Пермский филиал федерального государственного автономного образовательного учреждения высшего образования

«Национальный исследовательский университет

«Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Образовательная программа бакалавриата «Программная инженерия»

**ОТЧЕТ**

**по проектной работе**

Выполнил студент группы ПИ-20-2

Перминов Николай Александрович

(подпись)

Руководитель проекта

Старший преподаватель

информационных технологий в бизнесе

Вячеслав Владимирович Ланин

\_\_\_\_\_\_\_\_\_\_\_

(оценка) (подпись)

(дата)

Пермь, 2021

Оглавление

[1 Постановка задачи 6](#_Toc75894310)

[1.1 Общая постановка задачи 6](#_Toc75894311)

[1.2 Личная постановка задачи 6](#_Toc75894312)

[2 Анализ аналогов 7](#_Toc75894313)

[3 Анализ проекта 9](#_Toc75894314)

[3.1 Анализ инструментов разработки 9](#_Toc75894315)

[3.2 Анализ реализации 9](#_Toc75894316)

[3.3 Анализ развертки бота 10](#_Toc75894317)

[4 Проектирование 11](#_Toc75894318)

[4.1 Класс Processing 11](#_Toc75894319)

[4.2 Класс Logging 11](#_Toc75894320)

[4.3 Общая диаграмма классов 11](#_Toc75894321)

[5 Код программы 14](#_Toc75894322)

[5.1 Первоначальный парсинг городов 14](#_Toc75894323)

[5.2 Разработка смещенного распределения Гауса 17](#_Toc75894324)

[5.3 Парсинг основной части данных 17](#_Toc75894325)

[5.4 Класс Processing 25](#_Toc75894326)

[5.5 Класс Logging 35](#_Toc75894327)

[6 Тестирование 37](#_Toc75894328)

[7 Заключение 38](#_Toc75894329)

[8 Список литературы 39](#_Toc75894330)

## Введение

Тема курсового проекта: «Разработка чат-бота для игры в города на языке C#» отчет курсового проекта выполнен на 38 страницах, содержит 0 таблицы, 5 рисунок, 12 использованных источников. Необходимость выполнения настоящего проекта обусловлена потребностью пользователей получения развлекательного и образовательного контента, путем игры в Города, в мессенджере Telegram, а также отсутствием приложений, которые бы предоставляли возможность игры в города без скачивания программы.

Цель работы: целью проекта является изучение особенностей конструирования и применения чат-ботов, разработка чат-бота для игры в города в мессенджере Telegram. Для выполнения поставленной цели необходимо решение следующих задач:

− изучить понятия чат-бота и его функций;

− рассмотреть преимущества использования мессенджеров для игры в Города;

− рассмотреть языки программирования и выбрать язык для разработки;

− выбрать мессенджер, в котором будет реализован чат-бот;

− разработать ИТ-проект по разработке чат-бота;

− продемонстрировать концепцию разработки.

Объектом исследования данной работы является программирование на языке C# для создания чат-бота, который позволит пользователям играть в Города прямо в мессенджере Telegram. Предметом данной работы является чат-бот, который позволит играть в Города прямо в мессенджере Telegream. Практическая значимость заключается в том, что данный чат-бот позволяет играть в города без скачивания каких-либо приложений и получать дополнительную информацию по данным городам.

## Аннотация

Курсовой проект посвящен разработке чат-бота, который позволит пользователю играть в Города, в мессенджере Telegram. Подробнее рассмотрена часть работы, которая выполнена автором отчета. Отчет состоит из семи частей. В первой части будет рассмотрено назначение разработки, во второй части будет произведена аналитика целей проекта, в третьей части будет проведен анализ аналогов и выявление преимуществ нашего продукта, в четвертой части будет приведен код части программы, над которой работал данный студент, в пятой части приведено руководство пользователя, в шестой части отчет производится проектирование работы и взаимодействий программы, в седьмой части произведено тестирование части программы, которую разрабатывал студент на предмет ошибок или некорректных ответов на запросы пользователя. В заключительной части будет сделан вывод о том, какая работа была проделана, какие особенности получил продукт разработки, что вызвало наибольшее затруднение, а что, наоборот, было простым и быстро удалось реализовать.

# Постановка задачи

## Общая постановка задачи

Разрабатываемое в ходе реализации проекта программное обеспечение, должно включать следующие возможности:

* поддержка полноценной игры в Города (Города́ — игра для нескольких человек, в которой каждый участник в свою очередь называет реально существующий город любой страны, название которого начинается на ту букву, которой оканчивается название предыдущего участника);
* предоставление пользователю геолокации города;
* предоставление пользователю ссылки на справки из Википедии по городу;
* предоставление пользователю координат города на карте;
* предоставление пользователю ссылки на фотографии города;
* предоставление пользователю ссылки на Яндекс или Гугл с запросом по городу.

## Личная постановка задачи

В мои задачи входит:

* разработать базу данных городов;
* разработать функцию запроса города из данных;
* разработать базу данных пользователей;
* собрать базу данных;
* развертка бота на сервер.

# Анализ аналогов

Анализ рынка программных продуктов позволил выделить три основных программы, используемых для игры в Города. К ним относятся такие программные продукты как «Игра в города» - бот в мессенджере Telegram, «Сыграем в Города» - приложение из Google Play для Android, «Города- Игра от А до Я» приложение из App Store. Рассмотрим особенности каждой из программ и выделим ее сильные и слабые стороны.

«Сыграем в Города» является приложением, предназначенным для Android. Для игры данное приложение нужно скачать бесплатно из Google Play. К плюсам данного приложения можно отнести удобный интерфейс, возможность поменять тему, а также возможность получать достижения в Google Play. К минусам - встроенную рекламу, долгие ответы приложения (примерно 10–15 секунд) и невозможность узнать дополнительную информацию о городе (предоставляется только название города и флаг страны, в которой он находится).

