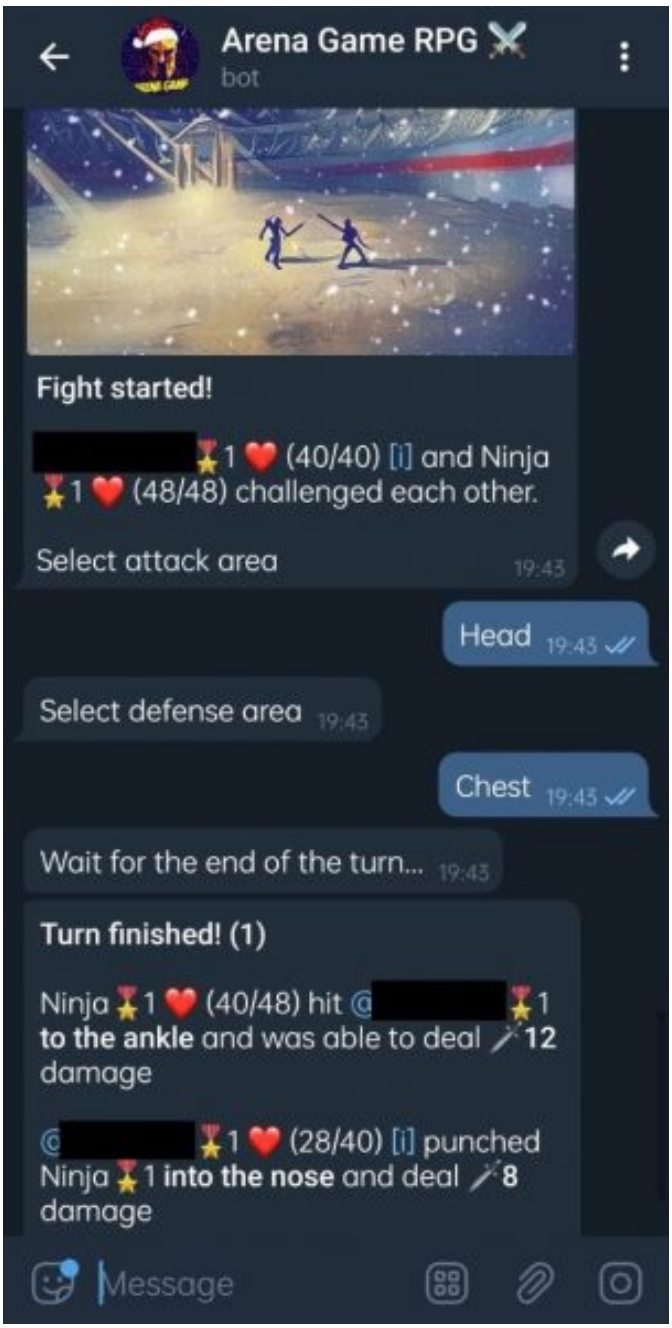


Рисунок 1.3.3 – приклад гри у боті «Arena Game RPG».

1.3.4 «Werewolf»

Гра «Перевертень» на подобі відомої гри «Мафія» та інших ігор подібного жанру. Гравці перебувають в селі, де ховаються таємничі істоти. У групі деякі гравці гратимуть за перевертнів, які збираються вбити всіх жителів села до закінчення гри. Решта гравців будуть селянами. Жителям села потрібно працювати разом, щоб з’ясувати, хто такі перевертні, щоб «лінчувати» їх, перш ніж вони вб’ють все населення. Наприкінці кожного раунду селяни матимуть

можливість проголосувати за того, ким, на їхню думку, є перевертень. Ви можете вигнати когось на основі більшості голосів, тому дуже важливо вибрати потрібну людину, за яку проголосувати (якщо ви не перевертень)!

Бот делегує час вовкам і гравцям, щоб обговорити та зробити свій вибір голосування в чаті.

Рисунок 1.3.4 – приклад гри у боті «Werewolf».

1.4 Таблиця порівняння ботів

Таблиця 1 – порівняння бота з аналогами

	Cybermancy	Village Game	Quizarium	Arena Game RPG	Werewolf
Особисті повідомлення	+	+	-	+	+
Чати	+	-	+	-	+
Вибір мови	+	+	+	+	-
Кіберпанк	+	-	-	-	-

2. РОЗРОБКА БОТА ТА ТЕХНОЛОГІЇ

2.1 Програмні засоби реалізації

2.1.1 Telegram Bot API

Telegram Bot API — це інтерфейс на основі HTTP, створений для розробників, які розроблюють ботів для Telegram.

Боти — це сторонні програми, які працюють через Telegram. Користувачі можуть взаємодіяти з ботами, надсилаючи їм повідомлення, команди та вбудовані запити. Боти керуються за допомогою HTTPS-запитів до API бота Telegram.

По суті, боти Telegram — це спеціальні облікові записи, для налаштування яких не потрібен додатковий номер телефону. Користувачі можуть взаємодіяти з ботами двома способами:

- Надсилати повідомлення та команди ботам, відкриваючи з ними чат або додаючи їх до груп.
- Надсилати запити безпосередньо з поля введення, ввівши @нікнейм бота та запит. Це дозволяє надсилати вміст від вбудованих ботів безпосередньо в будь-який чат, групу чи канал.

Повідомлення, команди та запити, надіслані користувачами, передаються програмному забезпеченню, запущеному виділених серверах де боти і знаходяться. Сервера-посередники Telegram обробляють за розробників ботів усе шифрування та зв’язок із Telegram API. Розробники спілкуються з цим сервером через простий HTTPS-інтерфейс, який пропонує спрощену версію API Telegram.

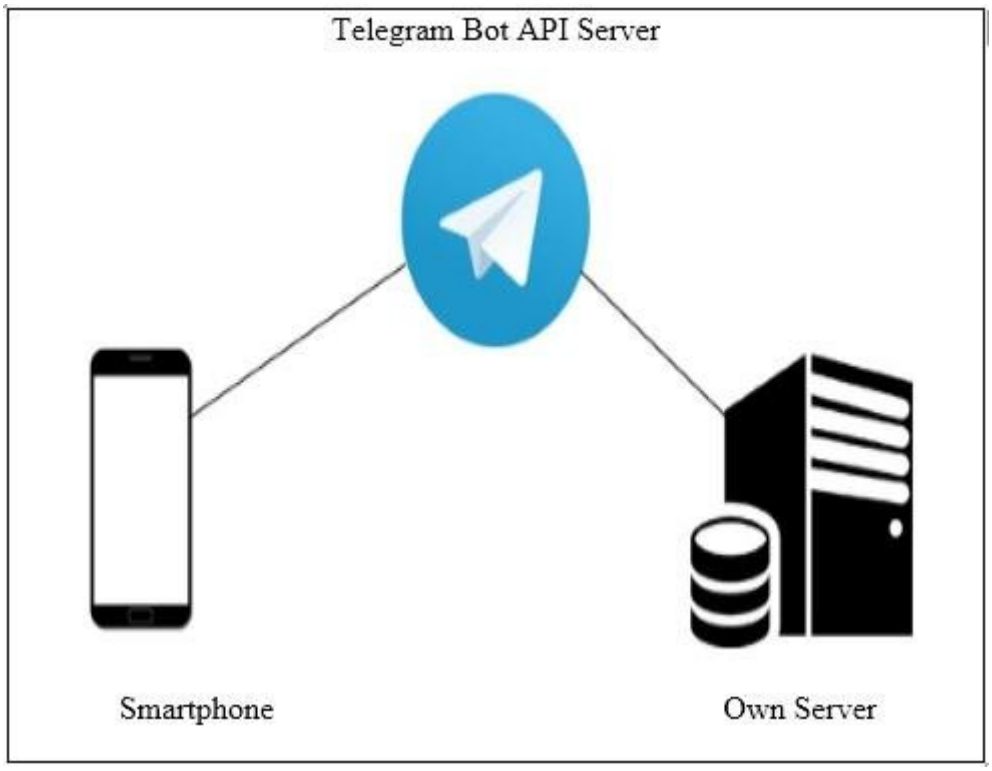


Рисунок 2.1.1.1 – проста схема взаємодії користувачів с серверами бота.

Є 2 способи взаємодії бота та серверів Telegram:

- «Long polling» - при якому бот сам постійно «питає» сервери телеграм на наявність нових запитів та отримує їх у відповідь.
- «Webhook» - більш складна система при якій сервери телеграм самі посилають запити до бота, а отже робить непотрібним постійне пінгування зі сторони бота. Мінус цього способа у тому, що для цього треба підіймати повноцінний сервер, налаштовувати сертифікати та багато іншого.

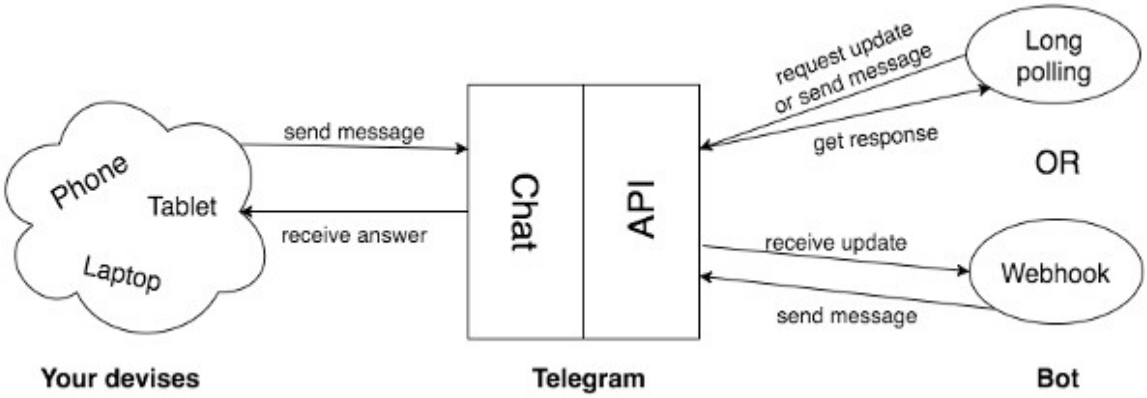


Рисунок 2.1.1.2 – схема типів взаємодії бота з серверами телеграму.

Для бота був обраний перший варіант, Long Polling. Це базовий варіант, який не потребує майже нічого, навідміну від вебхуків.

2.1.2 pyTelegramBotAPI

Проста та найпопулярніша реалізація Telegram Bot API для мови програмування Python.

2.1.3 Python

Python — це багатопарадигмальна інтерпретована мова програмування. Об’єктно-орієнтоване програмування та структурне програмування повністю підтримуються, і багато його функцій підтримують функціональне програмування та аспектно-орієнтоване програмування (включаючи метапрограмування та метаоб’єкти (магічні методи)). Багато інших парадигм підтримуються за допомогою розширень, включаючи проектування за контрактом і логічне програмування.

Python використовує динамічний тип типізації та комбінацію підрахунку посилань і збирача сміття, що визначає цикл, для управління пам’яттю. Він використовує динамічне визначення імен (пізнє прив’язування), яке зв’язує імена методів та змінних під час виконання програми.

Замість того, щоб вбудовувати всю свою функціональність у своє ядро, Python був розроблений так, щоб бути дуже розширюваним за допомогою модулів. Ця компактна модульність зробила його особливо популярним як засіб додавання програмованих інтерфейсів до існуючих програм.

2.1.4 TinyDB

База даних TinyDB – це проста NoSQL база даних з відкритим кодом написана на Python. Особливості TinyDB:

- **Компактна:** поточний вихідний код містить 1800 рядків коду (з приблизно 40% документації) і 1600 рядків тестів.
- **Документо-орієнтовна:** як і у MongoDB, можна зберігати будь-який документ (представлений як словник) у TinyDB.
- **Простота:** TinyDB розроблено, щоб бути простим у використанні, надаючи простий і чистий API.
- **Написаний на чистому Python:** TinyDB не потребує ані зовнішнього сервера (наприклад, PyMongo), ані будь-яких залежностей від PyPI.
- **Розширювана:** можна легко розширити TinyDB, написавши нові сховища або змінити поведінку сховищ за допомогою проміжного програмного забезпечення.

2.1.5 PyCharm

PyCharm — це інтегроване середовище розробки (IDE), яке використовується в програмуванні, спеціально для Python. Його розробляє чеська компанія JetBrains (раніше відома як IntelliJ). Він забезпечує аналіз коду, графічний налагоджувач, інтегрований модуль тестування, інтеграцію з системами контролю версій і підтримує веб-розробку за допомогою Django, а також науку про дані з Anaconda.

2.2 Розробка бота

2.2.1 Рєстрація бота через BotFather

Першим кроком у створенні бота є реєстрація аккаунту бота у телеграмі через офіційного влаштованого бота BotFather, який використовуються для створення та налаштувань аккаунтів ботів.

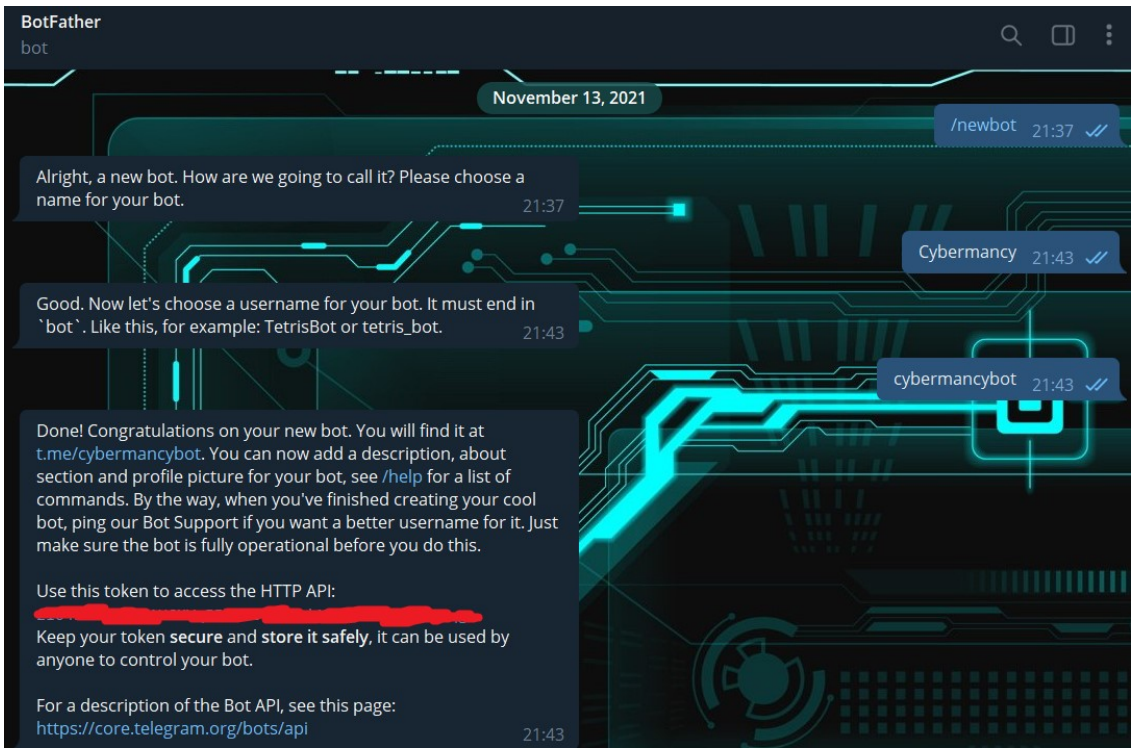


Рисунок 2.2.1.1 – первинна реєстрація бота в Telegram (Червоним замальований унікальний токен бота, який використовується для доступу до аккаунта бота).

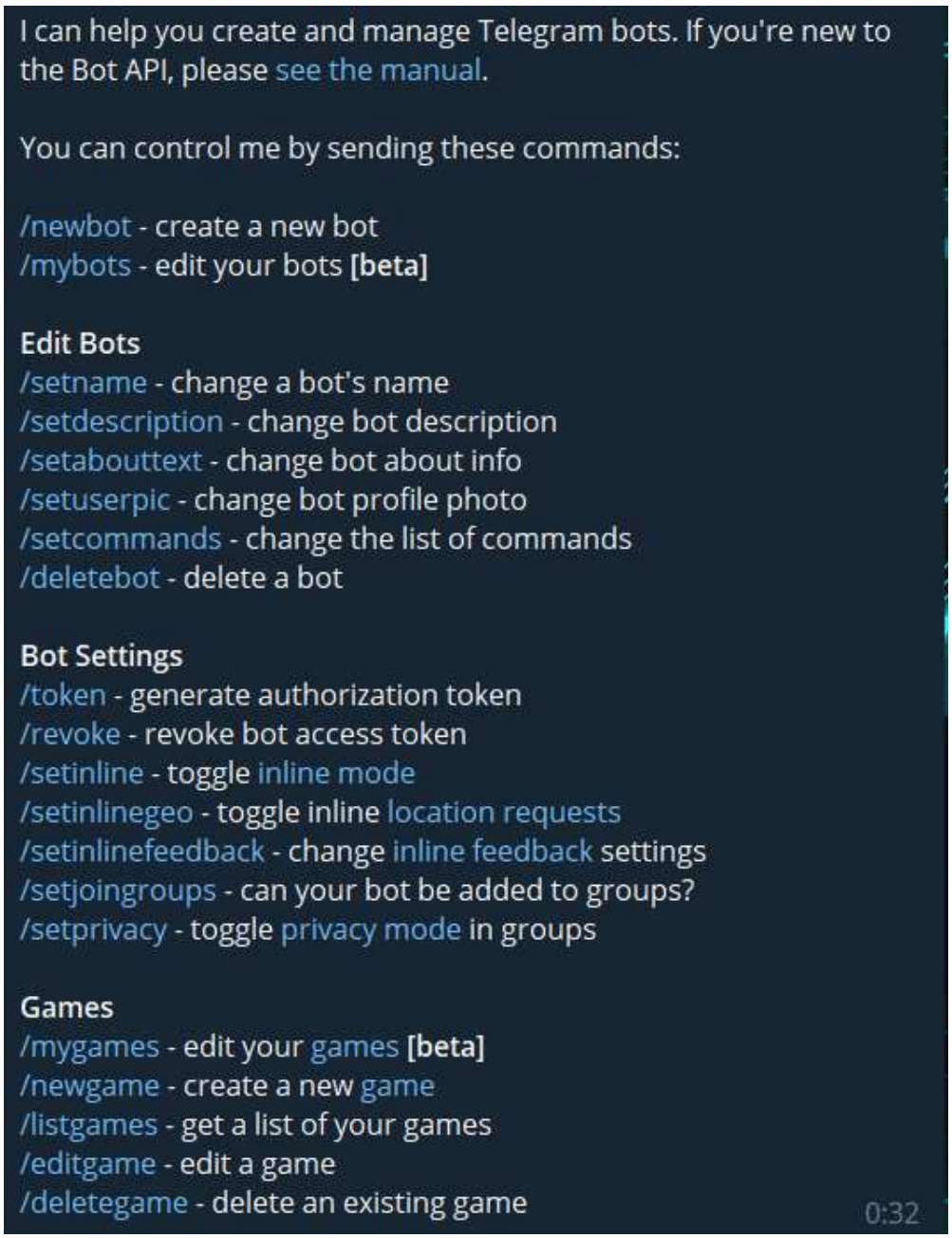


Рисунок 2.2.1.2 – повний перелік команд бота BotFather.

2.2.2 Інтерфейс програми

Користувачі можуть взаємодіяти з ботом кількома способами:

1. Через текстові повідомлення.
2. Через inline-режим, вводячи назву бота та отримуючи перелік можливих взаємодій.

У проекті наразі використовується лише текстовий варіант взаємодії. Текстових способій взаємодії є теж 2:

1. Через команди (слово англійскими літерами з символом «/» на початку, наприклад «/start»)
2. Через звичайний текст.

Серед цих способів був обраний спосіб взаємодії за допомогою команд, тому що це не тільки визначає реалізацію боту, але й визначає фільтрацію запитів зі сторони серверів телеграм. У випадку з командами боту треба буде оброблювати сюттєво менше повідомлень, на відміну від обробки взагалі всіх повідомлень, що важливо для швидкодії.

Рисунок 2.2.2.1 – приклад меню вибору команд. Відображувані команди налаштовуються за допомогою BotFather.

Таблиця 2 – реалізовані команди:

Назва	**Особисті повідомлення**	**Чати**	**Опис**
/start	+	-	Обов’язкова команда для початку роботи з ботом та створення персонажа.
/stats	+	-	Команда для відображення характеристик та навичок персонажа, і їх покращення.
/help	+	+	Команда для виводу посилання на інтернет-сторінку з всією потрібною інформацією про

			бота.
/lang	+	+	Команда для вибору мови у особистих повідомленнях, або у чаті.
/join	-	+	Команда для переміщення до якого-небудь чату, бо персонаж може знаходитись лише в одному чаті в будь який момент часу.
/work	+	+	Піти на роботу, щоб заробити ігрову валюту. Можна використати лише раз на день.
/rest	+	+	Команда для відпочинку, щоб трохи відновити сили та здоров'я. Можна використати лише раз на день.
/corp	-	+	Команда для взаємодій з корпораціями.
/stopthebot	+	-	Команда виключно для розобника для відключення бота.

```
@bot.message_handler(func=base_filter, chat_types=['private', 'supergroup'], commands=['rest'])
def rest_command(message):
    player = Player(message.from_user.id)
    if(player.cooldowns["rest"]==True):
        lang.main.you_already_rested(message)
        return
    player.staminaChange(50)
    player.healthChange(10)
    player.save()
    bot.reply_to(message, lang.main.basic_rest(message))
```

Рисунок 2.2.2.2 – приклад функції обробки команди у файлі main.py

На зображенні вище на прикладі оборобки команди «/rest» зображений синтаксис обробників команд за допомогою pyTelegramBotAPI.

Конструкція «@bot.message_handler()» - декоратор який маркерує функцію «rest_command()» як хендлер для обробки команд. Існують і інші типи хендлерів для обробки різних типів запитів.

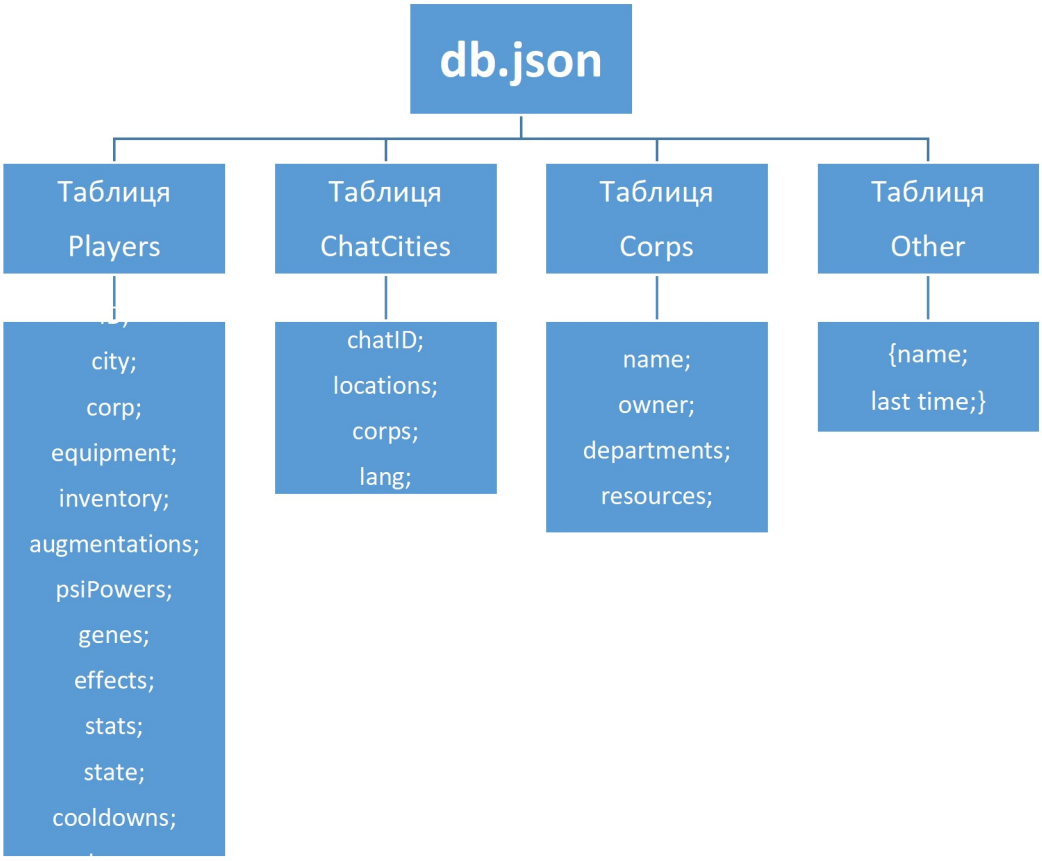
«func=base_filter» - аргумент декоратора, який визначає функцію-фільтр, яка перевіряє чи підходить повідомлення для цієї конкретної функцію. Перевірка фільтром виконується вже після перевірки на тип команди та на тип чата. Функція фільтру повинна повертати True, або False в залежності від того підходить запит для цього обробника чи ні.