«Города- Игра от А до Я» является приложением, предназначенным для iOS. Для игры данное приложение нужно скачать за $0.99 из App Store. В программе предусмотрены три режима: классический, на время и пятиминутка, в каждом из которых реализован ввод города с уже написанной первой буквой, что безусловно является плюсом. Также в приложении предусмотрен удобный набор очков, позволяющих пользователю взять подсказку. К недостаткам данного аналога можно отнести встроенную рекламу, отсутствие возможности играть с другими пользователями и отсутствие подсказок для игрока.

«Игра в города» является ботом в мессенджере Telegram. Для игры пользователю нужно иметь доступ в мессенджер Telegram. В данном боте эргономично расположены правила игры в Города, условия пользования ботом и его возможности, сделано это с помощью команд, которые пользователь может вызвать даже в момент игры. Недостатки данного бота заключаются в невозможности начать новую игру, кроме того, игра происходит одновременно со всеми участниками, подписавшимися на бота.  
Приведенный анализ показал, что все из рассмотренных программ имеют низкие требования к системным ресурсам, однако все из них требуют скачивания. Также во всех приведенных программах нет возможности получить дополнительную информацию о городе, что с положительной стороны выделяет наш продукт из всех аналогов.

# Анализ проекта

## Анализ инструментов разработки

Для реализации бота было принято решение использовать язык программирования C#, по причине наибольшего опыта разработки на данном языке у разработчиков.

Для получения (парсинга) данных используется язык программирования Python.

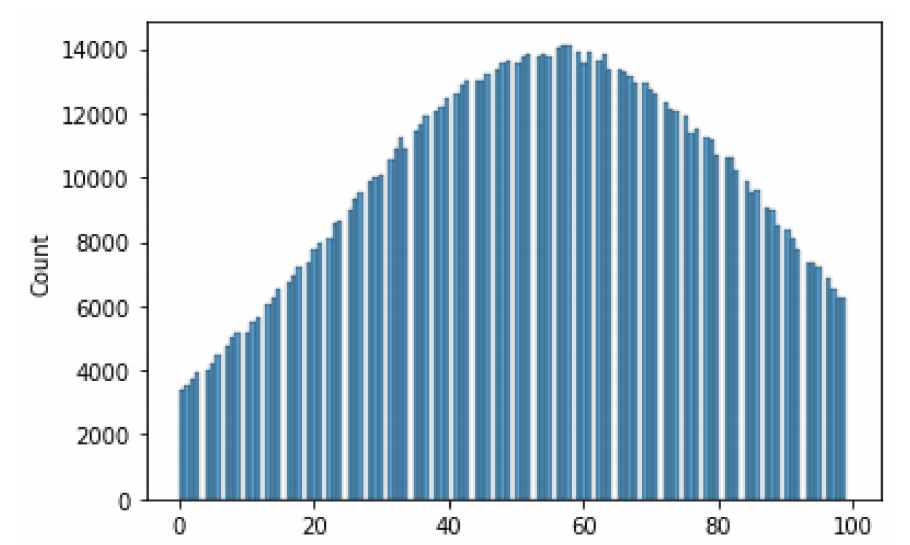
## Анализ реализации

Базу данных городов мне пришлось брать из открытой библиотеки Wikipedia, на ней были найдены данные из «РосРеестр».

Для парсинда этих данных использовались библиотеки openpyxl и pandas.

Дополнительные данные берутся из открытых источников OpenStreetMap, Wikipedia, Yandex и Google. Для этого используются библиотеки «pandas», «requests», «lxml», «geopy», «pgeocode», «wikipediaapi». Для ускорения парсинга используется распараллеливание по средствам библиотеки «multiprocessing.dummy».

Для выбора города из базы данных (рассортированной по населению города) используется смещенное распределение Гаусса. Ниже представлен пример распределения.



1. Пример распределения шанса «выпасть» городу

## Анализ развертки бота

Бот будет развертываться на виртуальной машине с ОС Ubuntu 20.04, 2 ядрами vCPU Intel Cascade Lake c гарантированной долей vCPU 5%, RAM 1 ГБ, объём дискового пространства 13 ГБ. Хостинг предоставляется сервисом «Yandex Cloud» на безвозмездной основе в рамках пробного периода.

# Проектирование

## Класс Processing

Отвечает за общение бота с базой данных.