«chat_types=['private', 'supergroup']» - аргумент декоратора, який перевіряє тип чата з якого прийшов запит. У данному випадку це приватні повідомлення та «супергрупа». Супергрупа відрізняється від звичайної групи тім, що може буди публічною, та відображає історію чата для всіх користувачів, істрорія супергрупи зберігається на сервері, а не тільки на девайсах користувачів.

«commands=['rest']» - аргемент декоратора, який визначає команду яку цей обробник повинен обробляти.

«rest_command(message)» - функція-обробник, яка отримує запит аргументом. Нічого не повертає.

2.2.3 Проектування бази даних бота



Для реалізації бота виникла необхідність у використанні бази даних для збереження разноманітної інформації про користувачів та не тільки.

Для цієї задачі було обрана NoSQL-база даних TinyDB. Зберігання даних у JSON краще вього підходить для зберігання вкладених даних, наприклад списків, словників, тощо. Зберігання даних у JSON дуже схоже на те як дані зберігаются в самих програмах мови Python.

Для проекту у базі даних було створено 4 таблиці:

- Players – таблиця, яка містить інформацію про гравців. Їх ID, місце знаходження, всі види спорядження, всі їх параметри як то здоров'я, навички, та багато іншого.
- ChatCities – таблиця, яка містить інформацію про чати куди був доданий бот. Зберігає інформацію про id чатів, згенеровані при додаванні бота у чат «локації», корпорації зараєстровані там, тощо.
- Corps – таблиця, яка містить інформацію про «корпорації». Корпорації це дещо на подобі кланів, об'єднання гравців. Участь у корпорації дає ряд своїх бонусів. Таблиця містить назву корпорації, рівні покращень корпорації та ресурси.
- Other – містить всю іншу інформацію яку треба зберігати, але для якої немає підходящих таблиць. На даний момент містить лише інформацію необхідну для роботи регулярних завдань ботом.

```
db = TinyDB('db.json')
PlayersTable = db.table("Players")
ChatCitiesTable = db.table("ChatCities")
CorpsTable = db.table("Corps")
OtherTable = db.table("Other")
```

Рисунок 2.2.3.1 – код для створення та підключення до бази даних та таблиць.

```
def __init__(self, ID: int):
    self.ID = ID
    player = PlayersTable.search(Query().ID == self.ID)
    if(player != []):
        player=player[0]
        self.city = player["city"]
        self.corp = player["corp"]
        self.equipment = player["equipment"]
        self.inventory = player["inventory"]
        self.augmentations = player["augmentations"]
        self.psiPowers = player["psiPowers"]
        self.effects = player["effects"]
        self.stats = player["stats"]
        self.state = player["state"]
        self.cooldowns = player["cooldowns"]
        self.lang = player["lang"]
```

Рисунок 2.2.3.2 – приклад використання бази даних. У даному випадку створення об’єкта гравця за допомогою збереженої у базі даних інформації.

```
def save(self):
    p = {"ID": self.ID,
        "city": self.city,
        "corp": self.corp,
        "equipment": self.equipment,
        "inventory": self.inventory,
        "augmentations": self.augmentations,
        "psiPowers": self.psiPowers,
        "genes": self.genes,
        "effects": self.effects,
        "stats": self.stats,
        "state": self.state,
        "cooldowns": self.cooldowns,
        "lang": self.lang}
    if(not PlayersTable.contains(Query().ID == self.ID)):
        PlayersTable.insert(p)
    else:
        PlayersTable.update(p, Query().ID == self.ID)
```

Рисунок 2.2.3.3 – приклад приклад збереження інформації у базу даних. У даному випадку це збереження інформації про гравця з об’єкта. У кінці можна бачити вибір між створенням та поновленням інформації у таблиці.

2.2.4 Серверна частина програми

Серверна частина бота повинна виконувати такі функції:

- Приймати запити від Telegram та оброблювати їх.
- Зберігати потрібні дані у базу даних, та діставати їх звідти.
- Надсилати повідомлення до Telegram.
- Виконувати код запланований по часу.

```
if __name__ == '__main__':
    atStart()
    thr = threading.Thread(target = scheduled_tasks)
    thr.daemon = True
    thr.start()
    while True:
        try:
            if(do_pulling):
                bot.polling(non_stop=True)
        except Exception as E:
            #print(E)
            if(E.args[0]=="stopthebot"):
                break
            else:
                print(traceback.format_exc())
        time.sleep(1)

    print("end")
```

Рисунок 2.2.4.1 – зображення центральної частини програми.

На відміну від багатьох інших мов програмування (наприклад C++, Java, C#) Python не має центральної функції яка викликається при старті програми і представляє саму програму. Замість цього у Python виконується весь центральний файл цілком.

Для того щоб імітувати поведінку функції main() з інших мов програмування у Pyhton використовується конструкція «if __name__ == '__main__':», яка є перевіркою того чи відкритий цей файл як програма, а не підключений до іншого коду як бібліотека. Тому цей код виконується тільки якщо безпосередньо почати програму з цього файлу, що і робить цей файл центральним у програмі.

Функція «atStart()» була створена як функція для виконання при кожному запуску бота.

```
def atStart():
    if(not CorpsTable.contains(Query().name == "Megacorp")):
        developer = Player(dev_id)
        megacorp = Corp("Megacorp", developer.ID)
        developer.corp = "Megacorp"
        developer.save()

        megacorp.budgetChange(100000)
        megacorp.influenceChange(1000)
        #megacorp.departments["R&D"] = 10  #закоментировано для тестов
        #megacorp.departments["Отдел кадров"] = 10
        #megacorp.departments["Служба охраны"] = 10
        #megacorp.departments["Химические лаборатории"] = 10
        #megacorp.departments["Отдел продаж"] = 10
        #megacorp.departments["Заводы"] = 10
        #megacorp.departments["Датацентр"] = 10
        #megacorp.departments["Связи"] = 10
        megacorp.save()

        OtherTable.insert({"name": "everyday cooldown", "last time": datetime.date.today().isoformat()})
```

Рисунок 2.2.4.2 – функція «atStart()».

На даний момент весь вміст функції призначений для виконання при першому запуску бота для спеціальних налаштувань, а також після кожної очистки бази даних (під час розробки, або у інших випадках). Перевірка на те чи перший раз був запущений бот була реалізована за допомогою перевірки бази даних на наявність у таблиці Corps корпорації «Megacorp», створюється у цій же функції та є спеціальною базовою корпорацією на яку працюють всі гравці які ще не вибрали собі корпорацію за невигідним тарифом, та яка зі старту має дуже багато ресурсів та все що треба.

Також у цій функції створюється одна єдина запис у таблицю Other, яка зберігає іншормацію про те коли останній раз було виконано обнуляння перезарядок щоденних речей, наприклад того чи ходив сьогодні гравець на роботу (інформація про це зберігається для кожного гравця).

```
thr = threading.Thread(target = scheduled_tasks)
thr.daemon = True
thr.start()
```

Рисунок 2.2.4.3 – шматок коду у main з оголошенням потоку для виконання запланованих завдань.

Ці 3 рядки коду створюють та налаштовують на можливість знищення потік для виконання запланованих завдань. Код потоку:

```
def scheduled_tasks():
    while True:
        if(OtherTable.search(Query().name == "everyday cooldown")[0]["last time"] != json.dumps(datetime.date.today().isoformat())):
            OtherTable.update({"last time": json.dumps(datetime.date.today().isoformat())}, Query().name == "everyday cooldown")
            PlayersTable.update({"work": False}, Query().cooldowns.work == True)
            PlayersTable.update({"rest": False}, Query().cooldowns.rest == True)
            print("Everyday task is done.")
        time.sleep(30)
```

Рисунок 2.2.4.4 – код функції запланованих завдань.

Дана функція виконується поза основним циклом програми і раз в 30 секунд перевіряє чи не наступив новий день для того, щоб обнулити всі перезарядки для речей які можуть бути зроблені 1 раз в день. Після виконання коду у консоль виводится рядок про те, що щоденне завдання виконане.

```
while True:
    try:
        if(do_pulling):
            bot.polling(non_stop=True)
    except Exception as E:
        if(E.args[0]=="stopthebot"):
            break
        else:
            print(traceback.format_exc())
    time.sleep(1)

print("end")
```

Рисунок 2.2.4.5 – основний цикл програми.

Остання частина аналогу main() у коді бота, основний цикл програми. «bot.polling(non_stop=True)» запускає пулінг – бескінечне пінгування ботом серверів телеграма для отримання запитів. У pyTelegramBotAPI раньше була тільки ця функція для пулінгу, але зараз є bot. infinity_polling(), який є обгорткою над bot.polling() з обробкою помилок, який продовжить роботу програми навіть після помилок опрацювання запитів. Але було вирішено відмовитися від цієї обгортки та реалізувати обробку помилок вручну для того, щоб реалізувати деякий свій код у цьому циклі обробки помилок.

Насправді бот може працювати і без пулінгу, але тоді він не зможе приймати будь які повідомлення. Але він все ще зможе відправляти повідомлення, якщо це потрібно. Такий шлях реалізації дуже рідкісний і лише для деяких дуже простих та спеціалізованих ботів.

Також тут в оброблювачі помилок наявний вивід помилок у повному вигляді у консоль зі стектрейсом а також реалізація вимкнення бота у випадку використання розробником команди /stopthebot

```
@bot.message_handler(func=dev_filter, chat_types=['private'], commands=['stopthebot'])
def stopthebot_command(message):
    raise Exception("stopthebot")
```

Рисунок 2.2.4.6 – простий код обробника команди /stopthebot з фільтром для розробника. Якщо розробник натисне цю команду, то обробник створить помилку з кодовим текстом, яка зупинить бота.

Як вже згадувалось вище, кожен оброблювач команд має свій фільтр. Ось всі наявні фильтри у проекті:

```
def base_filter(message):
    if(not PlayersTable.contains(Query().ID == message.from_user.id)):
        bot.reply_to(message, lang.main.say_start(message))
        return False
    return True

def dev_filter(message):#
    if((not PlayersTable.contains(Query().ID == message.from_user.id)) and (message.from_user.id == dev_id)):
        return False
    return True
```

Рисунок 2.2.4.7 – функції-фільтри.

У проекті реалізовані 2 фільтра:

- «base_filter(message):» - базовий фільтр для звичайних користувачів. Перевіряє чи зареєстрований користувач у боті. У випадку якщо у користувача немає персонажа, то замість обробки команди бот відповідає користувачу, пропонуючи йому ввести «/start» в особистих повідомленнях боту (якщо діалог з ботом пустий, то замість текстового поля користувачам завжди доступна лише кнопка для початку діалогу, яка автоматично вводить «/start» від імені користувача. Тому гравцям достатньо почати діалог з ботом для створення і реєстрації персонажа.
- «dev_filter(message):» - фільтр для команд, які може використовувати лише розробник. Перевіряє чи збігається унікальний ID відправника повідомлення зі вписаним у код бота ID у змінній dev_id, яка знаходится у файлі Common.py.

Серед обробників є не тільки обробники повідомлень. Прикладом інших обробників є обробник змінення статусу бота у чаті. Його код наведено нижче:

```
@bot.my_chat_member_handler(chat_types=['supergroup'])
def my_chat_member(me):
    if(me.new_chat_member.status=="member"):
        if(ChatCitiesTable.contains(Query().chatID == me.chat.id)):
            bot.send_message(me.chat.id, lang.main.city_welcome(me))
        else:
            newChat = ChatCity(me.chat.id)########
            newChat.save()
            bot.send_message(me.chat.id, lang.main.city_welcome_back(me))#########

    else:
        #ChatCitiesTable.remove(me.chat.id)
        pass
```

Рисунок 2.2.4.8 – функція обробки додавання бота у супергрупи.

У випадку якщо бот був доданий до чату вперше – бот реєструє чат у базі даних та вітається вперше. У іншому випадку бот вітається особливим повідомленням

```
@bot.my_chat_member_handler(chat_types=['group'])
def group_answer(message):
    bot.send_message(message.chat.id, lang.main.only_for_supergroups(message))
```

Рисунок 2.2.4.9 – функція обробки додавання бота у звичайну групу.

Також був створений обробник спеціально для звичайних груп. Бот був налаштований на те, щоб не працювати у звичайних групах, а лише в супергрупах. Це було зроблено через те, що при покращенні групи до супергрупи змінюється її ID, що робить проблемним реєстрацію чату у базі даних.

Також важливою частиною файлу main.py та його обробників є обробник колбеків:

```python
@bot.callback_query_handler(func=lambda call: True)
def callback(call):
    request=call.data.split() #####
    if((call.from_user.id != call.message.reply_to_message.from_user.id) and not "FOR_EVERYONE" in request):
        bot.answer_callback_query(callback_query_id=call.id, text=lang.main.you_already_have_character(call.message))
        return
    match request[0]:
        case "UPGRADE":
            Logic.statsUpgrade(call.message, call)
        case "WORK":
            pass
        case "MAP":
            pass
        case "FIGHT":
            pass
        case "MARKET":
            pass
        case "REST":
            pass
        case "CORP":
            if(request[1] == "UPGRADE"):
                corp = Corp(PlayersTable.search(Query().ID == call.from_user.id)[0]["corp"])
                if(len(request) == 3):
                    try:
                        match request[2]:
                            case "R&D":
                                match corp.departments["R&D"]+1:
                                    case 1:
                                        corp.departmentUP("Factories")
                                    case 2:
                                        corp.departmentUP("Datacenter")
```

Рисунок 2.2.4.10 – невеликий шматок коду обробника колбеків. Це найбільфа функція у файлі main.py.

Callback працює на основі callback-кнопок. Коли користувач натискає callback-кнопку (кнопку зворотнього дзвінка), повідомлення в чат не надсилаються. Натомість ваш бот просто отримує відповідний запит. Отримавши запит, ваш бот може відобразити певний результат у вигляді декількох видів сповіщень, або здійснити інші інші дії, наприклад змінити текст повідомлення.

Функція «@bot.callback_query_handler()» є функцією, що оброблює всі колбеки які поступають у бота зі всіх кнопок на повідомленнях.

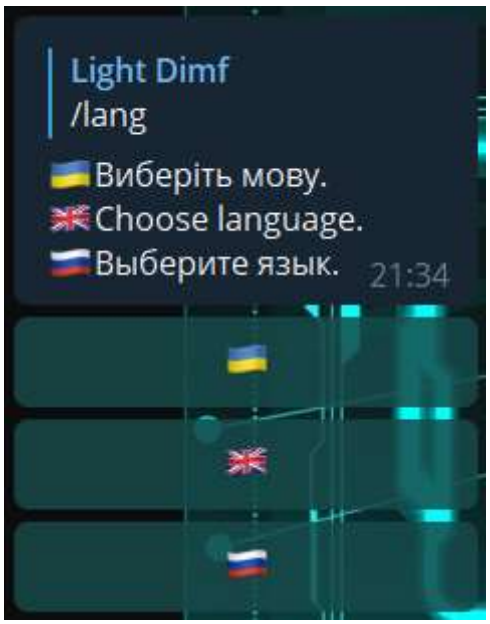


Рисунок 2.2.4.11 – приклад повідомлення з callback-кнопками.

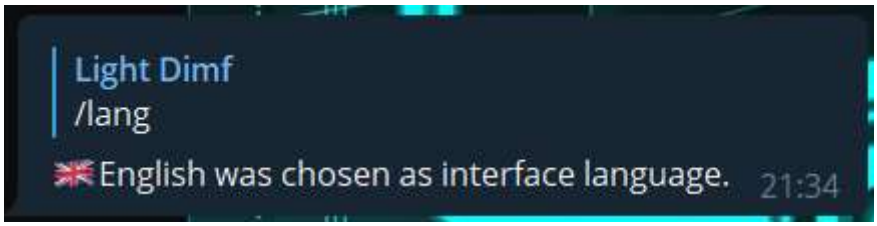


Рисунок 2.2.4.12 – після натиснення другої кнопки у повідомленні повідомлення було переписане ботом. На відміну від повідомлень користувачів відредактовані повідомлення у ботів не відображуються з характерною позначкою.

```
keyboard = telebot.types.InlineKeyboardMarkup()
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.HACK(message), callback_data="UPGRADE SKILL HACK"))
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.BIOTECH(message), callback_data="UPGRADE SKILL BIOTECH"))
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.ELECTROTECH(message), callback_data="UPGRADE SKILL ELECTROTECH"))
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.NEGOTIATION(message), callback_data="UPGRADE SKILL NEGOTIATION"))
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.RANGED(message), callback_data="UPGRADE SKILL RANGED"))
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.MELEE(message), callback_data="UPGRADE SKILL MELEE"))
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.STEALTH(message), callback_data="UPGRADE SKILL STEALTH"))
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.SURVIVAL(message), callback_data="UPGRADE SKILL SURVIVAL"))
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.button_move_to_attributes(message), callback_data="UPGRADE ATTRIBUTE MOVE"))
bot.edit_message_text(lang.logic.upgrade_skills(message, player), message.chat.id, callback.message.message_id, reply_markup=keyboard)
```

Рисунок 2.2.4.13 – приклад коду для створення кнопок та змінення повідомлення.

Архітектура зворотніх викликів (колбеків) у проекті була розроблена по принципу передачі інформації словами у верхньому регістрі де кожне слово по порядку означає:

1. Основне повідомлення, команду якою було створено повідомлення. У даному випадку це «UPGRADE».
2. Слово яке позначає у який із доступних станів повинно перейти повідомлення. У даному випадку майже всі кнопки виконують функцію покращення персонажа і не змінюють стран повідомлення, тому вони мають другим словом «SKILL», завдяки якому при редактуванні повідомлення повернеться у цей самий стан. Остання кнопка має навпаки слово «ATTRIBUTE», що переведе повідомлення у інший стан.
3. Додаткова інформація. У даному випадку кнопки покращення мають інформацію про те яку саме навичку потрібно покращити при натисканні кнопки. У деякий не частих випадках потрібно більш ніж одне слово для передачі даних, наприклад для збереження назв, які можуть складатися з декількох слів, для таких випадків у методі обробки колбеків є код для зліплення всього що йде після перших двух слів у один рядок.

Крім основного файлу програми main.py у проекті є ще більш загальна бібліотека Common.py у якій зібрані різні речі які використовуються у всіх інших файлах бота:

```
import telebot
from tinydb import TinyDB

bot = telebot.TeleBot('[redacted]')

db = TinyDB('db.json')
PlayersTable = db.table("Players")
ChatCitiesTable = db.table("ChatCities")
CorpsTable = db.table("Corps")
OtherTable = db.table("Other")

dev_id = 345694869
```

Рисунок 2.2.4.14 – код файлу Common.py де оголошщена змінна для доступу до бота через функціонал бібліотеки pyTelegramBotAPI (telebot скорочено), база даних та таблиці до неї, а також dev_id – id розробника.

Файл Localisations.py з класом lang створений для того, щоб реалізувати зміну мови у повідомленнях бота:

```
from Common import PlayersTable, ChatCitiesTable
from tinydb import Query

class lang:
    """Методи - str в різних місцях програми. Приймають повідомлення аргументом."""

    @staticmethod
    def _checkLang(message):
        if(message.chat.type=="private"):
            return PlayersTable.search(Query().ID == message.chat.id)[0]["lang"]
        else:
            return ChatCitiesTable.search(Query().chatID == message.chat.id)[0]["lang"]

    class main:
        @staticmethod
        def say_start(message):
            text = {
                "ua": "Напишіть боту /start для створення персонажа.",
                "en": "Type /start to the bot to create a character.",
                "ru": "Напише боту /start в лс для создания персонажа."
            }
            return text[lang._checkLang(message)]

        @staticmethod
        def welcome_to_cybermancy(message):
            text = {
                "ua": "Вітаємо у Cybermancy.\n\nВикористайте /help щоб отримати інформацію.\nВикористайте /lang щоб змінити мову.",
                "en": "Welcome to the Cybermancy.\n\nUse /help to get info.\nUse /lang to change language.",
                "ru": "Добро пожаловать в Cybermancy.\n\nИспользуйте /help чтобы получить информацию.\nИспользуйте /lang чтобы изменить язык."
            }
            return text[lang._checkLang(message)]
```

Рисунок 2.2.4.15 – початок коду файлу Localisations.py.

Клас lang має декілька вкладених класів для того, щоб розмежувати локалізації для різних файлів бота для більш простої орієнтації. Кожен метод класу представляє з себе статичний метод, який приймає на вході запит надісланий боту (або більше аргументів якщо треба) і повертає один з трьох рядків на українській, англійській чи російській мовах в залежності від налаштувань користувача, або чату.

Передача оброблюваного запиту треба для псевдо-приватного (Python не має повністю приватніх методів) методу «_checkLang()», який не тільки визначає налаштування мови, але й визначає звідки саме було відправлене повідомлення.

```
@staticmethod
def upgrade_attribute(message, player):
    text = {
        "ua": ("Розвиток персонажа.\n\nХарактеристики:\n" +
               "Сила: {}  [{}]\nВитривалість: {}  [{}]\n" +
               "Спритність: {}  [{}]\nІнтеллект: {}  [{}]\n" +
               "Сприйняття: {}  [{}]\n" +
               ("Псі: {psi}  [{psi_cost}]\n" if "PSI" in player.augmentations else "") + ###

               "\nДосвід: {}"),
        "en": ("Character upgrade.\n\nAttributes:\n" +
               "Strength: {}  [{}]\nEndurance: {}  [{}]\n" +
               "Agility: {}  [{}]\nIntelligence: {}  [{}]\n" +
               "Perception: {}  [{}]\n" +
               ("PSI: {psi}  [{psi_cost}]\n" if "PSI" in player.augmentations else "") + ###

               "\nXP: {}"),
        "ru": ("Развитие персонажа.\n\nХарактеристики:\n" +
               "Сила: {}  [{}]\nВыносливость: {}  [{}]\n" +
               "Ловкость: {}  [{}]\nИнтеллект: {}  [{}]\n" +
               "Восприятие: {}  [{}]\n" +
               ("Пси: {psi}  [{psi_cost}]\n" if "PSI" in player.augmentations else "") + ###

               "\nОпыт: {}")
    }
    return text[lang._checkLang(message)].format(player.STR_base, player.XP_cost("STR"), player.END_base, player.XP_cost("END"),
                                                 player.AGI_base, player.XP_cost("AGI"), player.INT_base, player.XP_cost("INT"),
                                                 player.PER_base, player.XP_cost("PER"), player.xp, psi=player.PSI_base, psi_cost=player.XP_cost("PSI"))
```

Рисунок 2.2.4.16 – приклад більш складного методу у класі lang з двума аргументами та форматуванням тексту.

Одним з найважливіших файлів програми є також Logic.py з класом «Logic» у який була перенесена більша частина логіки програми, щоб зменшити обсяг коду в main.py та покращити його читаємість, бо main.py призначений перш за все для обробки повідомлень, для фронтенду, в той час як Logic.py створений спеціально для загальної логіки програми з різними функціями.

```
from classes.Player import Player
from classes.Corp import Corp
import telebot
from Common import bot, PlayersTable, CorpsTable
from tinydb import Query
from Localisations import lang

class Logic:
    @staticmethod
    def statsUpgrade(message, callback=None):
        player = Player(message.chat.id)
        request = None if callback is None else callback.data.split()
        if(request!=None):
            try:
                match request[2]:
                    case "STR": player.statUP("STR")
                    case "END": player.statUP("END")
                    case "AGI": player.statUP("AGI")
                    case "INT": player.statUP("INT")
                    case "PER": player.statUP("PER")
                    case "PSI": player.statUP("PSI")
                    case "HACK": player.statUP("HACK")
                    case "BIOTECH": player.statUP("BIOTECH")
                    case "ELECTROTECH": player.statUP("ELECTROTECH")
                    case "NEGOTIATION": player.statUP("NEGOTIATION")
                    case "RANGED": player.statUP("RANGED")
                    case "MELEE": player.statUP("MELEE")
                    case "STEALTH": player.statUP("STEALTH")
                    case "SURVIVAL": player.statUP("SURVIVAL")
                    case "MOVE":
                        pass
```

Рисунок 2.2.4.17 – приклад того як виглядає файл Logic.py. Також зображений початок методу «statsUpgrade()», який виконує функцію покращення гравця.

У проекті також використовуються модель кінцевих автоматів для реалізації переходів між різними станами повідомлень, де за допомогою кнопок можна повністю змінювати повідомлення без потреби відправляти нове, що значно зменшує кількість спаму повідомленнями бота, що дуже корисно для чатів, щоб тримати їх у чистоті, що дуже часто буває проблемою для подібних ботів.

Кінцевий автомат - це деяка абстрактна модель, що містить кінцеве число станів чогось. Використовується для представлення та керування потоком виконання будь-яких команд.

Кінцевий автомат (або просто FSM - Finite-state machine) це модель обчислень, заснована на гіпотетичній машині станів. Одночасно тільки один стан може бути активним. Отже, для виконання будь-яких дій машина має змінювати свій стан.

```
@staticmethod
def corpMain(message, callback=None):
    corp = Corp(PlayersTable.search(Query().ID == message.from_user.id)[0]["corp"])
    owner = bot.get_chat_member(message.chat.id, message.from_user.id).user
    keyboard = telebot.types.InlineKeyboardMarkup()
    request = None if callback is None else callback.data.split()
    if(request == None or request[1]=="MAIN"):
        if(owner.id == message.from_user.id):
            keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.corp_main_upgrade(message), callback_data="CORP UPGRADE"))
            keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.corp_main_convert(message), callback_data="CORP CONVERT"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.corp_main_production(message), callback_data="CORP PRODUCTION"))
        bot.reply_to(message, lang.logic.corp_main_message(message, corp, owner), reply_markup=keyboard)
    elif(request[1]=="UPGRADE"):...
    elif(request[1]=="CONVERT"):
        convert_keyboard = telebot.types.InlineKeyboardMarkup()
        convert_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.corp_convert_money(message), callback_data="CORP CONVERT MONEY"))
        convert_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.corp_convert_influence(message), callback_data="CORP CONVERT INFLUENCE"))
        convert_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.cancel(message), callback_data="CORP MAIN"))
        bot.edit_message_text(lang.logic.corp_convert_recources_message(message, corp), message.chat.id, callback.message.message_id, reply_markup=convert_
    elif(request[1]=="PRODUCTION"):
        pass
```

Рисунок 2.2.4.18 – реалізація кінцевих автоматів на прикладі функції corpMain(), основної функції для обробки команди /corp. Працює стандартно і через колбеки. Якщо функція була викликана через команду /corp, то виконається перший виділений червоним варіант, якщо функція була викликана після натиснення кнопок, то виконається варіант до якого ця кнопка вела. Наприклад до покращень (код згорнутий за допомогую функціоналу PyCharm).

```python
@staticmethod
def statsUpgrade(message, callback=None):
    player = Player(message.chat.id)
    request = None if callback is None else callback.data.split()
    if(request!=None):...

    if(request==None or request[1]=="ATTRIBUTE"):
        keyboard = telebot.types.InlineKeyboardMarkup()
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.STR(message), callback_data="UPGRADE ATTRIBUTE STR"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.END(message), callback_data="UPGRADE ATTRIBUTE END"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.AGI(message), callback_data="UPGRADE ATTRIBUTE AGI"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.INT(message), callback_data="UPGRADE ATTRIBUTE INT"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.PER(message), callback_data="UPGRADE ATTRIBUTE PER"))
        if("PSI" in player.augmentations): ###
            keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.PSI(message), callback_data="UPGRADE PSI"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.button_move_to_skills(message), callback_data="UPGRADE SKILL MOVE")) #
        if(request==None):
            bot.reply_to(message, lang.logic.upgrade_attribute(message, player), reply_markup=keyboard)
        else:
            bot.edit_message_text(lang.logic.upgrade_attribute(message, player), message.chat.id, callback.message.message_id, reply_markup=keyboard)
    elif(request[1]=="SKILL"):
        keyboard = telebot.types.InlineKeyboardMarkup()
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.HACK(message), callback_data="UPGRADE SKILL HACK"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.BIOTECH(message), callback_data="UPGRADE SKILL BIOTECH"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.ELECTROTECH(message), callback_data="UPGRADE SKILL ELECTROTECH"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.NEGOTIATION(message), callback_data="UPGRADE SKILL NEGOTIATION"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.RANGED(message), callback_data="UPGRADE SKILL RANGED"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.MELEE(message), callback_data="UPGRADE SKILL MELEE"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.STEALTH(message), callback_data="UPGRADE SKILL STEALTH"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.SURVIVAL(message), callback_data="UPGRADE SKILL SURVIVAL"))
        keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.button_move_to_attributes(message), callback_data="UPGRADE ATTRIBUTE MOVE")) #
        bot.edit_message_text(lang.logic.upgrade_skills(message, player), message.chat.id, callback.message.message_id, reply_markup=keyboard)
```

Рисунок 2.2.4.19 – ще один приклад кінцевих автоматів у функції «statsUpgrade()». У даному випадку це переключення між двума станами: відображенням/покращенням характеристик та між відображенням/покращенням навичок персонажу.

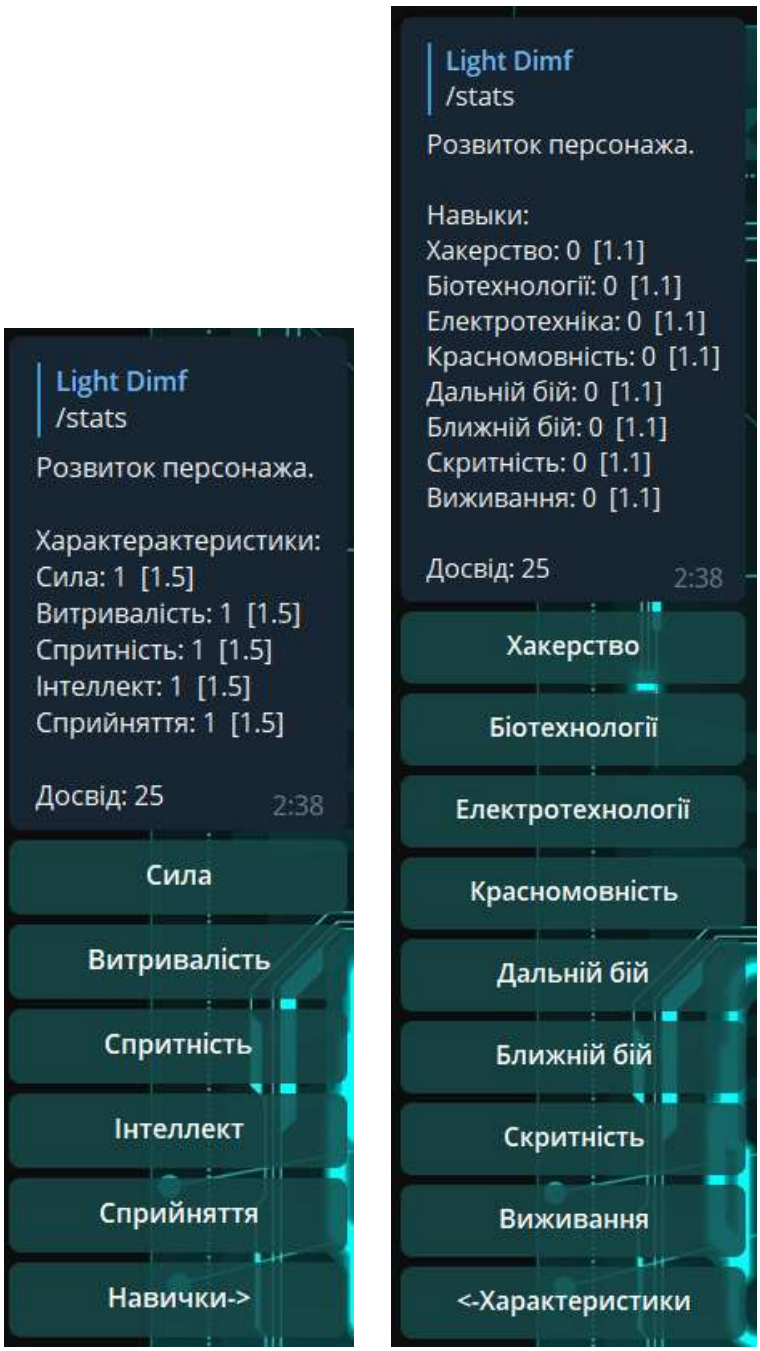


Рисунок 2.2.4.20-21 – скріншоти двох станів одного повідомлення яке було створено на відповідь на команду «/stats». Оба стани перемикаются один на одного за допомогою останньої кнопки.

Наступним по важливості є клас гравця «Player» із файлу Player.py:

```
class Player:
    ID: int
    city = "None"
    corp = "None"
    equipment = {
        "head": "None",
        "body": "None",
        "weapon1": "None",
        "weapon2": "None",
        "other1": "None",
        "other2": "None"}
    inventory = []
    augmentations = []
    psiPowers = []
    genes = []
    effects = []
    stats = {
        "health": 100,
        "mental": 100,
        "stamina": 100,
        "xp": 25,
        "xp_total_spend": 0, #####
        "heat": 0,
        "money": 10000, #####
        "attributes": {
            "STR": 1,
            "END": 1,
            "AGI": 1,
            "INT": 1,
            "PER": 1,
            "PSI": 0
        },
```

Рисунок 2.2.4.22 – приклад коду класу гравця. Базові параметри гравців.

Кожен раз коли у коді програми треба провзаємодіяти з інформацією про гравця – створюється об’якт класу Player (який або створюється з нуля, або дістає інформацію з бази даних) з яким проводятся маніпуляції і коли код завершується без помилок гравця зберігають у базу даних. Також клас містить багато методів для взаємодії з гравцем. Наприклад:

- «XP_cost(self, stat: str)» - повертає ціну покращення навички, або характеристики у очках досвіду.

- «statUP(self, stat: str)» - покращує навичку, або характеристику. Викликає помммиииилку у ряді випадків, наприклад якщо гравець покращився до свого ліміту.
- «healthChange(self, amount)» та «staminaChange(self, amount)» - змінення здоров'я та витривалості гравця з додатковою обробкою можливих сценаріїв.

```
def XP_cost(self, stat: str):
    match stat:
        case "STR":
            return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["STR"]-9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
        case "END":
            return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["END"]-9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
        case "AGI":
            return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["AGI"]-9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
        case "INT":
            return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["INT"]-9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
        case "PER":
            return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["PER"]-9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
        case "PSI":
            return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["PSI"]-9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
        case "HACK":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["HACK"]-9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "BIOTECH":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["BIOTECH"]-9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "ELECTROTECH":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["ELECTROTECH"]-9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "NEGOTIATION":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["NEGOTIATION"]-9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "RANGED":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["RANGED"]-9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "MELEE":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["MELEE"]-9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "STEALTH":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["STEALTH"]-9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "SURVIVAL":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["SURVIVAL"]-9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
```

Рисунок 2.2.4.23.1 – метод визначення ціни покращення характеристики, або навички гравця, яка вираховується за допомогою формули логістичної функції та з урахуванням всього витраченного досвіду. Також результат округлюється до першого знака після коми.

Формула логістичної функції (2.2.4)

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \quad (2.2.4)$$

де L - максимальне значення кривої;

k - логістичний темп зростання або крутизна кривої;

x0 - значення x середньої точки сигмовидної кістки;

x – змінна.

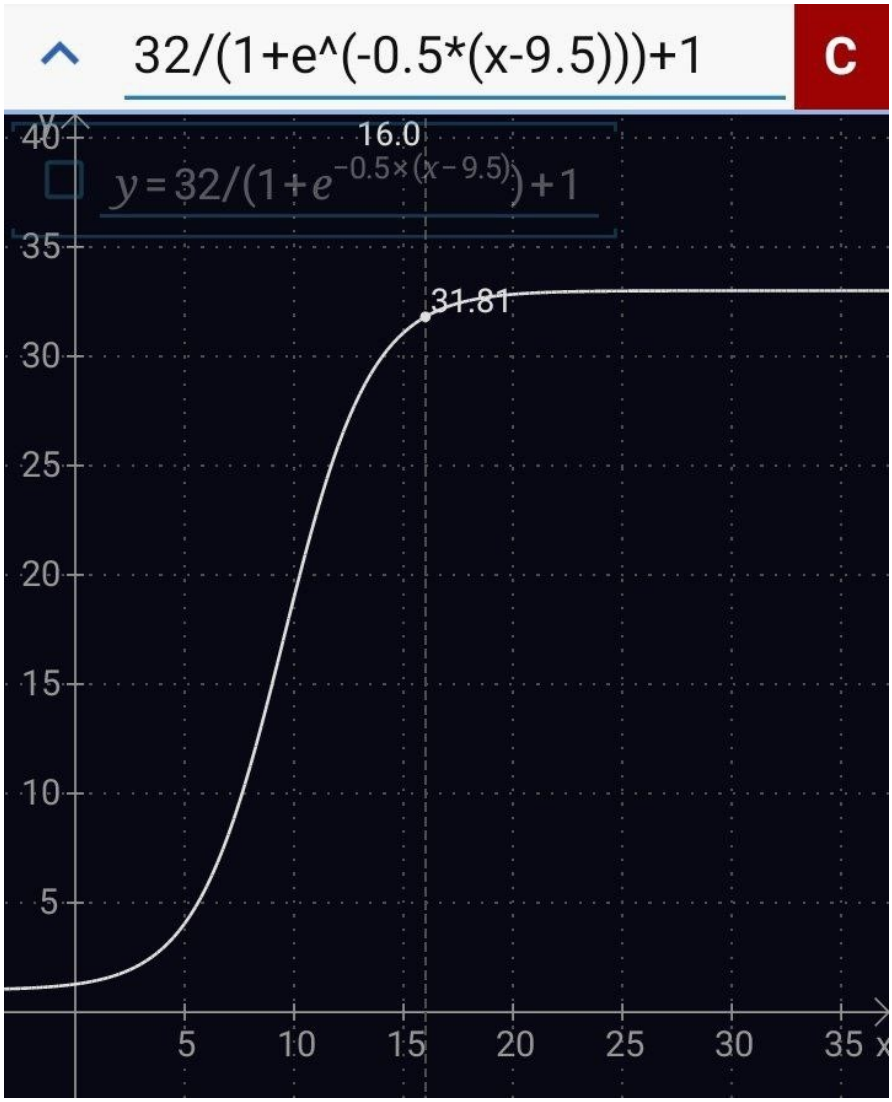


Рисунок 2.2.4.23.3 – графік логістичної функції використаної для покращення характеристик персонажа. x=16 це максимальний можливий рівень покращення.

```
def statUP(self, stat: str):
    if(stat in self.stats["attributes"].keys()):
        if(self.stats["attributes"][stat]+1 > 16):
            raise Exception("Предел человеческих возможностей")
        if(self.stats["xp"] < self.XP_cost(stat)):
            raise Exception("Недостаточно опыта")
        self.stats["xp_total_spend"] = float(format(self.stats["xp_total_spend"] + self.XP_cost(stat), '.1f'))
        self.stats["xp"] = float(format(self.stats["xp"] - self.XP_cost(stat), '.1f'))
        self.stats["attributes"][stat] += 1
    else:
        if(self.stats["skills"][stat]+1 > 16):
            raise Exception("Предел человеческих возможностей")
        if(self.stats["xp"] < self.XP_cost(stat)):
            raise Exception("Недостаточно опыта")
        self.stats["xp_total_spend"] = float(format(self.stats["xp_total_spend"] + self.XP_cost(stat), '.1f'))
        self.stats["xp"] = float(format(self.stats["xp"] - self.XP_cost(stat), '.1f'))
        self.stats["skills"][stat] += 1
```

Рисунок 2.2.4.24 – код методу покращення навичок та характеристик.

```
def healthChange(self, amount): ###
    if(amount>0):
        if(self.stats["health"] + amount < 100):
            self.stats["health"] += amount
        else:
            self.stats["health"] = 100
        return True
    else:
        if(self.stats["health"] + amount > 0):
            self.stats["health"] -= amount/100 * (100 - self.stats["attributes"]["END"]*2)
            return True
        else:
            return False

def staminaChange(self, amount):
    if(amount>0):
        if(self.stats["stamina"] + amount < 100):
            self.stats["stamina"] += amount
        else:
            self.stats["stamina"] = 100
        return True
    else:
        if(self.stats["stamina"] + amount > 0):
            self.stats["stamina"] -= amount/100 * (100 - self.stats["attributes"]["END"]*2)
            return True
        else:
            return False
```

Рисунок 2.2.4.25 – код методів змінення здоров’я та витривалості.

```
@property
def STR_base(self):
    return self.stats["attributes"]["STR"]
@property
def STR(self):
    return self.stats["attributes"]["STR"] #####

@property
def END_base(self):
    return self.stats["attributes"]["END"]
@property
def END(self):
    return self.stats["attributes"]["END"] #####

@property
def AGI_base(self):
    return self.stats["attributes"]["AGI"]
@property
def AGI(self):
    return self.stats["attributes"]["AGI"] #####

@property
def INT_base(self):
    return self.stats["attributes"]["INT"]
@property
def INT(self):
    return self.stats["attributes"]["INT"] #####
```

Рисунок 2.2.4.26 – частина методів які повертають характеристики та навички персонажу. «getters», як це називається. Половина з них просто повертає базове значення параметру, друга половина повинна повертати їх з урахуванням спорядження та інших опціональних поркащень, але нажаль це не було реалізовано.

Наступним класом є клас ChatCity у файлі Chat_City.py, який зв’язаний з чатами:

```
class ChatCity:
    chatID: int
    locations = []
    corps = []
    lang = "ua"
```

Рисунок 2.2.4.27 – клас ChatCity зберігає доволі мало інформації.

```
def __init__(self, chatID):
    self.chatID = chatID
    chat = ChatCitiesTable.search(Query().ID == self.chatID)
    if(chat != []):
        chat=chat[0][str(chatID)]
        self.locations = chat["locations"]
        self.corps = chat["corps"]
        self.lang = chat["lang"]
    else:
        self.generateLocations()
```

Рисунок 2.2.4.28.1 – створення об’єкту чату.

При першому додаванні бота у чат там також генерується унікальний список локацій. Це виконується цім методом:

```
def generateLocations(self):
    size=bot.get_chat_member_count(self.chatID)
    locs=Locations.copy()
    if(size>50): #1-49
        while len(self.locations)<4:
            loc=locs.pop(list(locs.keys())[((random.randint(0, len(locs)-1)))])
            if(loc.get(1) != None):
                self.locations.append(loc.get(1))
    elif(size<150): #50-149
        while len(self.locations)<6:
            loc=locs.pop(list(locs.keys())[((random.randint(0, len(locs)-1)))])
            if(loc.get(2) != None):
                self.locations.append(loc.get(2))
            else:
                if(loc.get(1) != None):
                    self.locations.append(loc.get(1))
    else: #150+
        while len(self.locations)<8:
            loc=locs.pop(list(locs.keys())[((random.randint(0, len(locs)-1)))])
            if(loc.get(3) != None):
                self.locations.append(loc.get(3))
            else:
                if(loc.get(2) != None):
                    self.locations.append(loc.get(2))
                else:
                    if(loc.get(1) != None):
                        self.locations.append(loc.get(1))
```

Рисунок 2.2.4.28.2 – метод «generateLocations()», який генерує локації в кількості та рівнями залежними від розміру чату, які ділятся на 3 категорії: 1-49, 50-149 та 150+.

Список можливих локацій визначений у файлі Locations.py:

```
Locations = {
    "Трущобы": {
        1: "Трущобы",
        3: "Нижний город"
    },
    "Промсектор":{
        1: "Заброшенный промсектор"
    },
    "Пустоши":{
        1: "Пустоши"
    },
    "Рынок":{
        1: "Рынок",
        2: "Полузаброшенный ТЦ",
        3: "Мегаплекс"
    },
    "Офисы":{
        1: "Деловой квартал",
        2: "Корпоративные офисы",
        3: "Корпоративный эпицентр"
    },
    "Жилое":{
        1: "Жилой район",
        3: "Улей"
    },
    "Канализации":{
        1: "Канализации"
    },
    "Арена":{
        1: "Подпольная арена"
    }
```

Рисунок 2.2.4.28.3 – файл Locations.py.

Locations.py має у собі лише ону змінну – «Locations», яка є словником. кожний елемент словника є типом локації, які, у свою чергу, теж всі є словниками, де є різні рівні локацій для різних категорій чатів. В залежності від категорії чату вибирається деяка кількість локацій. При цьому вибір йде серед типів локацій, після чого вибераєтся найбільший наявний підходящий тип локації. Так для

малих чатів підходять лише локації першого рівня. Для чатів з розміром 150+ буде вибиратися підтип локації третього рівня. Якщо такого рівня немає для цього типу локації, то буде вибраний рівень 2. Якщо немає і його, то рівень 1.

Останній файл проекту це Corp.py, який містить клас Corp:

```
from Common import PlayersTable, CorpsTable
from tinydb import Query
from Localisations import lang

class Corp:
    name: str
    owner: int
    departments = {
        "R&D": 0,
        "HR": 0,
        "Security Service": 0,
        "Chemical Labs": 0,
        "Sales department": 0,
        "Factories": 0,
        "Datacenter": 0,
        "Contacts": 0
    }
    resources = {
        "Budget": 0,
        "Influence": 0
    }

    def __init__(self, name: str, owner = None):
        self.name = name
        if(owner!=None):
            self.owner = owner
        else:
            corp = CorpsTable.search(Query().name == self.name)[0]
            self.departments = corp["departments"]
            self.resources = corp["resources"]

    def save(self):
```

Рисунок 2.2.4.29 – початок коду Corp.py

Клас Corp призначений для корпорацій, які є аналогами кланів, які є в багатьох багатокористувацьких іграх. Клас зберігає свою назву (Замість ID), ID власника, департаменти (покращення корпорації) та ресурси.

Наймассивнішою частиною класу є змінна «departmentsPriceList», яка є списком, який зберігає інформацію про ресурсну ціну та опис кожного рівня кожного покращення корпорації:

```
departmentsPriceList = {
    "R&D": {
        1: {"price": {"Budget": 2500, "Influence": 0},
            "description": "Открывает заводы."},
        2: {"price": {"Budget": 3000, "Influence": 0},
            "description": "Открывает датацентр."},
        3: {"price": {"Budget": 5000, "Influence": 0},
            "description": "Открывает химические лаборатории."},
        4: {"price": {"Budget": 7500, "Influence": 10},
            "description": "Повышает максимальный уровень улучшений до 5."},
        5: {"price": {"Budget": 10000, "Influence": 20},
            "description": "Открывает доступ к особым биомодам."},
        6: {"price": {"Budget": 15000, "Influence": 50},
            "description": "Открывает доступ к особым аугментациям."},
        7: {"price": {"Budget": 20000, "Influence": 75},
            "description": "Повышает максимальный уровень улучшений до 7."},
        8: {"price": {"Budget": 25000, "Influence": 100},
            "description": "Повышает максимальный уровень улучшений до 8."},
        9: {"price": {"Budget": 50000, "Influence": 150},
            "description": "Повышает максимальный уровень улучшений до 9."},
        10: {"price": {"Budget": 75000, "Influence": 250},
            "description": "Повышает максимальный уровень улучшений до 10."}
    },
    "HR": {
        1: {"price": {"Budget": 1000, "Influence": 0},
            "description": "Повышает максимальное количество сотрудников до 5 (сейчас 3)."},
        2: {"price": {"Budget": 1500, "Influence": 0},
            "description": "Повышает максимальное количество сотрудников до 7 (сейчас 5)."},
        3: {"price": {"Budget": 2000, "Influence": 0},
            "description": "Повышает максимальное количество сотрудников до 10 (сейчас 7)."},
        4: {"price": {"Budget": 2500, "Influence": 0},
            "description": "Повышает максимальное количество сотрудников до 12 (сейчас 10)."},
```

Рисунок 2.2.4.30 – змінна «departmentsPriceList».