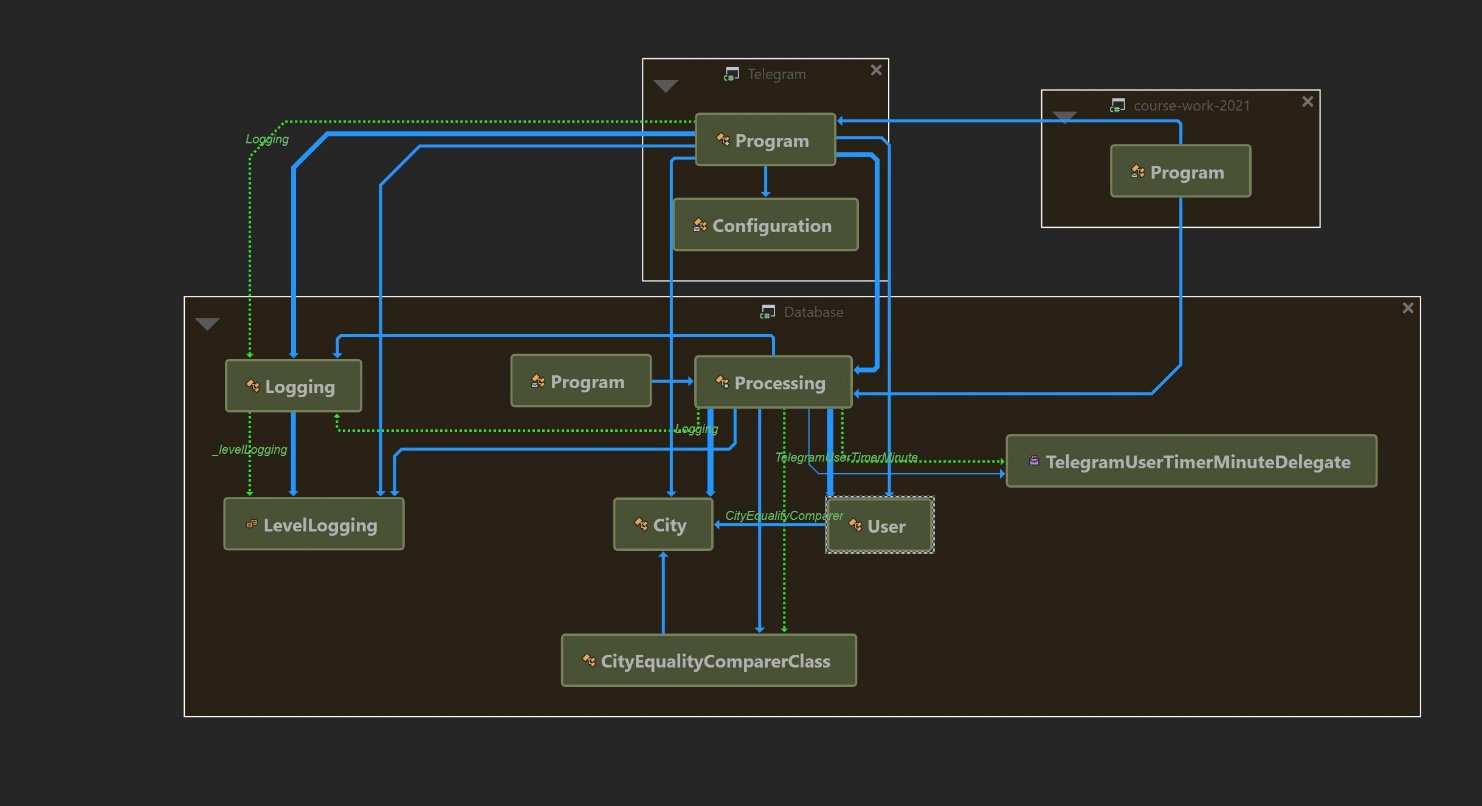
Бот связывается с базой данных по средствам функций:

* «Get» ­– получить ответ на город, заданный пользователем;
* «DeleteUser» – удалить пользователя из базы данных (начать игру заново);
* ReadCsv – первоначальная загрузка данных в программу;
* NextLetterUser – запрос следующей буквы для пользователя.

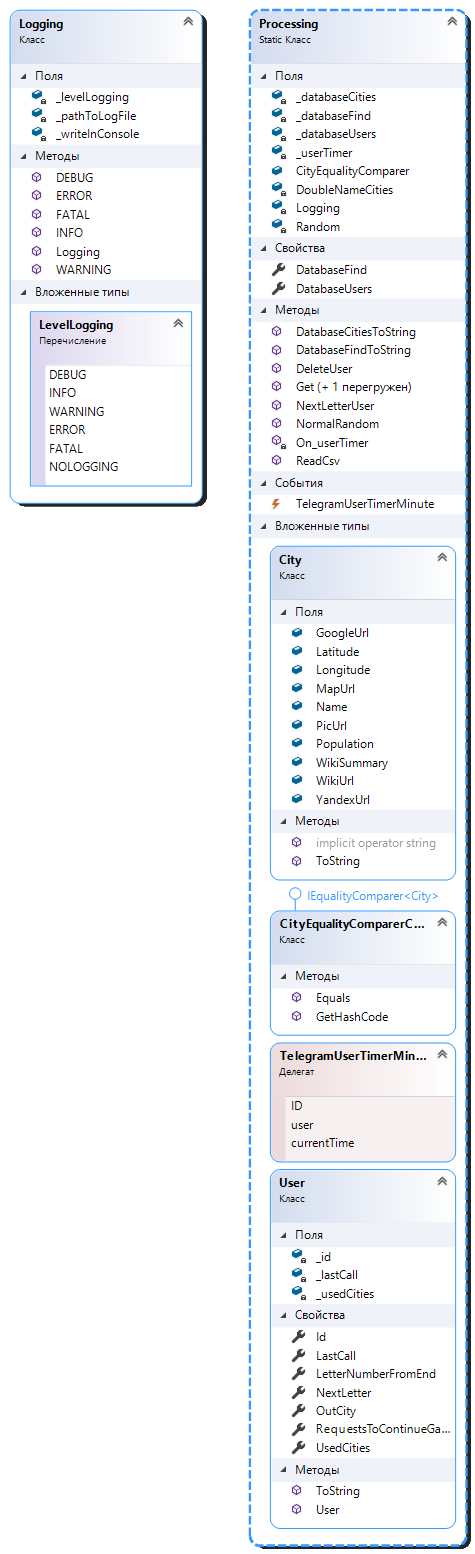
## Класс Logging

Класс логирования. Производит запись в заданные файлы, с разным уровнем ошибки. Аналог библиотеки «Logging» для python.