Кожний рівень кожного покращення або щось покращує, або щось відкриває. Так R&D (Reseach and Development) відкриває максимальні рівні для інших покращень, а також деякі типи покращень.

```
elif(request[1]=="UPGRADE"):
    upgrade_keyboard = telebot.types.InlineKeyboardMarkup()
    msg_text = "Покращення корпорації:\n\n"
    if(corp.departments["R&D"]<10):
        msg_text += corp.departmentUpgradeInfo("R&D", message)+"\n\n"
        upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text="R&D", callback_data="CORP UPGRADE R&D"))
    if(corp.departments["HR"]<10):
        msg_text += corp.departmentUpgradeInfo("HR", message)+"\n\n"
        upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.HR(message), callback_data="CORP UPGRADE HR"))
    if(corp.departments["Security Service"]<10):
        msg_text += corp.departmentUpgradeInfo("Security Service", message)+"\n\n"
        upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Security_Service(message), callback_data="CORP UPGRADE Security_Service"))
    if(0<corp.departments["Chemical Labs"]<10):
        msg_text += corp.departmentUpgradeInfo("Chemical Labs", message)+"\n\n"
        upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Chemical_Labs(message), callback_data="CORP UPGRADE Chemical_Labs"))
    if(corp.departments["Sales department"]<10):
        msg_text += corp.departmentUpgradeInfo("Sales department", message)+"\n\n"
        upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Sales_department(message), callback_data="CORP UPGRADE Sales_department"))
    if(0<corp.departments["Factories"]<10):
        msg_text += corp.departmentUpgradeInfo("Factories", message)+"\n\n"
        upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Factories(message), callback_data="CORP UPGRADE Factories"))
    if(0<corp.departments["Datacenter"]<10):
        msg_text += corp.departmentUpgradeInfo("Datacenter", message)+"\n\n"
        upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Datacenter(message), callback_data="CORP UPGRADE Datacenter"))
    if(corp.departments["Contacts"]<10):
        msg_text += corp.departmentUpgradeInfo("Contacts", message)+"\n\n"
        upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Contacts(message), callback_data="CORP UPGRADE Contacts"))
```

Рисунок 2.2.4.31 – код покращення департаментів у файлі Logic.py.

```
def departmentUP(self, department: str):
    if(department != "R&D"):
        if(self.departments[department]+1>=5 and self.departments["R&D"]+1<4):
            raise Exception("R&D_level_is_too_low")
        if(self.departments[department]+1>=7 and self.departments["R&D"]+1<7):
            raise Exception("R&D_level_is_too_low")
        if(self.departments[department]+1>=8 and self.departments["R&D"]+1<8):
            raise Exception("R&D_level_is_too_low")
        if(self.departments[department]+1>=9 and self.departments["R&D"]+1<9):
            raise Exception("R&D_level_is_too_low")
        if(self.departments[department]+1>=10 and self.departments["R&D"]+1<10):
            raise Exception("R&D_level_is_too_low")
    self.budgetChange(-self.departmentsPriceList[department][self.departments[department]+1]["price"]["Budget"])#
    self.influenceChange(-self.departmentsPriceList[department][self.departments[department]+1]["price"]["Influence"])
    self.departments[department] += 1
```

Рисунок 2.2.4.32 – функція покращення департаментів у файлі Corp.py, яка використовується у коді вище.

3 ПРИКЛАДИ ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ БОТА

3.1 Початок діалогу та /start

Щоб почати роботу з ботом треба використати команду /start, яка пишеться автоматично при натисканні кнопки початку діалогу з ботом, після чого бот привітається з користувачем та виведе первинне покращення персонажу (якщо гравець вже зареєстрований, то виведе інше повідомлення і без повідомлення з покращенням):

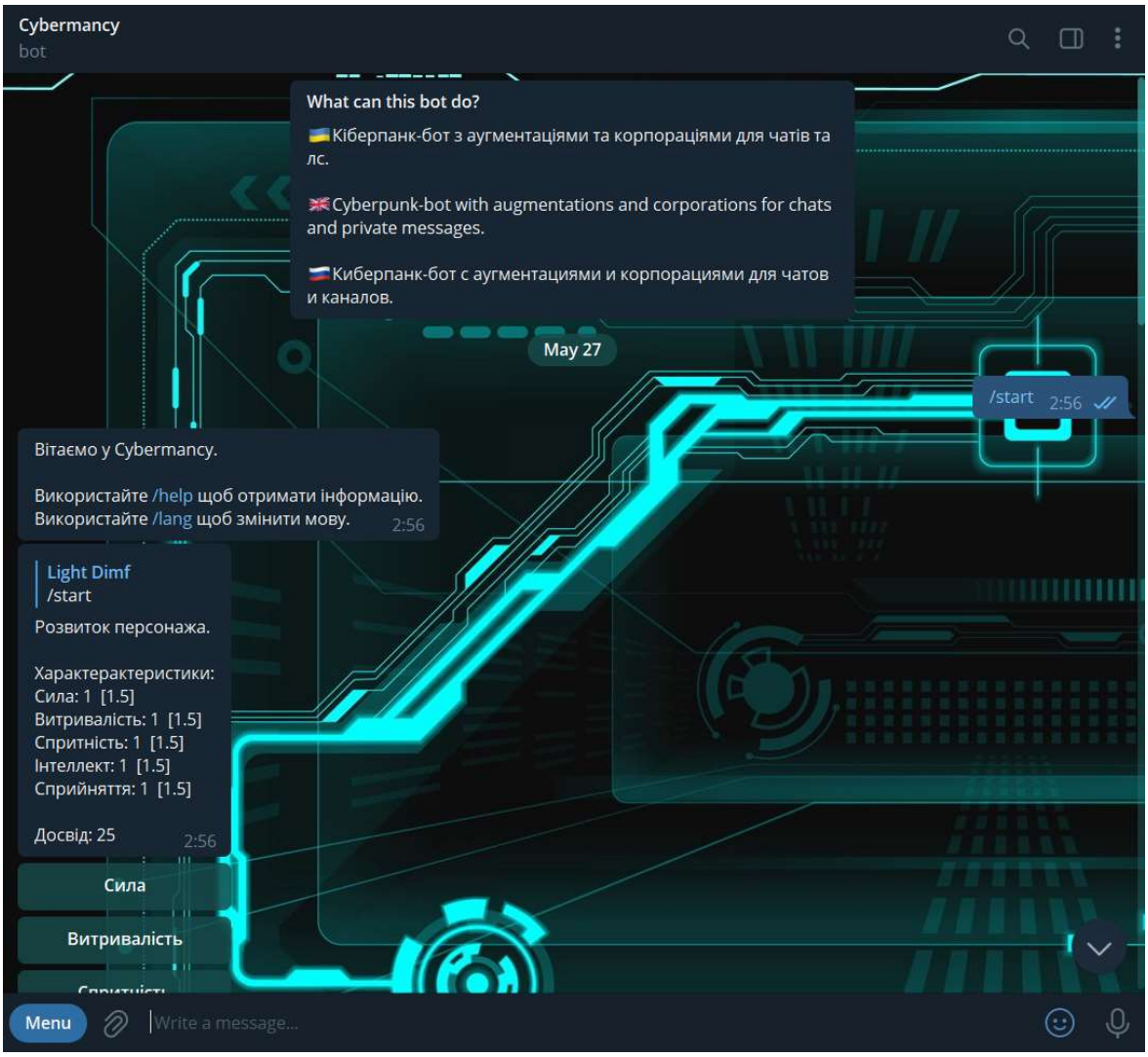


Рисунок 3.1.1 – початок діалогу з ботом.

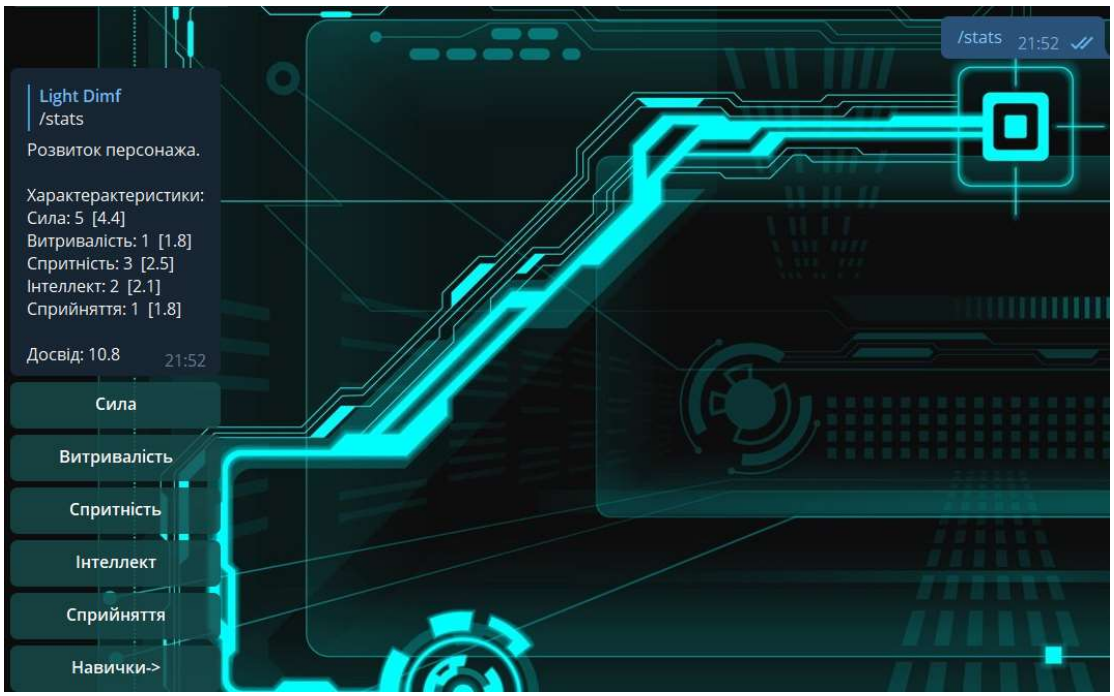


Рисунок 3.1.2 – приклад роботи покращення після натискання кнопок.

3.2 Додавання бота у чат та приєнання до чату

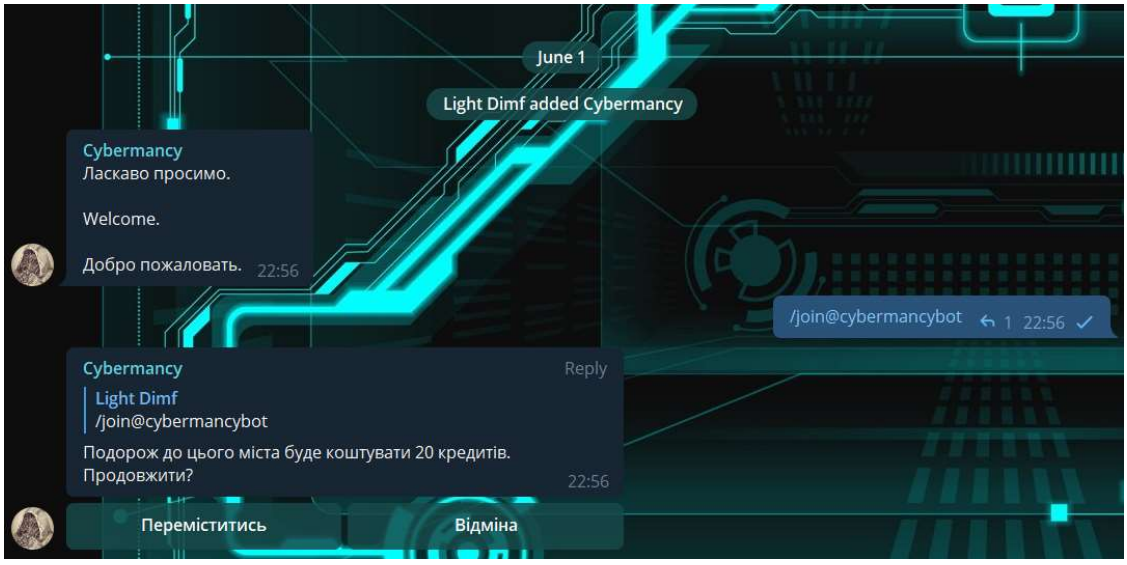


Рисунок 3.2.1 – додавання бота у чат та приєнання до чату.

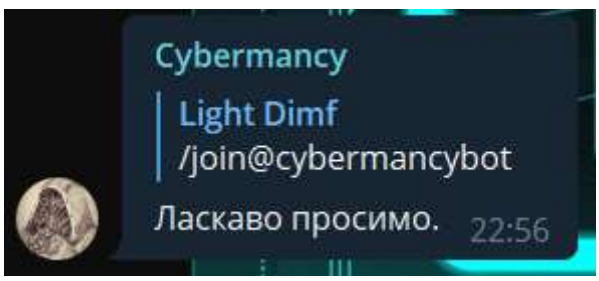


Рисунок 3.2.2 – після підтвердження переміщення персонажу до чату.

3.3 Рутинні задачі

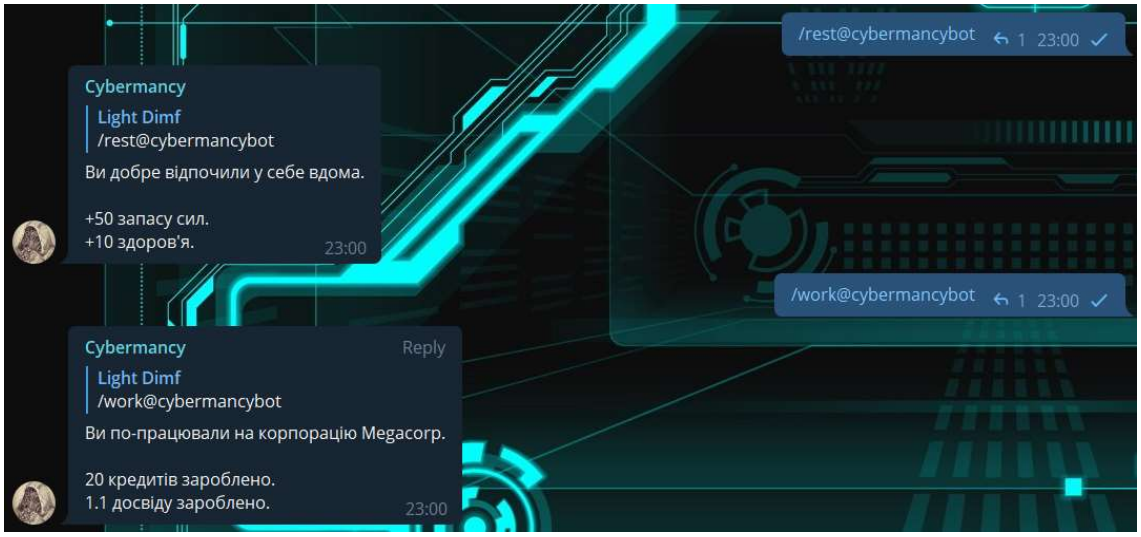


Рисунок 3.3 – відпочинок та робота.

3.4 Корпорації

Так як для розвитку потрібно багато часу, то для демострації стартові параметри були модифіковані, щоб мати доступ до всього функціоналу.

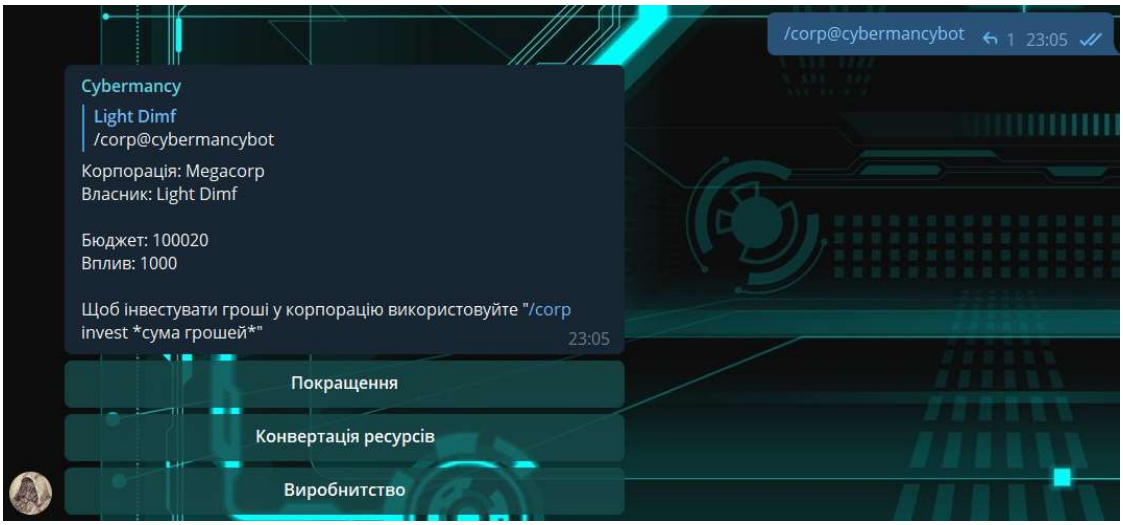


Рисунок 3.4.1 – базове повідомлення у вже створеної корпорації. Так як до розробника вже прив’язана корпорація «Megacorp», то створювати її не довелося.

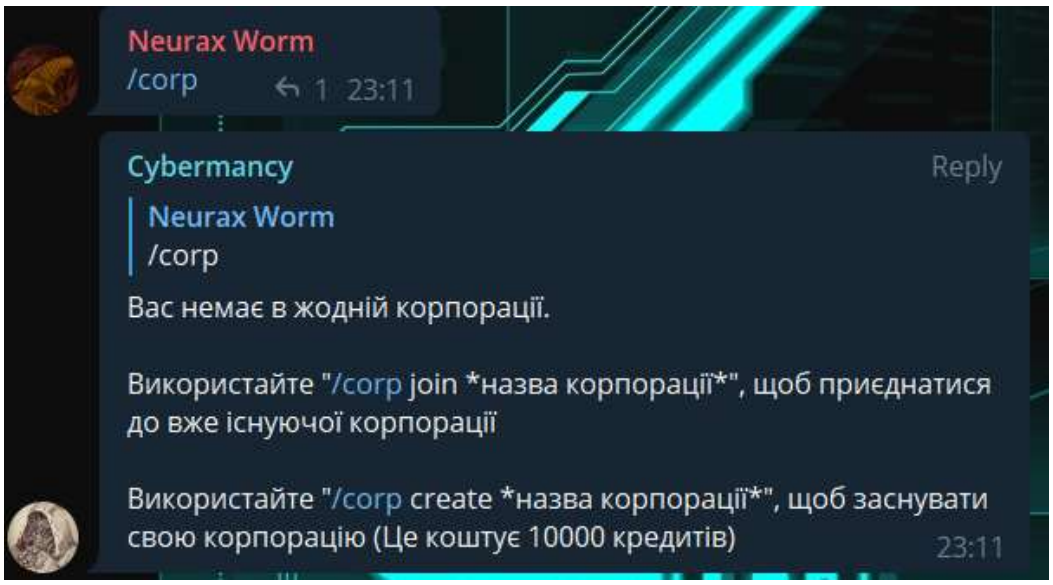
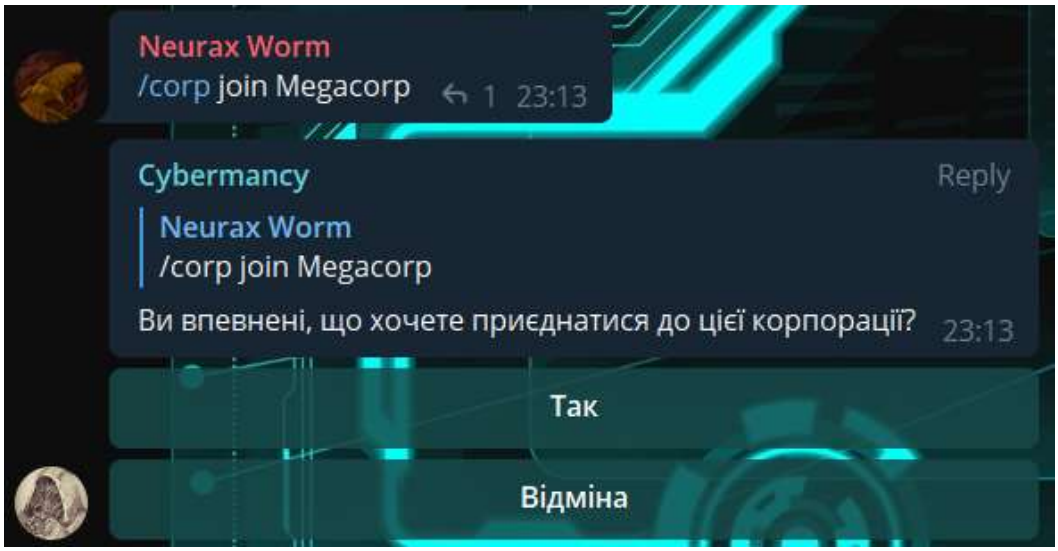


Рисунок 3.4.2 – у випадку відсутності своєї корпорації.



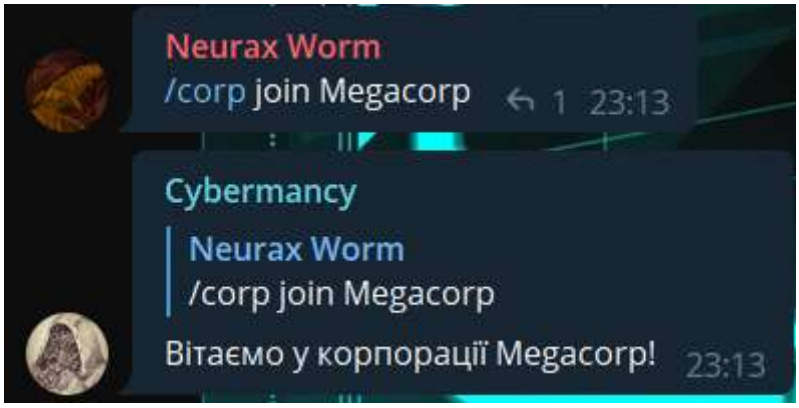
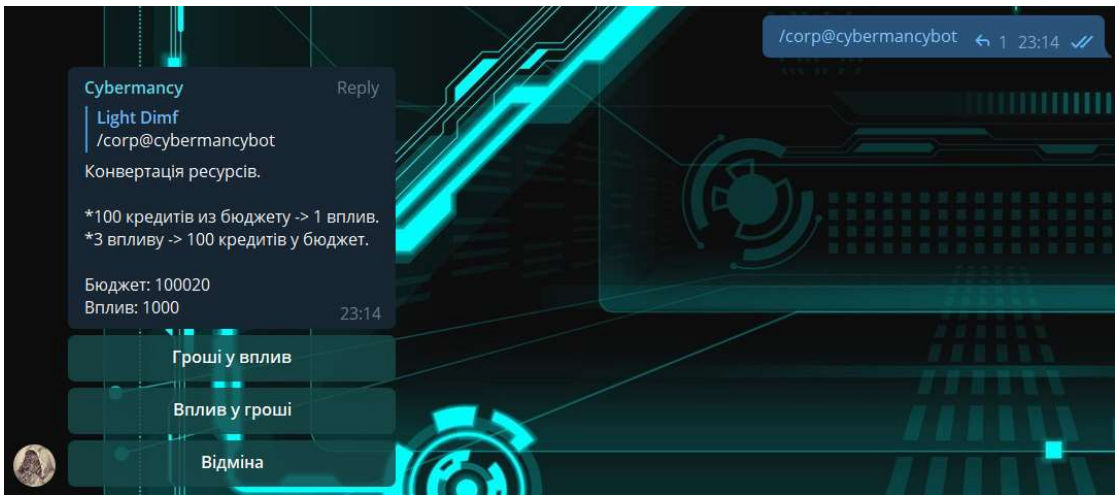


Рисунок 3.4.3-4 – приєднання до вже існуючої корпорації.



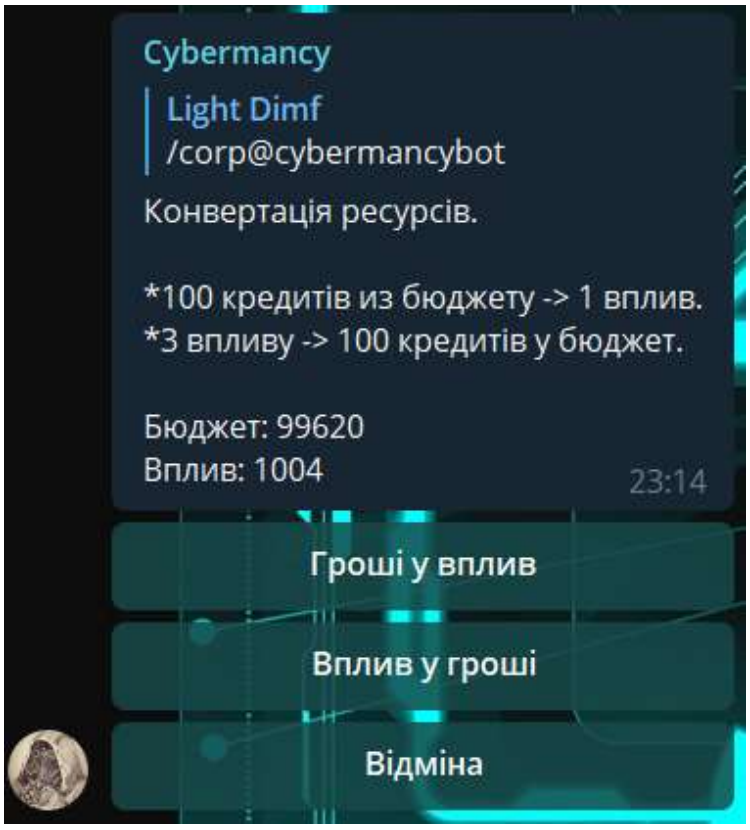


Рисунок 3.4.5-6 – конвертація ресурсів корпорації.

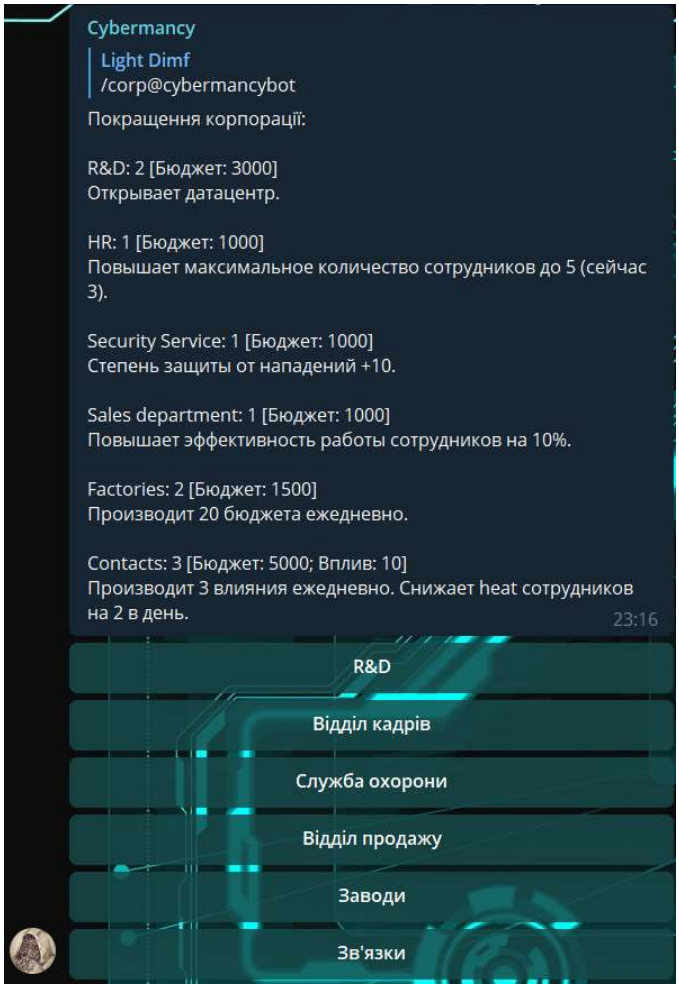


Рисунок 3.4.7-8 – покращення корпорації

ВИСНОВКИ

Робота присвячена розробці ігрового чат-боту у жанрі «Кіберпанк», що підвищить комерційну конкуренцію та приведе до покращення майбутніх ігрових механік цього жанру.

Проведено дослідження котрі обґрунтовують актуальність роботи та наукову новизну. А саме рівень зацікавленості(популярності) ігрового жанру «Кіберпанк», конкурентоспроможність ринку, проаналізовано переваги та недоліки ігрових застосунків цього жанру.

Враховуючи переваги та недоліки існуючих ігрових чат-ботів, було проаналізовано вимоги та спроектовано гру та її структурні елементи. Спроектований додаток відповідає усім виявленим вимогам.

Проведено аналіз існуючих програмних засобів для розробки чат-ботів для Telegram. Таким чином для розробки додатку було обрано мову програмування Python та середовище розробки PyCharm.

Для перевірки правильності роботи чат-боту та відповідності усім вимогам були проведені тести.

Розроблено ігровий чат-бот у жанрі «Кіберпанк» з урахуванням усіх виявлених вимог. Бота може використовувати кожен користувач месенджеру Telegram написавши боту, або додавши його до чату.

Результати досліджень бакалаврської роботи апробовані на всеукраїнських науково-технічній конференції: Науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційних технолоніях», м.Київ: ДУТ, 20 квітня 2022 року.

ПЕРЕЛІК ПОСИЛАНЬ

1. Telegram Bot API [Електронний ресурс] - Режим доступу: https://core.telegram.org/bots/api

2. TinyDB [Електронний ресурс] - Режим доступу: https://tinydb.readthedocs.io/en/latest/intro.html

3. Python [Електронний ресурс] - Режим доступу: https://www.python.org/about/

4. Bots: An introduction for developers [Електронний ресурс] - Режим доступу: https://core.telegram.org/bots

5. GitHub [Електронний ресурс]: pyTelegramBotAPI, - Режим доступу: https://github.com/eternnoir/pyTelegramBotAPI

Додаток А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Telegram-бота з багатокористувацькою грою у жанрі кіберпанк

Виконав студент 5 курсу
Групи ППЗ-51
Якіменко Дмитрій Вячеславович
Керівник роботи
Негоденко Олена Василівна

Київ – 2022

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – забезпечити користувача можливістю грати напряму у месенджері Telegram у діалозі з ботом, або у чатах.
- **Об'єкт дослідження -** геймінг на платформі телеграм за допомогу чат-бота.
- **Предмет дослідження -** чат-бот написаний на мові програмування Python.

ТЕРМІНИ

- **Кіберпанк** – піджанр наукової фантастики, що виник у 1980-х роках; концентрується на зображенні високих комп'ютерних технологій, що сусідують з низьким рівнем життя людей.
- **Корпорація** у контексті проекту – об’єднання гравців на подобі клану, яке є вигідним для його учасників та відриває нові механіки.
- **Cybermancy** – назва розробленого бота.
- **Telegram** – популярний месенджер (програма для обміну повідомленнями).
- **Telegram-бот** – спеціальний обліковий запис, який керується не людиною, а програмою за допомогою Telegram Bot API, яка і є серверною частиною бота.

АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

	Cybermancy	Village Game	Quizarium	Arena Game RPG	Werewolf
Особисті повідомлення	+	+	-	+	+
Чати	+	-	+	-	+
Вибір мови	+	+	+	+	-
Кіберпанк	+	-	-	-	-

ТЕХНІЧНІ ЗАВДАННЯ

1. Розробити можливість створення персонажа.
2. Розробити можливість покращення персонажа.
3. Розробити можливість додати бота у чат, який буде слугувати містом. Між «містами» можна буде переміщуватися за ігрову валюту.
4. Розробити можливість працювати на корпорації, щоб заробити ігрову валюту, або щось інше.
5. Розробити інші можливі дії для гравця.
6. Розробити можливість створити свою корпорацію, покращувати її, тощо.
7. Розробити можливість зміни мови для окремих користувачів та чатів.

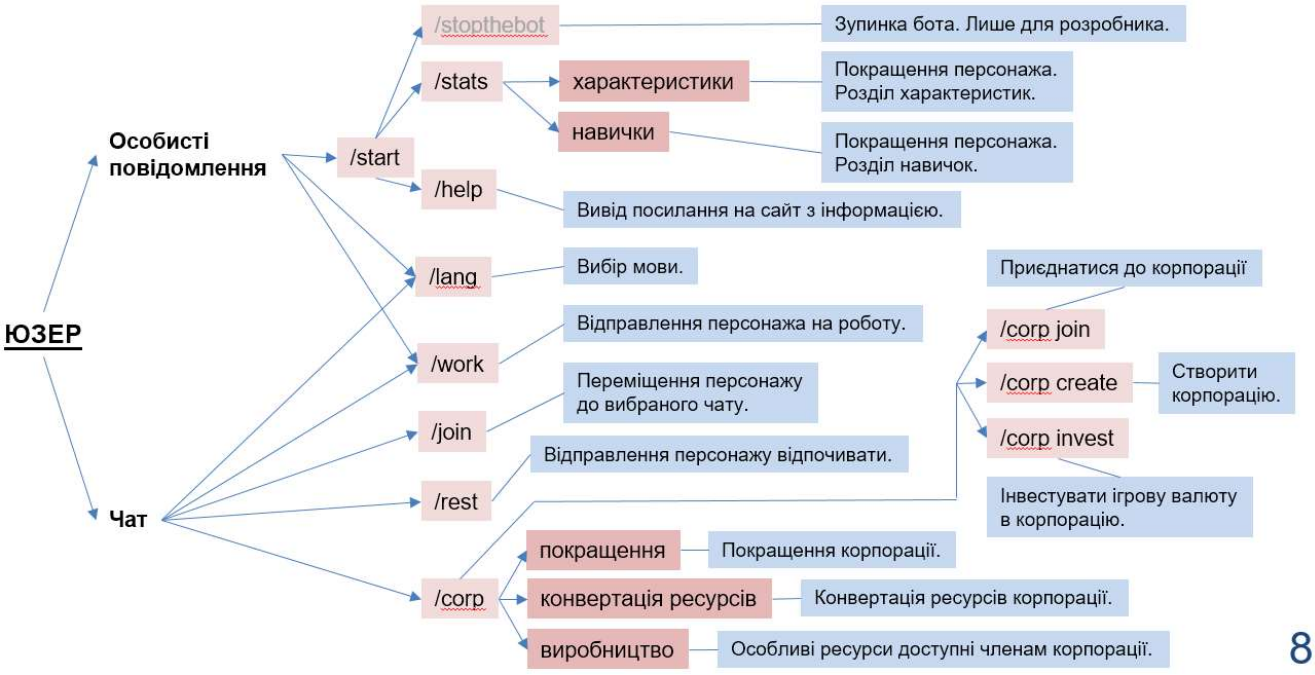
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

- pyTelegramBotAPI
- Python
- TinyDB
- PyCharm

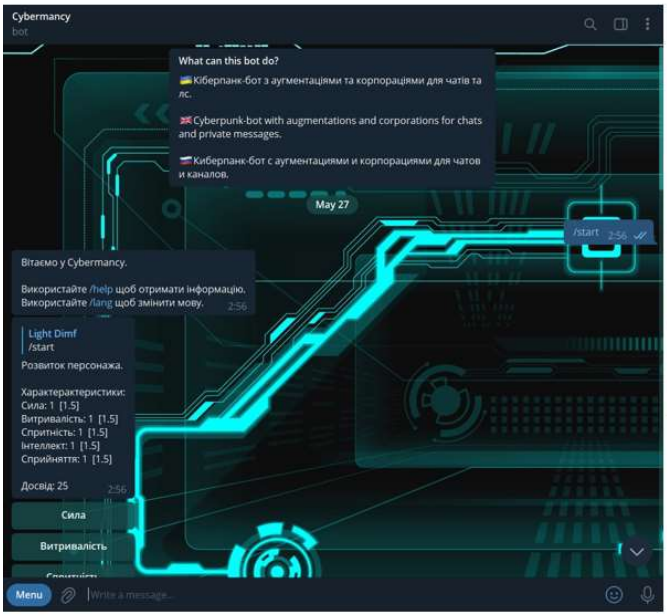
АНАЛІЗ ФАЙЛІВ ТА КЛАСІВ

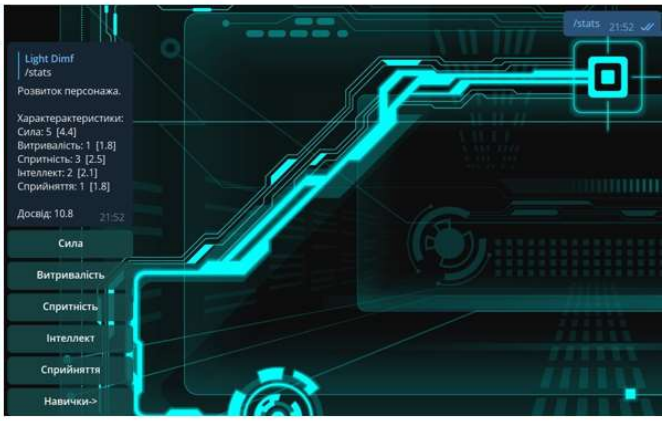
- **main.py** – початковий клас програми з якого починається його робота. Інша основна функція окрім основного циклу програми – обробка запитів від серверів Telegram, які містять введені користувачами команди, натискання кнопок, додавання або видалення за чатів, тощо.
- **Logic.py** – містить клас Logic, який містить виключно статичні функції у які була перенесена більша частина логіки з файлу main.py для збільшення чистоти його коду.
- **Common.py** – файл що містить деякі змінні, які використовуються по всіх файлах проекту (Наприклад змінна для звернення к pyTelegramBotAPI для роботи з ботом, або змінні для роботи з таблицями у базі даних)
- **Player.py, Chat_City.py, Corp.py** – файли, що містять класи Player, ChatCity, Corp відповідно. Ці класи створюються кожен раз коли треба прочитати, змінити, або опрацювати інформацію про гравців, чати, або корпорації. Містять атрибути, методи створення (з нуля, або з бази даних) та збереження (у базу даних), а також інші методи для роботи з цими класами.
- **Localisations.py** – файл який містить клас lang, який містить вкладені класи для кожного файлу проекту. Кожен з них містить безліч статичних функцій для всіх текстових рядків які використовуються для відправлення користувачам. Кожен рядок має 3 екземпляра для української, англійської та російської мов. Функції повертають рядок в залежності від налаштувать мови користувача, або чату.
- **Locations.py** – файл, що містить лише один словник для класу Chat_City.py, який містить інформацію про генеруємі локації для кожного міста.

СХЕМА РОБОТИ ТЕЛЕГРАМ-БОТА

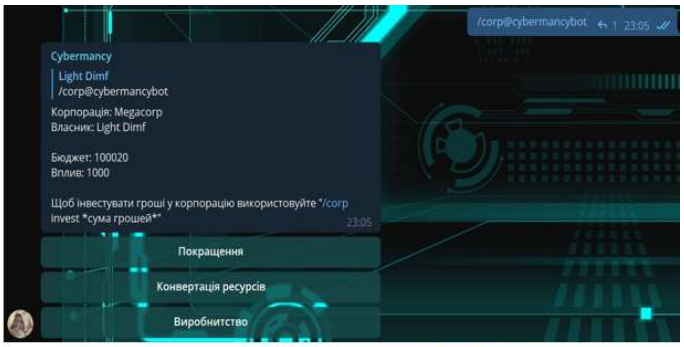


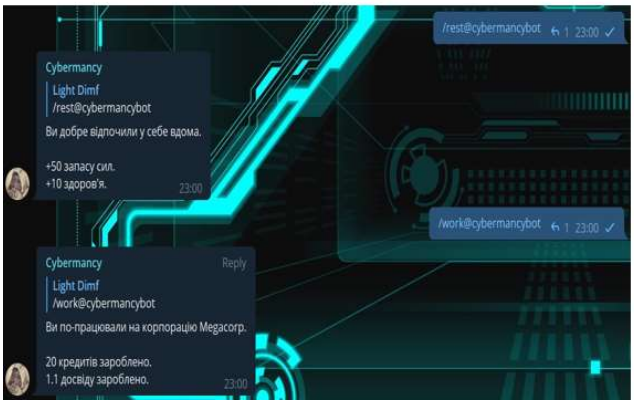
ЕКРАННІ ФОРМИ БОТУ

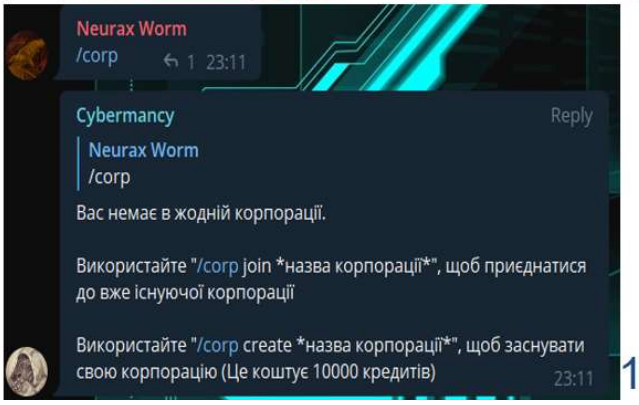




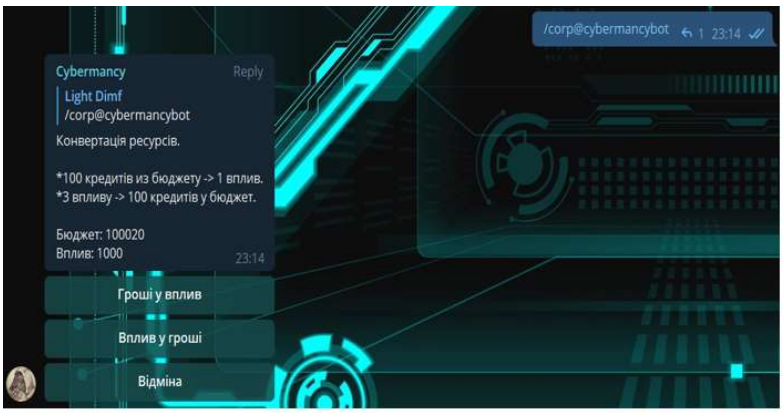
9

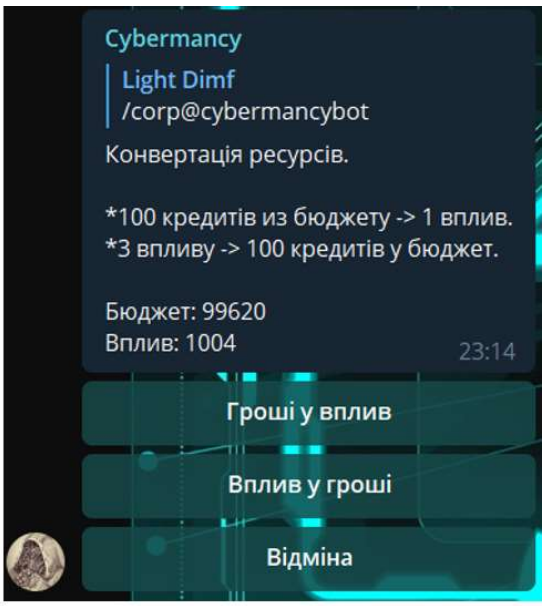






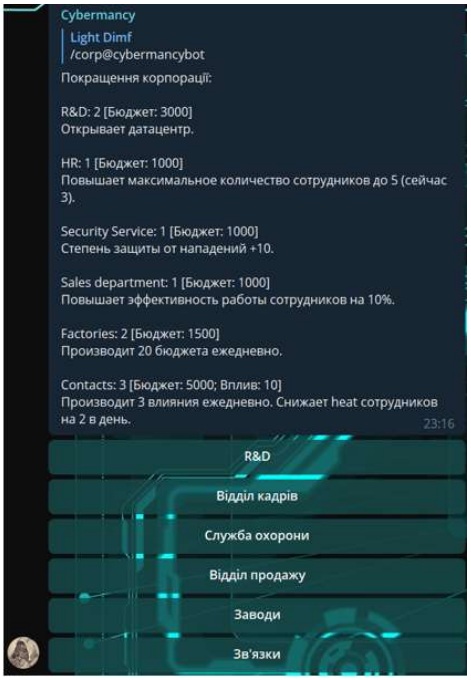
10


/corp@cybermancybot 1 23:14
Cybermancy
Reply
Light Dimf
/corp@cybermancybot
Конвертація ресурсів.
*100 кредитів из бюджету -> 1 вплив.
*3 впливу -> 100 кредитів у бюджет.
Бюджет: 100020
Вплив: 1000
23:14
Гроші у вплив
Вплив у гроші
Відміна


Cybermancy
Light Dimf
/corp@cybermancybot
Конвертація ресурсів.
*100 кредитів из бюджету -> 1 вплив.
*3 впливу -> 100 кредитів у бюджет.
Бюджет: 99620
Вплив: 1004
23:14
Гроші у вплив
Вплив у гроші
Відміна

11


/corp@cybermancybot 1 23:16
Cybermancy
Light Dimf
/corp@cybermancybot
Покращення корпорації:
R&D: 1 [Бюджет: 2500]
Открывает заводы.
HR: 1 [Бюджет: 1000]
Повышает максимальное количество сотрудников до 5 (сейчас 3).
Security Service: 1 [Бюджет: 1000]
Степень защиты от нападений +10.
Sales department: 1 [Бюджет: 1000]
Повышает эффективность работы сотрудников на 10%.
Contacts: 1 [Бюджет: 2000]
Производит 1 влияния ежедневно.
23:16
R&D
Відділ кадрів
Служба охорони
Відділ продажу
Зв'язки


Cybermancy
Light Dimf
/corp@cybermancybot
Покращення корпорації:
R&D: 2 [Бюджет: 3000]
Открывает датацентр.
HR: 1 [Бюджет: 1000]
Повышает максимальное количество сотрудников до 5 (сейчас 3).
Security Service: 1 [Бюджет: 1000]
Степень защиты от нападений +10.
Sales department: 1 [Бюджет: 1000]
Повышает эффективность работы сотрудников на 10%.
Factories: 2 [Бюджет: 1500]
Производит 20 бюджета ежедневно.
Contacts: 3 [Бюджет: 5000; Вплив: 10]
Производит 3 влияния ежедневно. Снижает heat сотрудников на 2 в день.
23:16
R&D
Відділ кадрів
Служба охорони
Відділ продажу
Заводи
Зв'язки

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Якіменко Д.В. Розробка Telegram-бота з багатокористувацькою грою у жанрі кіберпанк: матеріали Міжнар. наук.-практ. конф., м. Київ 05.04.2022/ Держ. ун-т Телекомунікацій, Ф-т інф. технологій. Київ, 2022.