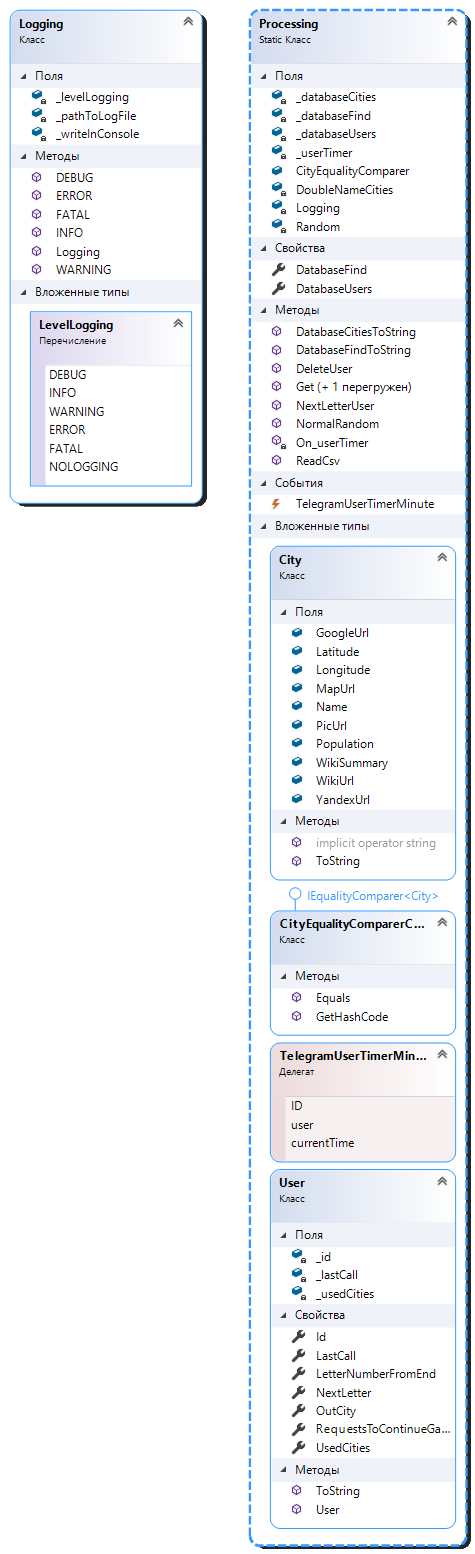
## Общая диаграмма классов



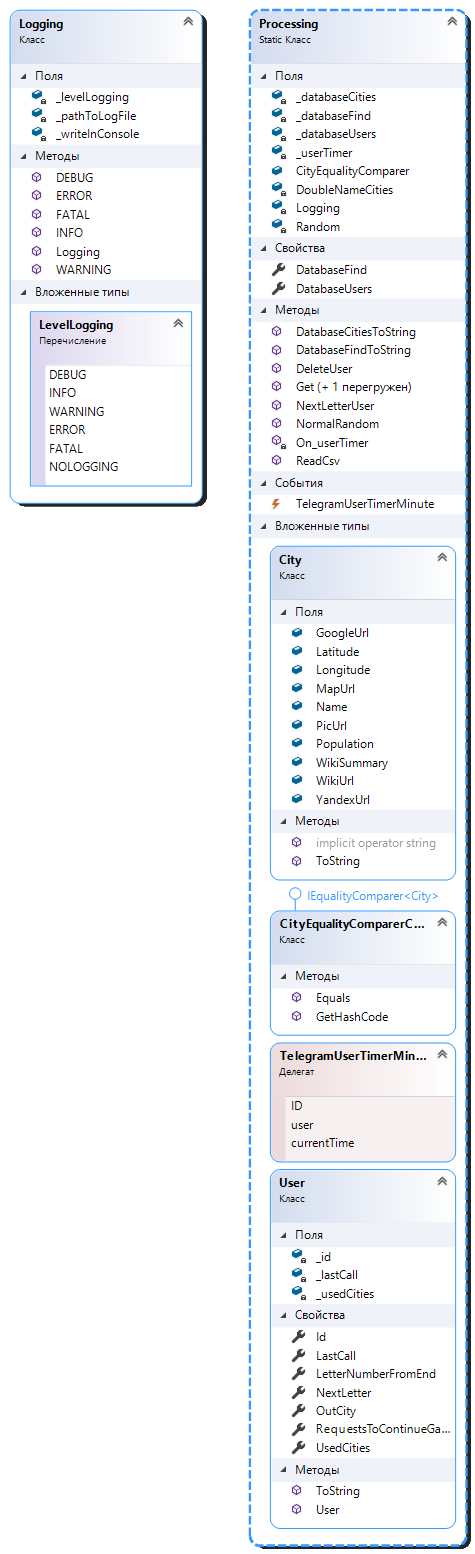
1. Диаграмма взаимодействия объектов классов



1. Диаграмма класса Logging



1. Диаграмма класса Processing (часть 1)



1. Диаграмма класса Processing (часть 2)

# Код программы

Полный текст программы удобнее смотреть на GitHub:   
<https://github.com/k0perX-X/curse-work-2021>.

## Первоначальный парсинг городов

#%%  
  
from openpyxl import load\_workbook  
import openpyxl  
import pandas as pd  
  
#%%  
  
wb = load\_workbook('Database/Russian\_cities.xlsx')  
  
#%%  
  
print(wb.get\_sheet\_names())  
  
#%%  
  
sheet = wb.get\_sheet\_by\_name('Численность по МО')  
sheet.title  
  
#%%  
  
cities = {}  
errors =[]  
for i in range(8, 23646):  
 x = str(sheet[f'B{i}'].value)  
 if 'г.' in x:  
 x = " ".join(x.replace('г.', '').split())  
 try:  
 if x in cities:  
 print(x, i, end=", ")  
 if int(sheet[f'C{i}'].value) > cities[x]:  
 cities[x] = int(sheet[f'C{i}'].value)  
 else:  
 cities[x] = int(sheet[f'C{i}'].value)  
 except:  
 if x in cities:  
 print(x, i, end=", ")  
 if int(sheet[f'D{i}'].value) > cities[x]:  
 cities[x] = int(sheet[f'D{i}'].value)  
 else:  
 cities[x] = int(sheet[f'D{i}'].value)  
print()  
print(cities)  
print(errors)  
  