ВИСНОВКИ

- Виконані всі поставлені задачі індивідуального завдання.
- Проведений аналіз існуючих телеграм-ботів з іграми.
- Розроблене технічне завдання для бота.
- Розроблений алгоритм роботи бота.
- Створений код телеграм-бота.
- Реалізовано можливість створення та покращення персонажа, інтеграція з чатами, корпорації та інше.

ДЯКУЮ ЗА УВАГУ!

Додаток Б

main.py:

```
import telebot
from Logic import Logic
from classes.Chat_City import ChatCity
from tinydb import Query
from Common import bot, PlayersTable, ChatCitiesTable, CorpsTable, dev_id,
OtherTable
from classes.Player import Player
from classes.Corp import Corp
import time
import threading
import traceback
import datetime
import json
from Localisations import lang

do_polling = True

def atStart():
    if(not CorpsTable.contains(Query().name == "Megacorp")):
        developer = Player(dev_id)
        megacorp = Corp("Megacorp", developer.ID)
        developer.corp = "Megacorp"
        developer.save()

        megacorp.budgetChange(100000)
        megacorp.influenceChange(1000)
        #megacorp.departments["R&D"] = 10  #закоментировано для тестов
```

```
        #megacorp.departments["Отдел кадров"] = 10
        #megacorp.departments["Служба охраны"] = 10
        #megacorp.departments["Химические лаборатории"] = 10
        #megacorp.departments["Отдел продаж"] = 10
        #megacorp.departments["Заводы"] = 10
        #megacorp.departments["Датацентр"] = 10
        #megacorp.departments["Связи"] = 10
        megacorp.save()

        OtherTable.insert({"name": "everyday cooldown", "last time":
datetime.date.today().isoformat()})

def base_filter(message):
    if(not PlayersTable.contains(Query().ID == message.from_user.id)):
        bot.reply_to(message, lang.main.say_start(message))
        return False
    return True

def dev_filter(message):#
    if((not PlayersTable.contains(Query().ID == message.from_user.id)) and
(message.from_user.id == dev_id)):
        return False
    return True

@bot.message_handler(chat_types=['private'], commands=['start'])
def private_start(message):
    if(not PlayersTable.contains(Query().ID == message.chat.id)):
        Player(message.chat.id).save()
        bot.send_message(message.chat.id, lang.main.welcome_to_cybermancy(message))
        Logic.statsUpgrade(message)
```

```
    else:
        bot.reply_to(message, lang.main.you_already_have_character(message))

@bot.message_handler(func=base_filter, chat_types=['private'], commands=['stats'])
def private_stats(message):
    Logic.statsUpgrade(message)

@bot.message_handler(chat_types=['private'], commands=['help']) #####
def private_help(message):
    bot.reply_to(message, "Тут має бути посилання.")

@bot.message_handler(func=base_filter, chat_types=['private'],
commands=['delete_account'])
def private_delete_account(message):
    pass ########



@bot.callback_query_handler(func=lambda call: True)
def callback(call):
    request=call.data.split() #####
    if((call.from_user.id != call.message.reply_to_message.from_user.id) and not
"FOR_EVERYONE" in request):
        bot.answer_callback_query(callback_query_id=call.id,
text=lang.main.you_already_have_character(call.message))
        return
    match request[0]:
        case "UPGRADE":
            Logic.statsUpgrade(call.message, call)
        case "WORK":
            pass
```

```
case "MAP":
    pass
case "FIGHT":
    pass
case "MARKET":
    pass
case "REST":
    pass
case "CORP":
    if(request[1] == "UPGRADE"):
        corp = Corp(PlayersTable.search(Query().ID == call.from_user.id)[0]["corp"])
        if(len(request) == 3):
            try:
                match request[2]:
                    case "R&D":
                        match corp.departments["R&D"]+1:
                            case 1:
                                corp.departmentUP("Factories")
                            case 2:
                                corp.departmentUP("Datacenter")
                            case 3:
                                corp.departmentUP("Chemical Labs")
                        corp.departmentUP("R&D")
                    case "HR":
                        corp.departmentUP("HR")
                    case "Security_Service":
                        corp.departmentUP("Security Service")
                    case "Chemical_Labs":
                        corp.departmentUP("Chemical Labs")
                    case "Sales_department":
```

```
                    corp.departmentUP("Sales department")
                case "Factories":
                    corp.departmentUP("Factories")
                case "Datacenter":
                    corp.departmentUP("Datacenter")
                case "Contacts":
                    corp.departmentUP("Contacts")
            corp.save()
        except Exception as E:
            if(E == "not_enough_budget"):
                bot.answer_callback_query(callback_query_id=call.id,
show_alert=False, text=lang.corp.not_enough_budget(call.message))
            elif(E == "not_enough_influence"):
                bot.answer_callback_query(callback_query_id=call.id,
show_alert=False, text=lang.corp.not_enough_influence(call.message))
            elif(E == "R&D_level_is_too_low"):
                bot.answer_callback_query(callback_query_id=call.id,
show_alert=False, text=lang.corp.research_level_is_too_low(call.message))
            else:
                print(traceback.format_exc())
                bot.answer_callback_query(callback_query_id=call.id,
show_alert=False, text=E)
        Logic.corpMain(call.message.reply_to_message, call)
    elif(request[1] == "CONVERT"):
        corp = Corp(PlayersTable.search(Query().ID == call.from_user.id)[0]["corp"])
        try:
            if(len(request)==3):
                if(request[2] == "MONEY"):
                    corp.budgetChange(-100)
                    corp.influenceChange(1)
```

```
                elif(request[2] == "INFLUENCE"):
                    corp.influenceChange(-3)
                    corp.budgetChange(100)
                corp.save()
            Logic.corpMain(call.message.reply_to_message, call)
        except Exception as E:
            bot.answer_callback_query(callback_query_id=call.id, text=E.args[0])
    elif(request[1] == "JOIN"):
        if(request[2] == "CANCEL"):
            bot.delete_message(call.message.chat.id, call.message.message_id)
            return
        player = Player(call.message.reply_to_message.from_user.id)
        player.corp = request[2]
        player.save()
        bot.edit_message_text(lang.main.welcome_to_corpname(call.message,
player.corp), call.message.chat.id, call.message.message_id)
    else:
        Logic.corpMain(call.message.reply_to_message, call)
case "JOIN":
    if(request[1] == "CONFIRM"):
        if(Logic.playerMoneyChange(call, -20)):
            player = Player(call.from_user.id)
            player.city = call.message.chat.id
            player.save()

            bot.edit_message_text(lang.main.welcome(call.message),
call.message.chat.id, call.message.message_id)
    else:
        bot.delete_message(call.message.chat.id, call.message.message_id)
case "PROFILE":
```

```
            pass
        case "SETTINGS":
            pass
        case "LANG":
            place: Player | ChatCity
            if(call.message.chat.type == "private"):
                place = Player(call.message.chat.id)
            else:
                place = ChatCity(call.message.chat.id)
            place.lang = request[1]
            place.save()
            bot.edit_message_text(lang.main.language_was_chosen(call.message),
call.message.chat.id, call.message.message_id)

@bot.message_handler(func=base_filter, chat_types=['private', 'supergroup'],
commands=['lang']) #вибір мови
def lang_command(message):
    if(message.chat.type == "supergroup"):
        if(bot.get_chat_member(message.chat.id, message.from_user.id).status not in
['administrator','creator']):
            bot.reply_to(message, "Ви не адміністратор.\nYou are not admin.\nВы не
администратор.")
            return
    lang_keyboard = telebot.types.InlineKeyboardMarkup()
    lang_keyboard.add(telebot.types.InlineKeyboardButton(text="UA",
callback_data="LANG ua"))
    lang_keyboard.add(telebot.types.InlineKeyboardButton(text="GB",
callback_data="LANG en"))
    lang_keyboard.add(telebot.types.InlineKeyboardButton(text="RU",
callback_data="LANG ru"))
```

```
    bot.reply_to(message, "uaВиберіть мову.\ngbChoose language.\nruВыберите язык.", reply_markup=lang_keyboard)

@bot.my_chat_member_handler(chat_types=['group'])
def group_answer(message):
    bot.send_message(message.chat.id, lang.main.only_for_supergroups(message))

@bot.my_chat_member_handler(chat_types=['supergroup'])
def my_chat_member(me):
    if(me.new_chat_member.status=="member"):
        if(not ChatCitiesTable.contains(Query().chatID == me.chat.id)):
            bot.send_message(me.chat.id, "Ласкаво просимо.\n\nWelcome.\n\nДобро пожаловать.")
            newChat = ChatCity(me.chat.id)########
            newChat.save()
        else:
            bot.send_message(me.chat.id, lang.main.city_welcome_back(me))#########

    else:
        #ChatCitiesTable.remove(me.chat.id)
        pass


@bot.message_handler(func=base_filter, chat_types=['private', 'supergroup'], commands=['work'])
def work_command(message):
    player = Player(message.from_user.id)
    if(player.cooldowns["work"] == True):
        bot.reply_to(message, lang.main.you_already_worked(message))
        return
```

```
    corp = Corp(player.corp)
    if(player.corp == "None"): #якщо не в корпорації, то працює на Megacorp
        player.moneyChange(15)
        corp.budgetChange(25)
        if(not player.staminaChange(-10)):
            bot.reply_to(message, lang.player.not_enough_stamina(message))
            return
        xp_gained=player.xpGain(1)
        bot.reply_to(message, lang.main.thanks_for_work_for_megacorp(message,
xp_gained))
    else: #якщо в корпорації, то працює на неї.
        match corp.departments["Sales department"]: #Підвищення ефективності
роботи залежно від рівня відділу продажів.
            case 0:
                player.moneyChange(20)
                corp.budgetChange(20)
            case 1:
                player.moneyChange(22)
                corp.budgetChange(22)
            case 2:
                player.moneyChange(24)
                corp.budgetChange(24)
            case 3:
                player.moneyChange(26)
                corp.budgetChange(26)
            case 4:
                player.moneyChange(28)
                corp.budgetChange(28)
            case 5:
                player.moneyChange(30)
```

```
            corp.budgetChange(30)
        case 6:
            player.moneyChange(32)
            corp.budgetChange(32)
        case 7:
            player.moneyChange(34)
            corp.budgetChange(34)
        case 8:
            player.moneyChange(36)
            corp.budgetChange(36)
        case 9:
            player.moneyChange(38)
            corp.budgetChange(38)
        case 10:
            player.moneyChange(40)
            corp.budgetChange(40)
    if(not player.staminaChange(-10)):
        bot.reply_to(message, lang.player.not_enough_stamina(message))
        return
    xp_gained=player.xpGain(1)
    bot.reply_to(message, lang.main.thanks_for_work(message, player.corp,
xp_gained))
  player.work_cooldown()
  player.save()
  corp.save()


@bot.message_handler(func=base_filter, chat_types=['supergroup'], commands=['map'])
#походы одному, или командой по локациям
def map_command(message):
```

```
    pass

@bot.message_handler(func=base_filter, chat_types=['supergroup'],
commands=['fight']) #пвп
def fight_command(message):
    pass

@bot.message_handler(func=base_filter, chat_types=['supergroup'],
commands=['market'])
def market_command(message):
    pass

@bot.message_handler(func=base_filter, chat_types=['private', 'supergroup'],
commands=['rest'])
def rest_command(message):
    player = Player(message.from_user.id)
    if(player.cooldowns["rest"]==True):
        lang.main.you_already_rested(message)
        return
    player.staminaChange(50)
    player.healthChange(10)
    player.save()
    bot.reply_to(message, lang.main.basic_rest(message))

@bot.message_handler(func=base_filter, chat_types=['supergroup'], commands=['c
def corp_command(message):
    corpRequest = message.text.split()
    if(len(corpRequest)>1):
        Logic.corpCommand(message)
```

```
    elif(PlayersTable.search(Query().ID == message.from_user.id)[0]["corp"] ==
"None"):
        bot.reply_to(message, lang.main.you_are_not_in_any_corporation(message))#
    else:
        Logic.corpMain(message)

@bot.message_handler(func=base_filter, chat_types=['supergroup'], commands=['join'])
def join_command(message):
    ChatCity(message.chat.id).getPlayers()
    if(PlayersTable.search(Query().ID == message.from_user.id)[0]["city"] !=
message.chat.id):
        Logic.join(message)
    else:
        bot.reply_to(message, lang.main.you_are_already_here(message))

@bot.message_handler(func=base_filter, chat_types=['supergroup'],
commands=['profile']) #профиль, инвентарь, статистика и всё такое.
def profile_command(message):
    pass

@bot.message_handler(func=base_filter, chat_types=['supergroup'],
commands=['settings']) #настройки чата для админов.
def settings_command(message):
    pass

@bot.message_handler(func=dev_filter, chat_types=['private'],
commands=['stopthebot'])
def stopthebot_command(message):
    raise Exception("stopthebot")
```

```
#@bot.message_handler(chat_types=['supergroup'])
#def public_chat_test(message):
    #print(bot.get_chat(message.chat.id))
    #bot.send_message(message.chat.id, message.from_user.id)
    #print(message)




def scheduled_tasks():
    while True:
        if(OtherTable.search(Query().name == "everyday cooldown")[0]["last time"] !=
json.dumps(datetime.date.today().isoformat())):
            OtherTable.update({"last time": json.dumps(datetime.date.today().isoformat())},
Query().name == "everyday cooldown")
            PlayersTable.update({"work": False}, Query().cooldowns.work == True)
            PlayersTable.update({"rest": False}, Query().cooldowns.rest == True)
            print("Everyday task is done.")
        time.sleep(30)



if __name__ == '__main__':
    atStart()
    thr = threading.Thread(target = scheduled_tasks)
    thr.daemon = True
    thr.start()
    while True:
        try:
            if(do_polling):
```

```
            bot.polling(non_stop=True)
        except Exception as E:
            if(E.args[0]=="stopthebot"):
                break
            else:
                print(traceback.format_exc())
        time.sleep(1)

    print("end")
```