#%%  
  
j = 0  
while j != len(cities):  
 j = 0  
 for i in cities:  
 if "Городское поселение" in i:  
 cities[" ".join(i.replace("Городское поселение", '').split())] = cities[i]  
 del cities[i]  
 print(i, end=", ")  
 break  
 if "Муниципальный округ" in i:  
 cities[" ".join(i.replace("Муниципальный округ", '').split())] = cities[i]  
 del cities[i]  
 print(i, end=", ")  
 break  
  
 if "Городской округ" in i:  
 cities[" ".join(i.replace("Городской округ", '').split())] = cities[i]  
 del cities[i]  
 print(i, end=", ")  
 break  
 if "- город федерального значения" in i:  
 cities[" ".join(i.replace("- город федерального значения", '').split())] = cities[i]  
 del cities[i]  
 print(i, end=", ")  
 break  
 j += 1  
print()  
j = 0  
cities['Нарьян-Мар'] = cities['Нарьян- Мар']  
del cities['Нарьян- Мар']  
cities['Каменск-Уральский'] = cities['Каменск - Уральский']  
del cities['Каменск - Уральский']  
while j != len(cities):  
 j = 0  
 for i in cities:  
 if "- " in i:  
 cities[" ".join(i.replace("- ", '').split())] = cities[i]  
 del cities[i]  
 print(i, end=", ")  
 break  
 if " -" in i:  
 cities[" ".join(i.replace(" -", '').split())] = cities[i]  
 del cities[i]  
 print(i, end=", ")  
 break  
 j += 1  
  
  
#%%  
  
from pprint import pprint  
pprint(cities)  
  
  
#%%  
  
letters = {}  
for i in range(32):  
 letters[chr(1072 + i)] = []  
print(letters)  
  
for i in cities:  
 letters[i[0].lower()].append([i, cities[i]])  
  
pprint(letters)  
  
print()  
  
for i in letters:  
 letters[i].sort(key=lambda x: x[1], reverse=True)  
  
pprint(letters)  
  
#%%  
  
cities = list(cities.keys())  
  
  
#%%  
  
print(cities)  
  
#%%  
  
cities = []  
for i in letters:  
 for j in letters[i]:  
 cities.append(j)  
pprint(cities)  
  
#%%  
  
df = pd.DataFrame(columns=['Letter', 'City', 'Population'])  
for i in letters:  
 for j in letters[i]:  
 df = df.append({  
 'Letter': i,  
 'City': j[0],  
 'Population': j[1]  
 }, ignore\_index=True)  
df  
  
#%%  
  
df.to\_csv('Database/Russian\_cities.csv')  
  
#%%

## Разработка смещенного распределения Гауса

#%%  
  
from random import random, gauss  
import math  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
#%%  
  
def randnormal1(mu = 0, sigma = 1, left = -1.7, right = 1.3):  
 u1 = 1.0 - random()  
 u2 = 1.0 - random()  
 rand = math.sqrt(-2.0 \* math.log(u1)) \* math.sin(2.0 \* math.pi \* u2)  
 while not (left <= rand < right):  
 u1 = 1.0 - random()  
 u2 = 1.0 - random()  
 rand = math.sqrt(-2.0 \* math.log(u1)) \* math.sin(2.0 \* math.pi \* u2)  
 rand = mu + sigma \* rand  
 return (rand - left) / abs(right - left)  
  
def randnormal(mu = 0, sigma = 1):  
 u1 = 1.0 - random()  
 u2 = 1.0 - random()  
 rand = math.sqrt(-2.0 \* math.log(u1)) \* math.sin(2.0 \* math.pi \* u2)  
 rand = mu + sigma \* rand  
 return rand  
  
#%%  
  
a = [randnormal1() for i in range(1000000)]  
print(min(a), max(a))  
count = 1  
a1 = [int(round((i - (1 / (count \* 2))) \* count)) for i in a]  
print(len(set(a1)), set(a1))  
sns.boxenplot(a1)  
  
#%%  
  
sns.histplot(a1)  
  
#%%  
  
a = [randnormal() for j in range(1000000)]  
sns.histplot(a)  
  
#%%  
  
b = [gauss(0, 1) for j in range(1000000)]  
sns.histplot(b)  
  
#%%

## Парсинг основной части данных

#%%  
  
import eventlet  
import pandas as pd  
from multiprocessing.dummy import Pool as ThreadPool  
import requests  
import lxml  
from lxml import etree  
from geopy.geocoders import Nominatim  
import geopy  
import pgeocode  
import wikipediaapi  
  
  
#%%  
  
df = pd.read\_csv('Database/Russian\_cities.csv', dtype=str)  
del df['Unnamed: 0']  
df = df.loc[df['City'] != 'Щелкино']  
df = df.reset\_index(drop=True)  
  
#%%  
  
df  
  
#%%  
  
a = []  
b = []  
c = []  
for i in df['City']:  
 a.append(f"https://www.google.com/search?q={i.replace(' ', '+')}")  
 b.append(f"https://yandex.ru/search/?text={i.replace(' ', '+')}")  
 c.append(f"https://ru.wikipedia.org/wiki/{i.replace(' ', '\_')}")  
  
a = pd.Series(a, name='GoogleUrl')  
b = pd.Series(b, name='YandexUrl')  
df = pd.concat([df, a], axis=1)  
df = pd.concat([df, b], axis=1)  
df  
  
#%%  
  
def checkurl(url):  
 tf = True  
 s = requests.get(url).text  
 page = lxml.etree.fromstring(s, parser=lxml.etree.HTMLParser())  
 for i in page.xpath('//\*[@id="noarticletext"]/p'):  
 s = ''.join(i.itertext())  
 if 'нет статьи' in s:  
 print(url + '\n', end='')  
 tf = False  
 if tf:  
 return url  
# pool = ThreadPool(20)  
# results = list(pool.map(checkurl, c))  
results = c  
# results = list(map(checkurl, c))  
  
# results  
  
#%%  
  
c = pd.Series(results, name='WikiUrl')  
df = pd.concat([df, c], axis=1)  
df  
# //\*[@class="infobox"]/tr[3]/td/span/span/a.href()  
# //\*[@id="mw-content-text"]/div[1]/table[1]/tr[3]/td/span/span/a.href()  
# //\*[@class="infobox-image"]/span/a  
# //\*[@class="wikidata-claim"]/span/a  
  
#%%  
  
df.loc[df['City'] == 'Алексеевка', 'WikiUrl'] = 'https://ru.wikipedia.org/wiki/Алексеевка\_(город,\_Белгородская\_область)'  
df.loc[df['City'] == 'Алексеевка', 'GoogleUrl'] = 'https://www.google.com/search?q=Город+Алексеевка+в+Белгородской+области'  
df.loc[df['City'] == 'Алексеевка', 'YandexUrl'] = 'https://yandex.ru/search/?text=Город+Алексеевка+в+Белгородской+области'  
  
df.loc[df['City'] == 'Буй', 'WikiUrl'] = 'https://ru.wikipedia.org/wiki/Буй\_(город)'  
df.loc[df['City'] == 'Буй', 'GoogleUrl'] = 'https://www.google.com/search?q=Буй+город'  
df.loc[df['City'] == 'Буй', 'YandexUrl'] = 'https://yandex.ru/search/?text=Буй+город'  
  
df.loc[df['City'] == 'Калач', 'WikiUrl'] = 'https://ru.wikipedia.org/wiki/Калач\_(город)'  
df.loc[df['City'] == 'Калач', 'GoogleUrl'] = 'https://www.google.com/search?q=Калач+город'  
df.loc[df['City'] == 'Калач', 'YandexUrl'] = 'https://yandex.ru/search/?text=Калач+город'  
  
df.loc[df['City'] == 'Донецк', 'WikiUrl'] = 'https://ru.wikipedia.org/wiki/Донецк\_(Россия)'  
df.loc[df['City'] == 'Донецк', 'GoogleUrl'] = 'https://www.google.com/search?q=Донецк+Россия'  
df.loc[df['City'] == 'Донецк', 'YandexUrl'] = 'https://yandex.ru/search/?text=Донецк+Россия'  
  
df.loc[df['City'] == 'Александровск', 'WikiUrl'] = 'https://ru.wikipedia.org/wiki/Александровск\_(Пермский\_край)'  
df.loc[df['City'] == 'Александровск', 'GoogleUrl'] = 'https://www.google.com/search?q=Александровск+Пермский+край'  
df.loc[df['City'] == 'Александровск', 'YandexUrl'] = 'https://yandex.ru/search/?text=Александровск+Пермский+край'  
  
#%%  
  
def checkurl(url):  
 tf = True  
 s = requests.get(url).text  
 page = lxml.etree.fromstring(s, parser=lxml.etree.HTMLParser())  
 if len(page.xpath('//\*[@class="infobox-image"]/span/a')) < 1:  
 if len(page.xpath('//\*[@class="wikidata-claim"]/span/a')) < 1:  
 if len(page.xpath('//\*[@class="infobox"]/tr[3]/td/span/span/a')) < 1:  
 # print(url + '\n', end='')  
 return url  
  
pool = ThreadPool(15)  
results = list(pool.map(checkurl, df['WikiUrl']))  
a = [i for i in results if i is not None]  
  
  
#%%  
  
b = [i + '\_(город)' for i in a]  
for i, v in enumerate(a):  
 df.loc[df['WikiUrl'] == v, 'WikiUrl'] = b[i]  
  
#%%  
  
pool = ThreadPool(20)  
results = list(pool.map(checkurl, b))  
  
#%%  
  
a = [i for i in results if i is not None]  
for i, v in enumerate(a):  
 df.loc[df['WikiUrl'] == v, 'WikiUrl'] = 'https://ru.wikipedia.org/w/index.php?search=' + \  
 v.replace('https://ru.wikipedia.org/wiki/', '').replace('\_(город)', '')  
  
#%%  
  
df  
  
#%%  
  
def checkurl(url):  
 while True:  
 try:  
 s = requests.get(url).text  
 page = lxml.etree.fromstring(s, parser=lxml.etree.HTMLParser())  
 break  
 except:  
 pass  
 if len(page.xpath('//\*[@class="infobox-image"]/span/a')) < 1:  
 if len(page.xpath('//\*[@class="wikidata-claim"]/span/a')) < 1:  
 if len(page.xpath('//\*[@class="infobox"]/tr[3]/td/span/span/a')) < 1:  
 try:  
 return page.xpath('//\*[@class="image"]')[0].get('href')  
 except:  
 print('4 ' + url + '\n', end='')  
 else:  
 try:  
 return page.xpath('//\*[@class="infobox"]/tr[3]/td/span/span/a]')[0].get('href')  
 except:  
 print('3 ' + url + '\n', end='')  
 try:  
 return page.xpath('//\*[@class="image"]')[0].get('href')  
 except:  
 print('34 ' + url + '\n', end='')  
 else:  
 try:  
 return page.xpath('//\*[@class="wikidata-claim"]/span/a')[0].get('href')  
 except:  
 print('2 ' + url + '\n', end='')  
 try:  
 return page.xpath('//\*[@class="image"]')[0].get('href')  
 except:  
 print('24 ' + url + '\n', end='')  
 else:  
 try:  
 return page.xpath('//\*[@class="infobox-image"]/span/a')[0].get('href')  
 except:  
 print('1 ' + url + '\n', end='')  
 try:  
 return page.xpath('//\*[@class="image"]')[0].get('href')  
 except:  
 print('14 ' + url + '\n', end='')  
  
  
pool = ThreadPool(10)  
results = list(pool.map(checkurl, df['WikiUrl']))  
  