Localisations.py:

```
from Common import PlayersTable, ChatCitiesTable
from tinydb import Query

class lang:
    """Методи - str в різних місцях програми. Приймають повідомлення аргументом."""

    @staticmethod
    def _checkLang(message):
        if(message.chat.type=="private"):
            return PlayersTable.search(Query().ID == message.chat.id)[0]["lang"]
        else:
            return ChatCitiesTable.search(Query().chatID == message.chat.id)[0]["lang"]

    class main:
        @staticmethod
        def say_start(message):
            text = {
                "ua": "Напишіть боту /start для створення персонажа.",
```

```
            "en": "Type /start to the bot to create a character.",
            "ru": "Напише боту /start в лс для создания персонажа."
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def welcome_to_cybermancy(message):
        text = {
            "ua": "Вітаємо у Cybermancy.\n\nВикористайте /help щоб отримати інформацію.\nВикористайте /lang щоб змінити мову.",
            "en": "Welcome to the Cybermancy.\n\nUse /help to get info.\nUse /lang to change language.",
            "ru": "Добро пожаловать в Cybermancy.\n\nИспользуйте /help чтобы получить информацию.\nИспользуйте /lang чтобы изменить язык."
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def you_already_have_character(message):
        text = {
            "ua": "У вас вже є персонаж.\nДля допомоги використайте /help\nЩоб покращити персонажа використайте /stats",
            "en": "You already have a character.\nTo get help use /help\nTo upgrade a character use /stats",
            "ru": "У вас уже есть персонаж.\nДля помощи используйте /help\nЧтобы улучшить персонажа используйте /stats"
        }
        return text[lang._checkLang(message)]

    @staticmethod
```

```
    def not_for_you(message):
        text = {
            "ua": "Це не для вас",
            "en": "It's not for you",
            "ru": "Это не для вас"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def welcome(message):
        text = {
            "ua": "Ласкаво просимо.",
            "en": "Welcome.",
            "ru": "Добро пожаловать."
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def only_for_supergroups(message):
        text = {
            "ua": "Нажаль [на данный момент?] бот працює тільки для
супергруп.\nБудь ласка, зробіть історію чата видимою для всіх користувачів (х
б раз), або зробіть групу відкритою.",
            "en": "Sorry, but [at least for this moment?] bot is working only for
supergroups.\nPlease, make chat history visible for all users (at least once), or make
group public.",
            "ru": "К сожалению [на данный момент?] бот работает только для
супергрупп.\nПожалуйста, сделайте историю чата видимой для новых
пользователей (хоть раз), или сделайте группу открытой."
        }
```

```
        return text[lang._checkLang(message)]

    #@staticmethod
    #def city_welcome(message):
    #    text = {
    #        "ua": "Ласкаво просимо.",
    #        "en": "Welcome.",
    #        "ru": "Добро пожаловать."
    #    }
    #    return text[lang._checkLang(message)]

    @staticmethod
    def city_welcome_back(message):
        text = {
            "ua": "Ласкаво просимо до Cybermancy знову.",
            "en": "Welcome to Cybermancy again.",
            "ru": "Добро пожаловать в Cybermancy снова."
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def you_are_not_in_any_corporation(message):
        text = {
            "ua": ("Вас немає в жодній корпорації.\n\n"
                   "Використайте \"/corp join *назва корпорації*\", щоб приєднатися до вже існуючої корпорації\n\n"
                   "Використайте \"/corp create *назва корпорації*\", щоб заснувати свою корпорацію (Це коштує 10000 кредитів)"),
            "en": ("You are not in any corporation.\n\n"
```

```
                "Use \"/corp join *corp name*\", to join already existing
corporation.\n\n"
                "Use \"/corp create *corp name*\", to create your own corporation (It
costs 10000 credits)"),
            "ru": ("Вы не состоите ни в одной корпорации.\n\n"
                "Используйте \"/corp join *название корпорации*\", чтобы
присоединиться к уже существующей корпорации\n\n"
                "Используйте \"/corp create *название корпорации*\", чтобы основать
свою корпорацию (Это стоит 10000 кредитов)")
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def you_are_already_here(message):
        text = {
            "ua": "Ви вже знаходитесь тут.",
            "en": "You are already here.",
            "ru": "Вы уже находитесь здесь."
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def language_was_chosen(message):
        text = {
            "ua": "UAУкраїнська мова вибрана мовою інтерфейсу.",
            "en": "GBEnglish was chosen as interface language.",
            "ru": "RUРусский язык был выбран языком интерфейса."
        }
        return text[lang._checkLang(message)]
```

```
    @staticmethod
    def you_already_worked(message):
        text = {
            "ua": "Ви вже працювали сьогодні.",
            "en": "You already worked today.",
            "ru": "Вы уже работали сегодня."
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def thanks_for_work(message, corp_name, xp_gained):
        text = {
            "ua": "Ви по-працювали на корпорацію {}.\n\n20 кредитів зароблено.\n{}
досвіду зароблено.",
            "en": "You worked for {} corp.\n\n20 credits gained.\n{} xp gained.",
            "ru": "Вы поработали на корпорацию {}.\n\n20 кредитов заработано.\n{}
опыта заработано."
        }
        return text[lang._checkLang(message)].format(corp_name, xp_gained)

    @staticmethod
    def thanks_for_work_for_megacorp(message, xp_gained):
        text = {
            "ua": "Дякуємо за роботу на Megacorp.\nВаша робота важлива для
нас.\n\n15 кредитів зароблено.\n{} досвіду зароблено.",
            "en": "Thanks for your work for Megacorp.\nYour work is important to
us.\n\n15 credits is gained.\n{} xp gained.",
            "ru": "Спасибо за работу на Megacorp.\nВаша работа важна для
нас.\n\n15 кредитов заработано.\n{} опыта заработано."
        }
```

```
        return text[lang._checkLang(message)].format(xp_gained)

    @staticmethod
    def welcome_to_corpname(message, corp_name):
        text = {
            "ua": "Вітаємо у корпорації {}!",
            "en": "Welcome to {}!",
            "ru": "Добро пожаловать в {}!"
        }
        return text[lang._checkLang(message)].format(corp_name)

    @staticmethod
    def basic_rest(message):
        text = {
            "ua": "Ви добре відпочили у себе вдома.\n\n+50 запасу сил.\n+10
здоров'я.",
            "en": "You are well rested in your basement.\n\n+50 stamina.\n+10 health.",
            "ru": "Вы хорошо отдохнули у себя дома.\n\n+50 запаса сил.\n+10
здоровья."
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def you_already_rested(message):
        text = {
            "ua": "Ви вже відпочивали сьогодні.",
            "en": "You already rested today.",
            "ru": "Вы уже отдыхали сегодня."
        }
        return text[lang._checkLang(message)]
```

```
class logic:
    @staticmethod
    def error(message):
        text = {
            "ua": "Помилка",
            "en": "Error.",
            "ru": "Ошибка."
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def button_move_to_skills(message):
        text = {
            "ua": "Навички->",
            "en": "Skills->",
            "ru": "Навыки->"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def button_move_to_attributes(message):
        text = {
            "ua": "<-Характеристики",
            "en": "<-Attributes",
            "ru": "<-Характеристики"
        }
        return text[lang._checkLang(message)]
```

```
    @staticmethod
    def upgrade_attribute(message, player):
        text = {
            "ua": ("Розвиток персонажа.\n\nХарактерактеристики:\n" +
                   "Сила: {}  [{}]\nВитривалість: {}  [{}]\n" +
                   "Спритність: {}  [{}]\nІнтеллект: {}  [{}]\n" +
                   "Сприйняття: {}  [{}]\n" +
                   ("Псі: {psi}  [{psi_cost}]\n" if "PSI" in player.augmentations else "") +
###

                   "\nДосвід: {}"),
            "en": ("Character upgrade.\n\nAttributes:\n" +
                   "Strength: {}  [{}]\nEndurance: {}  [{}]\n" +
                   "Agility: {}  [{}]\nIntelligence: {}  [{}]\n" +
                   "Perception: {}  [{}]\n" +
                   ("PSI: {psi}  [{psi_cost}]\n" if "PSI" in player.augmentations else "") +
###

                   "\nXP: {}"),
            "ru": ("Развитие персонажа.\n\nХарактерактеристики:\n" +
                   "Сила: {}  [{}]\nВыносливость: {}  [{}]\n" +
                   "Ловкость: {}  [{}]\nИнтеллект: {}  [{}]\n" +
                   "Восприятие: {}  [{}]\n" +
                   ("Пси: {psi}  [{psi_cost}]\n" if "PSI" in player.augmentations else "") +
###

                   "\nОпыт: {}")
        }
        return text[lang._checkLang(message)].format(player.STR_base,
player.XP_cost("STR"), player.END_base, player.XP_cost("END"),
```

```
                                    player.AGI_base, player.XP_cost("AGI"),
player.INT_base, player.XP_cost("INT"),
                                    player.PER_base, player.XP_cost("PER"), player.xp,
psi=player.PSI_base, psi_cost=player.XP_cost("PSI"))

    @staticmethod
    def upgrade_skills(message, player):
        text = {
            "ua": ("Розвиток персонажа.\n\nНавыки:\n" +
                   "Хакерство: {}  [{}]\nБіотехнології: {}  [{}]\n" +
                   "Електротехніка: {}  [{}]\nКрасномовність: {}  [{}]\n" +
                   "Дальній бій: {}  [{}]\nБлижній бій: {}  [{}]\n" +
                   "Скритність: {}  [{}]\nВиживання: {}  [{}]\n" +

                   "\nДосвід: {}"),
            "en": ("Character upgrade.\n\nSkills:\n" +
                   "Hacking: {}  [{}]\nBiotechnology: {}  [{}]\n" +
                   "Electrotech: {}  [{}]\nNegotiation: {}  [{}]\n" +
                   "Ranged: {}  [{}]\nMelee: {}  [{}]\n" +
                   "Stealth: {}  [{}]\nSurvival: {}  [{}]\n" +

                   "\nXP: {}"),
            "ru": ("Развитие персонажа.\n\nНавыки:\n" +
                   "Хакерство: {}  [{}]\nБиотехнологии: {}  [{}]\n" +
                   "Электротехника: {}  [{}]\nКрасноречие: {}  [{}]\n" +
                   "Дальний бой: {}  [{}]\nБлижний бой: {}  [{}]\n" +
                   "Скрытность: {}  [{}]\nВыживание: {}  [{}]\n" +

                   "\nОпыт: {}")
        }
```

```
        return text[lang._checkLang(message)].format(player.HACK_base,
player.XP_cost("HACK"), player.BIOTECH_base, player.XP_cost("BIOTECH"),
                                                player.ELECTROTECH_base,
player.XP_cost("ELECTROTECH"), player.NEGOTIATION_base,
player.XP_cost("NEGOTIATION"),
                                                player.RANGED_base,
player.XP_cost("RANGED"), player.MELEE_base, player.XP_cost("MELEE"),
                                                player.STEALTH_base,
player.XP_cost("STEALTH"), player.SURVIVAL_base,
player.XP_cost("SURVIVAL"), player.xp)

    @staticmethod
    def button_join_move(message):
        text = {
            "ua": "Переміститись",
            "en": "Move",
            "ru": "Переместиться"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def cancel(message):
        text = {
            "ua": "Відміна",
            "en": "Cancel",
            "ru": "Отмена"
        }
        return text[lang._checkLang(message)]

    @staticmethod
```

```
    def trip_will_cost(message):
        text = {
            "ua": "Подорож до цього міста буде коштувати 20 кредитів.
Продовжити?",
            "en": "Trip to this city will cost 20 credits. Continue?",
            "ru": "Путешествие в этот город будет стоить 20 кредитов.
Продолжить?"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def corp_main_upgrade(message):
        text = {
            "ua": "Покращення",
            "en": "Upgrade",
            "ru": "Улучшения"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def corp_main_convert(message):
        text = {
            "ua": "Конвертація ресурсів",
            "en": "Convert resources",
            "ru": "Ковертация ресурсов"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def corp_main_production(message):
```

```
        text = {
            "ua": "Виробнитство",
            "en": "Production",
            "ru": "Производство"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def corp_main_message(message, corp, owner):
        text = {
            "ua": "Корпорація: {}\nВласник: {} {}\n\nБюджет: {}\nВплив: {}"+
                "\n\nЩоб інвестувати гроші у корпорацію використовуйте \"/corp
invest *сума грошей*\"",
            "en": "Corporation: {}\nOwner: {} {}\n\nBudget: {}\nInfluence: {}"+
                "\n\nTo invest money use \"/corp invest *amount of money*\"",
            "ru": "Корпорация: {}\nВладелец: {} {}\n\nБюджет: {}\nВлияние: {}"+
                "\n\nЧтобы инвестировать деньги в корпорацию используйте \"/corp
invest *сумма денег*\""
        }
        return text[lang._checkLang(message)].format(corp.name, owner.first_name,
owner.last_name, corp.resources["Budget"], corp.resources["Influence"])

    @staticmethod
    def corp_convert_money(message):
        text = {
            "ua": "Гроші у вплив",
            "en": "Money to influence",
            "ru": "Деньги во влияние"
        }
        return text[lang._checkLang(message)]
```

```
    @staticmethod
    def corp_convert_influence(message):
        text = {
            "ua": "Вплив у гроші",
            "en": "Influence to money",
            "ru": "Влияние в деньги"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def corp_convert_recources_message(message, corp):
        text = {
            "ua": "Конвертація ресурсів.\n\n*100 кредитів из бюджету -> 1 вплив.\n*3 впливу -> 100 кредитів у бюджет.\n\nБюджет: {}\nВплив: {}",
            "en": "Recource convertion.\n\n*100 credits to budget -> 1 influence.\n*3 influence -> 100 credits to budget.\n\nBudget: {}\nInfluence: {}",
            "ru": "Конвертация ресурсов.\n\n*100 кредитов из бюджета -> 1 влияние.\n*3 влияния -> 100 кредитов в бюджет.\n\nБюджет: {}\nВлияние: {}"
        }
        return text[lang._checkLang(message)].format(corp.resources["Budget"], corp.resources["Influence"])

    @staticmethod
    def corp_wrong_name(message):
        text = {
            "ua": "Хибна назва корпорації.",
            "en": "Wrong corp name.",
            "ru": "Неверное название корпорации."
        }
```

```
        return text[lang._checkLang(message)]

    @staticmethod
    def you_already_in_corp(message):
        text = {
            "ua": "Ви вже знаходитесь у корпорації.\n\nВикористайте \"/corp leave\",
щоб покинути корпорацію.",
            "en": "You are already in corp.\n\nUse \"/corp leave\", to leave a corporation.",
            "ru": "Вы уже находитесь в корпорации.\n\nиспользуйте \"/corp leave\",
чтобы покинуть корпорацию."
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def not_enough_money(message):
        text = {
            "ua": "Недостатньо коштів.",
            "en": "Not enough money.",
            "ru": "Недостаточно денег."
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def corp_created(message, corp):
        text = {
            "ua": "Корпорація \"{}\" успішно створена.",
            "en": "Corp \"{}\" successfully created.",
            "ru": "Корпорация \"{}\" успешно создана."
        }
        return text[lang._checkLang(message)].format(corp.name)
```

```
@staticmethod
def such_corp_already_exist(message):
    text = {
        "ua": "Така корпорація вже існує.",
        "en": "Such corporation already exist.",
        "ru": "Такая корпорация уже существует."
    }
    return text[lang._checkLang(message)]

@staticmethod
def you_are_not_in_corporation(message):
    text = {
        "ua": "Ви не знаходитесь ні в одній корпорації.",
        "en": "You are not in any corporation.",
        "ru": "Вы не находитесь ни в одной корпорации."
    }
    return text[lang._checkLang(message)]

@staticmethod
def you_cant_invest_negitive(message):
    text = {
        "ua": "Ви не можете інвестувати від'ємну суму грошей.",
        "en": "You can't invest negative amount of money.",
        "ru": "Вы не можете инветсировать отрицательное количество денег."
    }
    return text[lang._checkLang(message)]

@staticmethod
def some_credits_invested(message):
```

```
        text = {
            "ua": " кредитів успішно інвестовано.",
            "en": " credits successfully invested.",
            "ru": " кредитов успешно инвестировано."
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def are_you_sure_you_want_to_join_this_corp(message):
        text = {
            "ua": "Ви впевнені, що хочете приєднатися до цієї корпорації?",
            "en": "Are you sure you want to join this corp?",
            "ru": "Вы уверены, что хотите присоединиться к этой корпорации?"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def yes(message):
        text = {
            "ua": "Так",
            "en": "Yes",
            "ru": "Да"
        }
        return text[lang._checkLang(message)]

class player:
    @staticmethod
    def STR(message):
        text = {
            "ua": "Сила",
```

```
            "en": "Strength",
            "ru": "Сила"
        }
        return text[lang._checkLang(message)]
    @staticmethod
    def END(message):
        text = {
            "ua": "Витривалість",
            "en": "Endurance",
            "ru": "Выносливость"
        }
        return text[lang._checkLang(message)]
    @staticmethod
    def AGI(message):
        text = {
            "ua": "Спритність",
            "en": "Agility",
            "ru": "Ловкость"
        }
        return text[lang._checkLang(message)]
    @staticmethod
    def INT(message):
        text = {
            "ua": "Інтеллект",
            "en": "Intelligence",
            "ru": "Интеллект"
        }
        return text[lang._checkLang(message)]
    @staticmethod
    def PER(message):
```

```
    text = {
        "ua": "Сприйняття",
        "en": "Perception",
        "ru": "Восприятие"
    }
    return text[lang._checkLang(message)]
@staticmethod
def PSI(message):
    text = {
        "ua": "Псі",
        "en": "PSI",
        "ru": "Пси"
    }
    return text[lang._checkLang(message)]
@staticmethod
def HACK(message):
    text = {
        "ua": "Хакерство",
        "en": "Hacking",
        "ru": "Хакерство"
    }
    return text[lang._checkLang(message)]
@staticmethod
def BIOTECH(message):
    text = {
        "ua": "Біотехнології",
        "en": "Biotech",
        "ru": "Биотехнологии"
    }
    return text[lang._checkLang(message)]
```

```
@staticmethod
def ELECTROTECH(message):
    text = {
        "ua": "Електротехнології",
        "en": "Electrotech",
        "ru": "Электротехнологии"
    }
    return text[lang._checkLang(message)]
@staticmethod
def NEGOTIATION(message):
    text = {
        "ua": "Красномовність",
        "en": "Negotiation",
        "ru": "Красноречие"
    }
    return text[lang._checkLang(message)]
@staticmethod
def RANGED(message):
    text = {
        "ua": "Дальній бій",
        "en": "Ranged",
        "ru": "Дальний бой"
    }
    return text[lang._checkLang(message)]
@staticmethod
def MELEE(message):
    text = {
        "ua": "Ближній бій",
        "en": "Melee",
        "ru": "Ближний бой"
```

```
        }
        return text[lang._checkLang(message)]
    @staticmethod
    def STEALTH(message):
        text = {
            "ua": "Скритність",
            "en": "Stealth",
            "ru": "Скрытность"
        }
        return text[lang._checkLang(message)]
    @staticmethod
    def SURVIVAL(message):
        text = {
            "ua": "Виживання",
            "en": "Survival",
            "ru": "Выживание"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def not_enough_stamina(message):
        text = {
            "ua": "Недостатньо витривалості.",
            "en": "Not enough stamina.",
            "ru": "Недостаточно выносливости."
        }
        return text[lang._checkLang(message)]


class city:
```

```
    pass


class corp:
    @staticmethod
    def not_enough_budget(message):
        text = {
            "ua": "Недостатньо коштів у бюджеті",
            "en": "Not enough budget",
            "ru": "Недостаточно денег в бюджете"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def not_enough_influence(message):
        text = {
            "ua": "Недостатньо впливу",
            "en": "Not enough influence",
            "ru": "Недостаточно влияния"
        }
        return text[lang._checkLang(message)]

    @staticmethod
    def HR(message):
        text = {
            "ua": "Відділ кадрів",
            "en": "HR",
            "ru": "Отдел кадров"
        }
        return text[lang._checkLang(message)]
```

```
@staticmethod
def Security_Service(message):
    text = {
        "ua": "Служба охорони",
        "en": "Security Service",
        "ru": "Отдел кадров"
    }
    return text[lang._checkLang(message)]

@staticmethod
def Chemical_Labs(message):
    text = {
        "ua": "Хімічні лабораторії",
        "en": "Chemical labs",
        "ru": "Химические лаборатории"
    }
    return text[lang._checkLang(message)]

@staticmethod
def Sales_department(message):
    text = {
        "ua": "Відділ продажу",
        "en": "Sales department",
        "ru": "Отдел продаж"
    }
    return text[lang._checkLang(message)]

@staticmethod
def Factories(message):
    text = {
```

```
        "ua": "Заводи",
        "en": "Factories",
        "ru": "Заводы"
    }
    return text[lang._checkLang(message)]

@staticmethod
def Datacenter(message):
    text = {
        "ua": "Датацентр",
        "en": "Datacenter",
        "ru": "Датацентр"
    }
    return text[lang._checkLang(message)]

@staticmethod
def Contacts(message):
    text = {
        "ua": "Зв'язки",
        "en": "Contacts",
        "ru": "Связи"
    }
    return text[lang._checkLang(message)]

@staticmethod
def departmentUpgradeInfoText(message, corp, dep):
    text = {
        "ua": ("{}: {} [Бюджет: {}"+
```

```
                ((", Вплив: {influence}") if
corp.departmentsPriceList[dep][int(corp.departments[dep]+1)]["price"]["Influence"]>0
else "")+
                "]\n{}"),
            "en": ("{}: {} [Budget: {}"+
                ((", Influence: {influence}") if
corp.departmentsPriceList[dep][int(corp.departments[dep]+1)]["price"]["Influence"]>0
else "")+
                "]\n{}"),
            "ru": ("{}: {} [Бюджет: {}"+
                ((", Влияние: {influence}") if
corp.departmentsPriceList[dep][int(corp.departments[dep]+1)]["price"]["Influence"]>0
else "")+
                "]\n{}")
        }
        return text[lang._checkLang(message)].format(dep, corp.departments[dep]+1,

corp.departmentsPriceList[dep][int(corp.departments[dep]+1)]["price"]["Budget"],

corp.departmentsPriceList[dep][corp.departments[dep]+1]["description"],

influence=corp.departmentsPriceList[dep][int(corp.departments[dep]+1)]["price"]["Infl
uence"])
    @staticmethod
    def research_level_is_too_low(message):
        text = {
            "ua": "Рівень відділу розробок недостатній.",
            "en": "R&D level is not enough.",
            "ru": "Уровень отдела разработок недостаточен."
        }
```

```
        return text[lang._checkLang(message)]

    class locations:
        pass
```

Logic.py:

```
from classes.Player import Player
from classes.Corp import Corp
import telebot
from Common import bot, PlayersTable, CorpsTable
from tinydb import Query
from Localisations import lang

class Logic:
    @staticmethod
    def statsUpgrade(message, callback=None):
        player = Player(message.chat.id)
        request = None if callback is None else callback.data.split()
        if(request!=None):
            try:
                match request[2]:
                    case "STR": player.statUP("STR")
                    case "END": player.statUP("END")
                    case "AGI": player.statUP("AGI")
                    case "INT": player.statUP("INT")
                    case "PER": player.statUP("PER")
                    case "PSI": player.statUP("PSI")
                    case "HACK": player.statUP("HACK")
                    case "BIOTECH": player.statUP("BIOTECH")
                    case "ELECTROTECH": player.statUP("ELECTROTECH")
```

```
                case "NEGOTIATION": player.statUP("NEGOTIATION")
                case "RANGED": player.statUP("RANGED")
                case "MELEE": player.statUP("MELEE")
                case "STEALTH": player.statUP("STEALTH")
                case "SURVIVAL": player.statUP("SURVIVAL")
                case "MOVE":
                    pass
                case _:
                    bot.answer_callback_query(callback_query_id=callback.id,
show_alert=False, text="Error")
            except Exception as E:
                bot.answer_callback_query(callback_query_id=callback.id, show_alert=False,
text=E.args[0])
                return None
            player.save()

        if(request==None or request[1]=="ATTRIBUTE"):
            keyboard = telebot.types.InlineKeyboardMarkup()

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.STR(message),
callback_data="UPGRADE ATTRIBUTE STR"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.END(messag
callback_data="UPGRADE ATTRIBUTE END"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.AGI(message),
callback_data="UPGRADE ATTRIBUTE AGI"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.INT(message),
callback_data="UPGRADE ATTRIBUTE INT"))
```

```
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.PER(message),
callback_data="UPGRADE ATTRIBUTE PER"))
        if("PSI" in player.augmentations): ###

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.PSI(message),
callback_data="UPGRADE PSI"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.button_move_to_skil
ls(message), callback_data="UPGRADE SKILL MOVE")) #
        if(request==None):
            bot.reply_to(message, lang.logic.upgrade_attribute(message, player),
reply_markup=keyboard)
        else:
            bot.edit_message_text(lang.logic.upgrade_attribute(message, player),
message.chat.id, callback.message.message_id, reply_markup=keyboard)
    elif(request[1]=="SKILL"):
        keyboard = telebot.types.InlineKeyboardMarkup()

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.HACK(message),
callback_data="UPGRADE SKILL HACK"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.BIOTECH(m
, callback_data="UPGRADE SKILL BIOTECH"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.ELECTROTECH(
message), callback_data="UPGRADE SKILL ELECTROTECH"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.NEGOTIATION(m
essage), callback_data="UPGRADE SKILL NEGOTIATION"))
```

```
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.RANGED(message)
, callback_data="UPGRADE SKILL RANGED"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.MELEE(message),
callback_data="UPGRADE SKILL MELEE"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.STEALTH(message)
, callback_data="UPGRADE SKILL STEALTH"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.player.SURVIVAL(messa
ge), callback_data="UPGRADE SKILL SURVIVAL"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.button_move_to_attri
butes(message), callback_data="UPGRADE ATTRIBUTE MOVE")) #
            bot.edit_message_text(lang.logic.upgrade_skills(message, player),
message.chat.id, callback.message.message_id, reply_markup=keyboard)
        else:
            pass

    @staticmethod
    def join(message):
        keyboard = telebot.types.InlineKeyboardMarkup()
        CONFIRM_button =
telebot.types.InlineKeyboardButton(text=lang.logic.button_join_move(message),
callback_data="JOIN CONFIRM")
        CANCEL_button =
telebot.types.InlineKeyboardButton(text=lang.logic.cancel(message),
callback_data="JOIN CANCEL")
        keyboard.add(CONFIRM_button, CANCEL_button)
```

```
        bot.reply_to(message, lang.logic.trip_will_cost(message), reply_markup=keyboard)

    @staticmethod
    def playerMoneyChange(call, amount: int):
        try:
            player = Player(call.from_user.id)
            player.moneyChange(amount)
            player.save()
            return True
        except Exception as E:
            if(E.args[0] == "not_enough_money"):
                bot.answer_callback_query(callback_query_id=call.id, show_alert=False,
text=lang.logic.not_enough_money(call.message))
            else:
                bot.answer_callback_query(callback_query_id=call.id, show_alert=False,
text=E.args[0])
            return False

    @staticmethod
    def corpMain(message, callback=None):
        corp = Corp(PlayersTable.search(Query().ID == message.from_user.id)[0]["c
        owner = bot.get_chat_member(message.chat.id, message.from_user.id).user
        keyboard = telebot.types.InlineKeyboardMarkup()
        request = None if callback is None else callback.data.split()
        if(request == None or request[1]=="MAIN"):
            if(owner.id == message.from_user.id):

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.corp_main_upgrade(
message), callback_data="CORP UPGRADE"))
```

```
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.corp_main_convert(
message), callback_data="CORP CONVERT"))

keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.corp_main_productio
n(message), callback_data="CORP PRODUCTION"))
        bot.reply_to(message, lang.logic.corp_main_message(message, corp, owner),
reply_markup=keyboard)
    elif(request[1]=="UPGRADE"):
        upgrade_keyboard = telebot.types.InlineKeyboardMarkup()
        msg_text = "Покращення корпорації:\n\n"
        if(corp.departments["R&D"]<10):
            msg_text += corp.departmentUpgradeInfo("R&D", message)+"\n\n"
            upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text="R&D",
callback_data="CORP UPGRADE R&D"))
        if(corp.departments["HR"]<10):
            msg_text += corp.departmentUpgradeInfo("HR", message)+"\n\n"

upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.HR(message)
, callback_data="CORP UPGRADE HR"))
        if(corp.departments["Security Service"]<10):
            msg_text += corp.departmentUpgradeInfo("Security Service",
message)+"\n\n"

upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Security_Ser
vice(message), callback_data="CORP UPGRADE Security_Service"))
        if(0<corp.departments["Chemical Labs"]<10):
            msg_text += corp.departmentUpgradeInfo("Chemical Labs", message)+"\n\n"
```

```
upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Chemical_L
abs(message), callback_data="CORP UPGRADE Chemical_Labs"))
        if(corp.departments["Sales department"]<10):
            msg_text += corp.departmentUpgradeInfo("Sales department",
message)+"\n\n"

upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Sales_depart
ment(message), callback_data="CORP UPGRADE Sales_department"))
        if(0<corp.departments["Factories"]<10):
            msg_text += corp.departmentUpgradeInfo("Factories", message)+"\n\n"

upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Factories(m
essage), callback_data="CORP UPGRADE Factories"))
        if(0<corp.departments["Datacenter"]<10):
            msg_text += corp.departmentUpgradeInfo("Datacenter", message)+"\n\n"

upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Datacenter(
message), callback_data="CORP UPGRADE Datacenter"))
        if(corp.departments["Contacts"]<10):
            msg_text += corp.departmentUpgradeInfo("Contacts", message)+"\n\n"

upgrade_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.corp.Contacts(me
ssage), callback_data="CORP UPGRADE Contacts"))

        bot.edit_message_text(msg_text, message.chat.id, callback.message.message_id,
reply_markup=upgrade_keyboard)
    elif(request[1]=="CONVERT"):
        convert_keyboard = telebot.types.InlineKeyboardMarkup()
```

```
convert_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.corp_convert_money(message), callback_data="CORP CONVERT MONEY"))

convert_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.corp_convert_influence(message), callback_data="CORP CONVERT INFLUENCE"))

convert_keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.cancel(message), callback_data="CORP MAIN"))
        bot.edit_message_text(lang.logic.corp_convert_recources_message(message, corp), message.chat.id, callback.message.message_id, reply_markup=convert_keyboard)
    elif(request[1]=="PRODUCTION"):
        pass

  @staticmethod
  def corpCommand(message):
    corpRequest = message.text.split()
    if(corpRequest[1] == 'join'):
      if(not CorpsTable.contains(Query().name == corpRequest[2])):
        bot.reply_to(message, lang.logic.corp_wrong_name(message))
        return
      player = Player(message.from_user.id)
      if(player.corp != "None"): #
        bot.reply_to(message, lang.logic.you_already_in_corp(message))
        return
      keyboard = telebot.types.InlineKeyboardMarkup()
      keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.yes(message), callback_data="CORP JOIN "+str(corpRequest[2])))
```

```
keyboard.add(telebot.types.InlineKeyboardButton(text=lang.logic.cancel(message),
callback_data="CORP JOIN CANCEL"))
        bot.reply_to(message,
lang.logic.are_you_sure_you_want_to_join_this_corp(message),
reply_markup=keyboard)
    elif(corpRequest[1] == 'create'): #Створення своєї корпорації
        if(not CorpsTable.contains(Query().name == corpRequest[2])):
            player = Player(message.from_user.id)
            if(player.corp != "None"): #якщо вже є в корпорації
                bot.reply_to(message, lang.logic.you_already_in_corp(message))
            elif(player.stats["money"]<10000): #якщо недостатньо коштів
                bot.reply_to(message, lang.logic.not_enough_money(message))
            else:
                corp = Corp(' '.join(corpRequest[2:]), player.ID)
                corp.save()
                player.corp = corp.name
                player.save()
                bot.reply_to(message, lang.logic.corp_created(message, corp))
        else: #Якщо така корпорація вже існує
            bot.reply_to(message, lang.logic.such_corp_already_exist(message))
    elif(corpRequest[1] == "invest"): #інвестування грошей у корпорацію
        try:
            amount = int(corpRequest[2])
            if(amount<=0):
                bot.reply_to(message, lang.logic.you_cant_invest_negitive(message))
                return
            player=Player(message.from_user.id)
            if(player.corp == "None"):
                bot.reply_to(message, lang.logic.you_are_not_in_corporation(message))
```

```
                return
            corp = Corp(player.corp)
            player.moneyChange(-amount)
            corp.budgetChange(amount)
            bot.reply_to(message, corpRequest[2] +
lang.logic.some_credits_invested(message))
            corp.save()
            player.save()
        except Exception as E:
            if(E == "not_enough_budget"):
                bot.reply_to(message, lang.corp.not_enough_budget(message))
            elif(E == "not_enough_influence"):
                bot.reply_to(message, lang.corp.not_enough_influence(message))
            elif(E == "not_enough_money"):
                bot.reply_to(message, lang.logic.not_enough_money(message))
```