  
#%%  
  
a = []  
for i in results:  
 if i is not None:  
 if 'https' in i:  
 print(i)  
 a.append(None)  
 else:  
 a.append('https://ru.wikipedia.org' + i)  
 else:  
 a.append(None)  
pd.DataFrame({  
 'url': df['WikiUrl'],  
 'pic': a  
}).to\_csv('123.csv')  
  
#%%  
  
# //\*[@id="file"]/a  
def download\_pic(url):  
 if url is not None:  
 try:  
 s = requests.get(url).text  
 except:  
 print('1 '+ url + '\n', end='')  
 return None  
 page = lxml.etree.fromstring(s, parser=lxml.etree.HTMLParser())  
 try:  
 return page.xpath('//\*[@id="file"]/a')[0].get('href')  
 except:  
 print(url + '\n', end='')  
 try:  
 return page.xpath('//\*[@class="image"]')[0].get('href')  
 except:  
 print('4 ' + url + '\n', end='')  
  
pool = ThreadPool(50)  
results = list(pool.map(download\_pic, a))  
  
#%%  
  
a = []  
for i in results:  
 if i is not None:  
 a.append('https:' + i)  
 else:  
 a.append(None)  
c = pd.Series(a, name='PicUrl')  
df = pd.concat([df, c], axis=1)  
df  
  
#%%  
  
# //\*[@id="sidebar\_content"]/div[2]/ul/li[1]/a  
c = pd.Series(['https://www.openstreetmap.org/search?query=' + i for i in df['City']], name='MapUrl')  
df = pd.concat([df, c], axis=1)  
df  
  
#%%  
  
# http://maps.google.com/maps/api/geocode/json?address=  
# key = 'b8b4742f21550c630a8bd5d1f803a2b6'  
# s = requests.get(f"http://api.openweathermap.org/geo/1.0/direct?q={'Perm'}&limit=1&appid={key}").text  
# s  
  
#%%  
  
# //\*[@id="search"]/table/tr[3]/td[2]/a[1]  
# def coordinates(url):  
# if url is not None:  
# s = requests.get(url).text  
# page = lxml.etree.fromstring(s, parser=lxml.etree.HTMLParser())  
# try:  
# return page.xpath('//\*[@id="search"]/table/tr[3]/td[2]/a')[0].get('href')  
# except:  
# return None  
#  
# a = list(df['MapUrl'])  
#  
# pool = ThreadPool(50)  
# results = list(pool.map(coordinates, a))  
  
  
#%%  
  
# a = 0  
# for i in results:  
# if i is None:  
# a += 1  
# a, len(results)  
  
#%%  
  
# a = ['http://www.geonames.org' + i for i in results]  
  
  
#%%  
  
# def coordinates(url):  
# if url is not None:  
# s = requests.get(url).text  
# page = lxml.etree.fromstring(s, parser=lxml.etree.HTMLParser())  
# try:  
# return page.xpath('//\*[@class="edit-mode move-mode"]')[0].Text, \  
# page.xpath('//\*[@class="edit-mode move-mode"]')[1].Text, s  
# except Exception as e:  
# return e, s  
#  
# b = [a[i] for i in range(100)]  
# pool = ThreadPool(50)  
# results = list(pool.map(coordinates, b))  
  
#%%  
  
# with open('123.html', 'w') as f:  
# f.write(results[0][1])  
#  
# results[0]  
  
#%%  
  
nomi = pgeocode.Nominatim('ru')  
nomi.query\_postal\_code('Perm')  
  
#%%  
  
nom = nomi.\_data\_frame  
nom['accuracy'] = nom['accuracy'].astype('float')  
nom = nom.sort\_values('accuracy', ascending=False)  
nom['place\_name'] = nom['place\_name'].str.lower()  
nom.to\_csv('123.csv')  
nom  
  
#%%  
  
l = list(df['City'])  
a = [nom.loc[nom['place\_name'] == i.lower().replace('ё','е')] for i in l]  
  
#%%  
  
j = 0  
for i in a:  
 if len(i) == 0:  
 j += 1  
j  
  
#%%  
  
def f(i):  
 try:  
 return tuple(i.reset\_index(drop=True).loc[0,['latitude', 'longitude']])  
 except:  
 return None  
  
b = [f(i) for i in a]  
  
#%%  
  
nones = df.loc[[i for i, e in enumerate(b) if e is None], 'City']  
nones  
  
#%%  
  
nom = Nominatim(user\_agent="http")  
loc = nom.geocode('пермь')  
loc.raw  
  
#%%  
  
for i, v in nones.items():  
 loc = nom.geocode('пермь')  
 b[i] = (loc.raw['lat'], loc.raw['lon'])  
  
#%%  
  
nones = df.loc[[i for i, e in enumerate(b) if e is None], 'City']  
nones  
  
#%%  
  
c = pd.Series([i[0] for i in b], name='Latitude')  
df = pd.concat([df, c], axis=1)  
df  
  
#%%  
  
c = pd.Series([i[1] for i in b], name='Longitude')  
df = pd.concat([df, c], axis=1)  
df  
#%%  
wiki\_wiki = wikipediaapi.Wikipedia('ru')  
page\_py = wiki\_wiki.page('Астрахань')  
page\_py.summary  
#%%  
def wiki(s):  
 try:  
 page\_py = wiki\_wiki.page(s).summary  
 return page\_py  
 except:  
 print(s + '\n', end="")  
 return None  
  
pool = ThreadPool(50)  
results = list(pool.map(wiki,  
 [i.replace("https://ru.wikipedia.org/wiki/", "") for i in df['WikiUrl']]))  
#%%  
c = pd.Series(results, name='WikiSummary')  
df = pd.concat([df, c], axis=1)  
df  
#%%  
df.loc[df['WikiSummary'] == ""]  
#%%  
df.to\_csv('Database/NewDatabase.csv')  
#%%