Player.py:

```
from Common import PlayersTable
from tinydb import Query
from Localisations import lang

class Player:
    ID: int
    city = "None"
    corp = "None"
    equipment = {
        "head": "None",
        "body": "None",
        "weapon1": "None",
        "weapon2": "None",
```

```
    "other1": "None",
    "other2": "None"}
inventory = []
augmentations = []
psiPowers = []
genes = []
effects = []
stats = {
    "health": 100,
    "mental": 100,
    "stamina": 100,
    "xp": 25,
    "xp_total_spend": 0, #####
    "heat": 0,
    "money": 10000, #####
    "attributes": {
        "STR": 1,
        "END": 1,
        "AGI": 1,
        "INT": 1,
        "PER": 1,
        "PSI": 0
    },
    "skills": {
        "HACK": 0,
        "BIOTECH": 0,
        "ELECTROTECH": 0,
        "NEGOTIATION": 0,
        "RANGED": 0,
        "MELEE": 0,
```

```
            "STEALTH": 0,
            "SURVIVAL": 0
        }}
    state = ["None"]
    cooldowns = {
        "work": False,
        "rest": False
    }
    lang = "ua"
        #Добавить статистику

    def __init__(self, ID: int):
        self.ID = ID
        player = PlayersTable.search(Query().ID == self.ID)
        if(player != []):
            player=player[0]
            self.city = player["city"]
            self.corp = player["corp"]
            self.equipment = player["equipment"]
            self.inventory = player["inventory"]
            self.augmentations = player["augmentations"]
            self.psiPowers = player["psiPowers"]
            self.effects = player["effects"]
            self.stats = player["stats"]
            self.state = player["state"]
            self.cooldowns = player["cooldowns"]
            self.lang = player["lang"]
    #with open("DB/Players/"+str(self.ID)+".json", "w", #encoding='utf8'#) as f:
##############
    def save(self):
```

```
        p = {"ID": self.ID,
            "city": self.city,
            "corp": self.corp,
            "equipment": self.equipment,
            "inventory": self.inventory,
            "augmentations": self.augmentations,
            "psiPowers": self.psiPowers,
            "genes": self.genes,
            "effects": self.effects,
            "stats": self.stats,
            "state": self.state,
            "cooldowns": self.cooldowns,
            "lang": self.lang}
        if(not PlayersTable.contains(Query().ID == self.ID)):
            PlayersTable.insert(p)
        else:
            PlayersTable.update(p, Query().ID == self.ID)
            #"state": {
            #    "chat": self.state.chat,
            #    "private": self.state.private
            #}}},  ensure_ascii=False, indent=2))


    def XP_cost(self, stat: str):
        match stat:
            case "STR":
                return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["STR"]-
9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
            case "END":
```

```
            return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["END"]-
9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
        case "AGI":
            return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["AGI"]-
9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
        case "INT":
            return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["INT"]-
9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
        case "PER":
            return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["PER"]-
9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
        case "PSI":
            return float(format(32/(1+2.71828**(-0.5*(self.stats["attributes"]["PSI"]-
9.5)))+1+self.stats["xp_total_spend"]*0.025, '.1f'))
        case "HACK":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["HACK"]-
9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "BIOTECH":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["BIOTECH"]-
9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "ELECTROTECH":
            return float(format(16/(1+2.71828**(-
0.5*(self.stats["skills"]["ELECTROTECH"]-9.5)))+1+self.stats["xp_total_spend"]
'.1f'))
        case "NEGOTIATION":
            return float(format(16/(1+2.71828**(-
0.5*(self.stats["skills"]["NEGOTIATION"]-9.5)))+1+self.stats["xp_total_spend"]*0.02,
'.1f'))
        case "RANGED":
```

```
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["RANGED"]-
9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "MELEE":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["MELEE"]-
9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "STEALTH":
            return float(format(16/(1+2.71828**(-0.5*(self.stats["skills"]["STEALTH"]-
9.5)))+1+self.stats["xp_total_spend"]*0.02, '.1f'))
        case "SURVIVAL":
            return float(format(16/(1+2.71828**(-
0.5*(self.stats["skills"]["SURVIVAL"]-9.5)))+1+self.stats["xp_total_spend"]*0.02,
'.1f'))

    def statUP(self, stat: str):
        if(stat in self.stats["attributes"].keys()):
            if(self.stats["attributes"][stat]+1 > 16):
                raise Exception("Предел человеческих возможностей")
            if(self.stats["xp"] < self.XP_cost(stat)):
                raise Exception("Недостаточно опыта")
            self.stats["xp_total_spend"] = float(format(self.stats["xp_total_spend"] +
self.XP_cost(stat), '.1f'))
            self.stats["xp"] = float(format(self.stats["xp"] - self.XP_cost(stat), '.1f'))
            self.stats["attributes"][stat] += 1
        else:
            if(self.stats["skills"][stat]+1 > 16):
                raise Exception("Предел человеческих возможностей")
            if(self.stats["xp"] < self.XP_cost(stat)):
                raise Exception("Недостаточно опыта")
            self.stats["xp_total_spend"] = float(format(self.stats["xp_total_spend"] +
self.XP_cost(stat), '.1f'))
```

```
        self.stats["xp"] = float(format(self.stats["xp"] - self.XP_cost(stat), '.1f'))
        self.stats["skills"][stat] += 1

    def healthChange(self, amount): ###
        if(amount>0):
            if(self.stats["health"] + amount < 100):
                self.stats["health"] += amount
            else:
                self.stats["health"] = 100
            return True
        else:
            if(self.stats["health"] + amount > 0):
                self.stats["health"] -= amount/100 * (100 - self.stats["attributes"]["END"]*2)
                return True
            else:
                return False

    def staminaChange(self, amount):
        if(amount>0):
            if(self.stats["stamina"] + amount < 100):
                self.stats["stamina"] += amount
            else:
                self.stats["stamina"] = 100
            return True
        else:
            if(self.stats["stamina"] + amount > 0):
                self.stats["stamina"] -= amount/100 * (100 - self.stats["attributes"]["END"]*2)
                return True
            else:
                return False
```

```
def xpGain(self, amount):
    new_xp = float(format(amount * (1 + self.stats["attributes"]["INT"]*0.05), '.1f'))
    self.stats["xp"] += new_xp
    return new_xp

@property
def STR_base(self):
    return self.stats["attributes"]["STR"]
@property
def STR(self):
    return self.stats["attributes"]["STR"] #####

@property
def END_base(self):
    return self.stats["attributes"]["END"]
@property
def END(self):
    return self.stats["attributes"]["END"] #####

@property
def AGI_base(self):
    return self.stats["attributes"]["AGI"]
@property
def AGI(self):
    return self.stats["attributes"]["AGI"] #####

@property
def INT_base(self):
    return self.stats["attributes"]["INT"]
```

```
@property
def INT(self):
    return self.stats["attributes"]["INT"] #####

@property
def PER_base(self):
    return self.stats["attributes"]["PER"]
@property
def PER(self):
    return self.stats["attributes"]["PER"] #####

@property
def PSI_base(self):
    return self.stats["attributes"]["PSI"]
@property
def PSI(self):
    return self.stats["attributes"]["PSI"] #####

@property
def HACK_base(self):
    return self.stats["skills"]["HACK"]
@property
def HACK(self):
    return self.stats["skills"]["HACK"] #####

@property
def BIOTECH_base(self):
    return self.stats["skills"]["BIOTECH"]
@property
def BIOTECH(self):
```

```
    return self.stats["skills"]["BIOTECH"] #####

@property
def ELECTROTECH_base(self):
    return self.stats["skills"]["ELECTROTECH"]
@property
def ELECTROTECH(self):
    return self.stats["skills"]["ELECTROTECH"] #####

@property
def NEGOTIATION_base(self):
    return self.stats["skills"]["NEGOTIATION"]
@property
def NEGOTIATION(self):
    return self.stats["skills"]["NEGOTIATION"] #####

@property
def RANGED_base(self):
    return self.stats["skills"]["RANGED"]
@property
def RANGED(self):
    return self.stats["skills"]["RANGED"] #####

@property
def MELEE_base(self):
    return self.stats["skills"]["MELEE"]
@property
def MELEE(self):
    return self.stats["skills"]["MELEE"] #####
```

```
    @property
    def STEALTH_base(self):
        return self.stats["skills"]["STEALTH"]
    @property
    def STEALTH(self):
        return self.stats["skills"]["STEALTH"] #####

    @property
    def SURVIVAL_base(self):
        return self.stats["skills"]["SURVIVAL"]
    @property
    def SURVIVAL(self):
        return self.stats["skills"]["SURVIVAL"] #####

    @property
    def xp(self):
        return self.stats["xp"]



    def moneyChange(self, amount: int):
        if(self.stats["money"]+amount > 0):
            self.stats["money"] += amount
        else:
            raise Exception("not_enough_money")

    def work_cooldown(self):
        self.cooldowns["work"] = True
```

Corp.py:

```
from Common import PlayersTable, CorpsTable
```

```
from tinydb import Query
from Localisations import lang

class Corp:
    name: str
    owner: int
    departments = {
        "R&D": 0,
        "HR": 0,
        "Security Service": 0,
        "Chemical Labs": 0,
        "Sales department": 0,
        "Factories": 0,
        "Datacenter": 0,
        "Contacts": 0
    }
    resources = {
        "Budget": 0,
        "Influence": 0
    }

    def __init__(self, name: str, owner = None):
        self.name = name
        if(owner!=None):
            self.owner = owner
        else:
            corp = CorpsTable.search(Query().name == self.name)[0]
            self.departments = corp["departments"]
            self.resources = corp["resources"]
```

```
def save(self):
    c = {"name": self.name,
         "departments": self.departments,
         "resources": self.resources}
    if(not CorpsTable.contains(Query().name == self.name)):
        CorpsTable.insert(c)
    else:
        CorpsTable.update(c, Query().name == self.name)

def getEmployees(self):
    return PlayersTable.search(Query().name == self.name)

def budgetChange(self, amount):
    if(self.resources["Budget"]+amount > 0):
        self.resources["Budget"] += amount
    else:
        raise Exception("not_enough_budget")
def influenceChange(self, amount):
    if(self.resources["Influence"]+amount > 0):
        self.resources["Influence"] += amount
    else:
        raise Exception("not_enough_influence")

def departmentUP(self, department: str):
    if(department != "R&D"):
        if(self.departments[department]+1>=5 and self.departments["R&D"]+1<4):
            raise Exception("R&D_level_is_too_low")
        if(self.departments[department]+1>=7 and self.departments["R&D"]+1<7):
            raise Exception("R&D_level_is_too_low")
        if(self.departments[department]+1>=8 and self.departments["R&D"]+1<8):
```

```
            raise Exception("R&D_level_is_too_low")
        if(self.departments[department]+1>=9 and self.departments["R&D"]+1<9):
            raise Exception("R&D_level_is_too_low")
        if(self.departments[department]+1>=10 and self.departments["R&D"]+1<10):
            raise Exception("R&D_level_is_too_low")
    self.budgetChange(-
self.departmentsPriceList[department][self.departments[department]+1]["price"]["Budg
et"])#
    self.influenceChange(-
self.departmentsPriceList[department][self.departments[department]+1]["price"]["Influ
ence"])
    self.departments[department] += 1

departmentsPriceList = {
    "R&D": {
        1: {"price": {"Budget": 2500, "Influence": 0},
            "description": "Открывает заводы."},
        2: {"price": {"Budget": 3000, "Influence": 0},
            "description": "Открывает датацентр."},
        3: {"price": {"Budget": 5000, "Influence": 0},
            "description": "Открывает химические лаборатории."},
        4: {"price": {"Budget": 7500, "Influence": 10},
            "description": "Повышает максимальный уровень улучшений до 5."}
        5: {"price": {"Budget": 10000, "Influence": 20},
            "description": "Открывает доступ к особым биомодам."},
        6: {"price": {"Budget": 15000, "Influence": 50},
            "description": "Открывает доступ к особым аугментациям."},
        7: {"price": {"Budget": 20000, "Influence": 75},
            "description": "Повышает максимальный уровень улучшений до 7."},
        8: {"price": {"Budget": 25000, "Influence": 100},
```

```
            "description": "Повышает максимальный уровень улучшений до 8."},
        9: {"price": {"Budget": 50000, "Influence": 150},
            "description": "Повышает максимальный уровень улучшений до 9."},
        10: {"price": {"Budget": 75000, "Influence": 250},
            "description": "Повышает максимальный уровень улучшений до 10."}
    },
    "HR": {
        1: {"price": {"Budget": 1000, "Influence": 0},
            "description": "Повышает максимальное количество сотрудников до 5 (сейчас 3)."},
        2: {"price": {"Budget": 1500, "Influence": 0},
            "description": "Повышает максимальное количество сотрудников до 7 (сейчас 5)."},
        3: {"price": {"Budget": 2000, "Influence": 0},
            "description": "Повышает максимальное количество сотрудников до 10 (сейчас 7)."},
        4: {"price": {"Budget": 2500, "Influence": 0},
            "description": "Повышает максимальное количество сотрудников до 12 (сейчас 10)."},
        5: {"price": {"Budget": 3000, "Influence": 10},
            "description": "Повышает максимальное количество сотрудников до 15 (сейчас 12)."},
        6: {"price": {"Budget": 4000, "Influence": 20},
            "description": "Повышает максимальное количество сотрудников до 20 (сейчас 15)."},
        7: {"price": {"Budget": 5000, "Influence": 50},
            "description": "Повышает максимальное количество сотрудников до 30 (сейчас 20)."},
        8: {"price": {"Budget": 7000, "Influence": 100},
```

"description": "Повышает максимальное количество сотрудников до 50 (сейчас 30)."},
9: {"price": {"Budget": 10000, "Influence": 200},
"description": "Повышает максимальное количество сотрудников до 75 (сейчас 50)."},
10: {"price": {"Budget": 15000, "Influence": 500},
"description": "Повышает максимальное количество сотрудников до 100 (сейчас 75)."}
},
"Security Service": {
1: {"price": {"Budget": 1000, "Influence": 0},
"description": "Степень защиты от нападений +10."},
2: {"price": {"Budget": 1500, "Influence": 0},
"description": "Степень защиты от нападений +20."},
3: {"price": {"Budget": 2500, "Influence": 0},
"description": "Степень защиты от нападений +30."},
4: {"price": {"Budget": 3000, "Influence": 0},
"description": "Степень защиты от нападений +40."},
5: {"price": {"Budget": 4000, "Influence": 10},
"description": "Степень защиты от нападений +50."},
6: {"price": {"Budget": 5000, "Influence": 20},
"description": "Степень защиты от нападений +60."},
7: {"price": {"Budget": 7500, "Influence": 50},
"description": "Степень защиты от нападений +70."},
8: {"price": {"Budget": 10000, "Influence": 100},
"description": "Степень защиты от нападений +80."},
9: {"price": {"Budget": 15000, "Influence": 200},
"description": "Степень защиты от нападений +90."},
10: {"price": {"Budget": 25000, "Influence": 400},
"description": "Степень защиты от нападений +100."}

```
    },
    "Chemical Labs": {
        1: {"price": {"Budget": 0, "Influence": 0},
            "description": "Производит 1 энергетический батончик в день для
каждого сотрудника."},
        2: {"price": {"Budget": 2000, "Influence": 0},
            "description": "Открывает аптечки."},
        3: {"price": {"Budget": 2500, "Influence": 0},
            "description": "Производит 2 батончика в день."},
        4: {"price": {"Budget": 3000, "Influence": 0},
            "description": "Аптечки стоят дешевле."},
        5: {"price": {"Budget": 4000, "Influence": 10},
            "description": "Открывает тоники."},
        6: {"price": {"Budget": 5000, "Influence": 20},
            "description": "Заменяет тоники до стимуляторами."},
        7: {"price": {"Budget": 7500, "Influence": 50},
            "description": "Открывает адреналин."},
        8: {"price": {"Budget": 10000, "Influence": 100},
            "description": "Открывает психостимуляторы."},
        9: {"price": {"Budget": 15000, "Influence": 200},
            "description": "Улучшает аптечки до медкомлпектов."},
        10: {"price": {"Budget": 25000, "Influence": 500},
             "description": "Все стимуляторы стоят дешевле."}
    },
    "Sales department": {
        1: {"price": {"Budget": 1000, "Influence": 0},
            "description": "Повышает эффективность работы сотрудников на 10%."},
        2: {"price": {"Budget": 1500, "Influence": 0},
            "description": "Повышает эффективность работы сотрудников на 20%
(сейчас 10%)."},
```

3: {"price": {"Budget": 2000, "Influence": 0},
"description": "Повышает эффективность работы сотрудников на 30% (сейчас 20%)."},
4: {"price": {"Budget": 2500, "Influence": 0},
"description": "Повышает эффективность работы сотрудников на 40% (сейчас 30%)."},
5: {"price": {"Budget": 3000, "Influence": 10},
"description": "Повышает эффективность работы сотрудников на 50% (сейчас 40%)."},
6: {"price": {"Budget": 4000, "Influence": 20},
"description": "Повышает эффективность работы сотрудников на 60% (сейчас 50%)."},
7: {"price": {"Budget": 5000, "Influence": 50},
"description": "Повышает эффективность работы сотрудников на 70% (сейчас 60%)."},
8: {"price": {"Budget": 7000, "Influence": 100},
"description": "Повышает эффективность работы сотрудников на 80% (сейчас 70%)."},
9: {"price": {"Budget": 10000, "Influence": 200},
"description": "Повышает эффективность работы сотрудников на 90% (сейчас 80%)."},
10: {"price": {"Budget": 15000, "Influence": 500},
"description": "Повышает эффективность работы сотрудников на 10 (сейчас 90%)."}
},
"Factories": {
1: {"price": {"Budget": 0, "Influence": 0},
"description": "Производит 10 бюджета ежедневно."},
2: {"price": {"Budget": 1500, "Influence": 0},
"description": "Производит 20 бюджета ежедневно."},

```
        3: {"price": {"Budget": 2500, "Influence": 0},
           "description": "Производит 50 бюджета ежедневно."},
        4: {"price": {"Budget": 3000, "Influence": 0},
           "description": "Открывает доступ к производству своего снаряжения."},
        5: {"price": {"Budget": 5000, "Influence": 10},
           "description": "Производит 75 бюджета ежедневно."},
        6: {"price": {"Budget": 7500, "Influence": 20},
           "description": "Открывает доступ к производству оружия."},
        7: {"price": {"Budget": 10000, "Influence": 50},
           "description": "Производит 100 бюджета ежедневно."},
        8: {"price": {"Budget": 15000, "Influence": 100},
           "description": "Всё производимое заводом и R&D стоит дешевле."},
        9: {"price": {"Budget": 25000, "Influence": 200},
           "description": "Производит 200 бюджета ежедневно."},
        10: {"price": {"Budget": 50000, "Influence": 500},
            "description": "Производит 250 бюджета ежедневно."}
    },
    "Datacenter": {
        1: {"price": {"Budget": 0, "Influence": 0},
           "description": "Увеличивает защиту от хакеров и силу своих хакеров на
5%."},
        2: {"price": {"Budget": 1500, "Influence": 0},
           "description": "Увеличивает защиту от хакеров и силу своих хакер
10%."},
        3: {"price": {"Budget": 2500, "Influence": 0},
           "description": "Увеличивает защиту от хакеров и силу своих хакеров на
15%."},
        4: {"price": {"Budget": 3000, "Influence": 0},
           "description": "Увеличивает защиту от хакеров и силу своих хакеров на
20%."},
```

5: {"price": {"Budget": 5000, "Influence": 10},
"description": "Увеличивает защиту от хакеров и силу своих хакеров на 25%."},
6: {"price": {"Budget": 7500, "Influence": 20},
"description": "Увеличивает защиту от хакеров и силу своих хакеров на 30%."},
7: {"price": {"Budget": 10000, "Influence": 50},
"description": "Увеличивает защиту от хакеров и силу своих хакеров на 35%."},
8: {"price": {"Budget": 15000, "Influence": 100},
"description": "Увеличивает защиту от хакеров и силу своих хакеров на 40%."},
9: {"price": {"Budget": 20000, "Influence": 150},
"description": "Увеличивает защиту от хакеров и силу своих хакеров на 45%."},
10: {"price": {"Budget": 40000, "Influence": 200},
"description": "Увеличивает защиту от хакеров и силу своих хакеров на 50%."}
},
"Contacts": {
1: {"price": {"Budget": 2000, "Influence": 0},
"description": "Производит 1 влияния ежедневно."},
2: {"price": {"Budget": 3000, "Influence": 0},
"description": "Производит 2 влияния ежедневно. Снижает heat сотрудников на 1 в день."},
3: {"price": {"Budget": 5000, "Influence": 10},
"description": "Производит 3 влияния ежедневно. Снижает heat сотрудников на 2 в день."},
4: {"price": {"Budget": 7500, "Influence": 25},

```
            "description": "Производит 4 влияния ежедневно. Снижает heat
сотрудников на 3 в день."},
        5: {"price": {"Budget": 10000, "Influence": 50},
            "description": "Производит 5 влияния ежедневно. Снижает heat
сотрудников на 5 в день."},
        6: {"price": {"Budget": 15000, "Influence": 75},
            "description": "Производит 6 влияния ежедневно. Снижает heat
сотрудников на 7 в день."},
        7: {"price": {"Budget": 20000, "Influence": 100},
            "description": "Производит 7 влияния ежедневно. Снижает heat
сотрудников на 10 в день."},
        8: {"price": {"Budget": 30000, "Influence": 250},
            "description": "Производит 8 влияния ежедневно. Снижает heat
сотрудников на 15 в день."},
        9: {"price": {"Budget": 50000, "Influence": 500},
            "description": "Производит 9 влияния ежедневно. Снижает heat
сотрудников на 20 в день."},
        10: {"price": {"Budget": 75000, "Influence": 750},
             "description": "Производит 10 влияния ежедневно. Снижает heat
сотрудников на 25 в день."}
    }

}

def departmentUpgradeInfo(self, dep, message):
    if(self.departments[dep]<=10):
        return lang.corp.departmentUpgradeInfoText(message, self, dep)
    else:
        return ""
```

Chat_City.py:

```
import telebot
from Common import bot, ChatCitiesTable, PlayersTable
from tinydb import Query
from Locations import Locations
import random

class ChatCity:
    chatID: int
    locations = []
    corps = []
    lang = "ua"



    def generateLocations(self):
        size=bot.get_chat_member_count(self.chatID)
        locs=Locations.copy()
        if(size>50): #1-49
            while len(self.locations)<4:
                loc=locs.pop(list(locs.keys())[((random.randint(0, len(locs)-1)))])
                if(loc.get(1) != None):
                    self.locations.append(loc.get(1))
        elif(size<150): #50-149
            while len(self.locations)<6:
                loc=locs.pop(list(locs.keys())[((random.randint(0, len(locs)-1) ))])
                if(loc.get(2) != None):
                    self.locations.append(loc.get(2))
                else:
                    if(loc.get(1) != None):
                        self.locations.append(loc.get(1))
```

```
    else: #150+
        while len(self.locations)<8:
            loc=locs.pop(list(locs.keys())[((random.randint(0, len(locs)-1)))])
            if(loc.get(3) != None):
                self.locations.append(loc.get(3))
            else:
                if(loc.get(2) != None):
                    self.locations.append(loc.get(2))
                else:
                    if(loc.get(1) != None):
                        self.locations.append(loc.get(1))



def __init__(self, chatID):
    self.chatID = chatID
    chat = ChatCitiesTable.search(Query().ID == self.chatID)
    if(chat != []):
        chat=chat[0][str(chatID)]
        self.locations = chat["locations"]
        self.corps = chat["corps"]
        self.lang = chat["lang"]
    else:
        self.generateLocations()

def save(self):
    c = {"chatID": self.chatID,
        "locations": self.locations,
        "corps": self.corps,
        "lang": self.lang}
    if(not ChatCitiesTable.contains(Query().chatID == self.chatID)):
```

```
            ChatCitiesTable.insert(c)
        else:
            ChatCitiesTable.update(c, Query().chatID == self.chatID)

    def getPlayers(self):
        return PlayersTable.search(Query().city == self.chatID)
```

Locations.py:

```
Locations = {
    "Трущоби": {
        1: "Трущоби",
        3: "Нижнє місто"
    },
    "Промсектор":{
        1: "Заброшений промсектор"
    },
    "Пустоши":{
        1: "Пустоши"
    },
    "Ринок":{
        1: "Ринок",
        2: "Полузаброшений ТЦ",
        3: "Мегаплекс"
    },
    "Офіси":{
        1: "Діловий квартал",
        2: "Корпоративні офиси",
        3: "Корпоративний епіцентр"
    },
    "Жиле":{
```

```
        1: "Жилий район",
        3: "Вулик"
    },
    "Канализації":{
        1: "Канализації"
    },
    "Арена":{
        1: "Підпольна арена"
    }
}
```

Common.py:

```
import telebot
from tinydb import TinyDB

bot = telebot.TeleBot('тут має знаходитись секретний токен')

db = TinyDB('db.json')
PlayersTable = db.table("Players")
ChatCitiesTable = db.table("ChatCities")
CorpsTable = db.table("Corps")
OtherTable = db.table("Other")

dev_id = 345694869
```