## Класс Processing

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Diagnostics;

using System.Diagnostics.CodeAnalysis;

using System.Globalization;

using System.IO;

using System.IO.Compression;

using System.Linq;

using System.Runtime.Serialization.Formatters;

using System.Threading;

using LumenWorks.Framework.IO.Csv;

namespace Database

{

public static class Processing

{

private static Dictionary<char, List<City>> \_databaseFind;

private static string[] \_databaseCities;

private static Dictionary<string, User> \_databaseUsers;

private static readonly Random Random = new Random();

private static Timer \_userTimer;

#if DEBUG

private static Logging Logging = new Logging(Logging.LevelLogging.DEBUG, "Database.log", true);

#else

private static Logging Logging = new Logging(Logging.LevelLogging.INFO, "Database.log", true);

#endif

public delegate void TelegramUserTimerMinuteDelegate(string ID, User user, DateTime currentTime);

public static event TelegramUserTimerMinuteDelegate TelegramUserTimerMinute;

public static Dictionary<string, User> DatabaseUsers => \_databaseUsers;

public static Dictionary<char, List<City>> DatabaseFind => \_databaseFind; //Внешний доступ к базе

private static readonly string[] DoubleNameCities = new[]

{

"нижняя", "петров", "сухой", "набережные", "верхняя", "новая", "красное", "малая", "белая", "советская",

"минеральные", "новый", "старая", "сергиев", "старый", "западная", "красный", "вятские", "верхний",

"мариинский", "гаврилов", "большой", "полярные", "лодейное", "дагестанские", "горячий", "сосновый",

"вышний", "нижние", "великие", "великий", "павловский", "нижний",

};

public class City

{

public string Name;

public int Population;

public string GoogleUrl;

public string YandexUrl;

public string WikiUrl;

public string PicUrl;

public string MapUrl;

public decimal Latitude;

public decimal Longitude;

public string WikiSummary;

public static implicit operator string(City c) => c.Name;

public override string ToString() => Name;

}

public class User

{

public string Id => \_id;

private readonly string \_id;

public DateTime LastCall => \_lastCall;

private DateTime \_lastCall;

private Dictionary<char, List<City>> \_usedCities;

public char NextLetter { get; set; }

public string OutCity { get; set; }

public byte LetterNumberFromEnd { get; set; }

public byte RequestsToContinueGame { get; set; }

public Dictionary<char, List<City>> UsedCities

{

get

{

\_lastCall = DateTime.Now;

RequestsToContinueGame = 0;

return \_usedCities;

}

set

{

\_usedCities = value;

RequestsToContinueGame = 0;

\_lastCall = DateTime.Now;

}

}

public User(string id)

{

\_id = id;

\_usedCities = new Dictionary<char, List<City>>();

foreach (char c1 in Enumerable.Range('а', 'я' - 'а' + 1).Select(c => (char)c))

{

\_usedCities.Add(c1, new List<City>());

}

\_lastCall = DateTime.Now;

}

public override string ToString()

{

return $"User: LastCall: {LastCall}, NextLetter: {NextLetter}, " +

$"OutCity: {OutCity}, LetterNumberFromEnd: {LetterNumberFromEnd}, " +

$"RequestsToContinueGame {RequestsToContinueGame}";

}

}

public class CityEqualityComparerClass : IEqualityComparer<City>

{

public bool Equals(City x, City y)

{

try

{

return String.Equals(x.Name, y.Name, StringComparison.CurrentCultureIgnoreCase);

}

catch

{

return false;

}

}

public int GetHashCode([DisallowNull] City obj)

{

return obj.Name.ToLower().GetHashCode();

}

}

public static readonly CityEqualityComparerClass CityEqualityComparer = new CityEqualityComparerClass();

public static char NextLetterUser(string id)

{

User user = \_databaseUsers[id];

if (user.LetterNumberFromEnd < user.OutCity.Length)

{

user.NextLetter = user.OutCity[^(++user.LetterNumberFromEnd)];

Logging.DEBUG(user.ToString());

}

return user.NextLetter;

}

public static void ReadCsv()

{

// инициализация коллекций

\_databaseFind = new Dictionary<char, List<City>>();

\_databaseUsers = new Dictionary<string, User>();

// open the file "data.csv" which is a CSV file with headers

using (CsvReader csv =

new CsvReader(new StreamReader("NewDatabase.csv"), true))

{

List<string> cities = new List<string>();

//int fieldCount = csv.FieldCount;

//string[] headers = csv.GetFieldHeaders();

while (csv.ReadNextRecord())

{

if (!\_databaseFind.ContainsKey(csv[1][0]))

{

\_databaseFind.Add(csv[1][0], new List<City>());

}

Debug.Print($"{csv[2]} {csv[3]} {csv[4]} {csv[5]} {csv[6]} {csv[7]} {csv[8]} {csv[9]} {csv[10]}");

\_databaseFind[csv[1][0]].Add(new City

{

Name = csv[2],

Population = int.Parse(csv[3]),

GoogleUrl = csv[4],

YandexUrl = csv[5],

WikiUrl = csv[6],

PicUrl = csv[7].Replace("https", "http"),

MapUrl = csv[8],

Latitude = decimal.Parse(csv[9].Replace(".", CultureInfo.InvariantCulture.NumberFormat.NumberDecimalSeparator), CultureInfo.InvariantCulture),

Longitude = decimal.Parse(csv[10].Replace(".", CultureInfo.InvariantCulture.NumberFormat.NumberDecimalSeparator), CultureInfo.InvariantCulture),

WikiSummary = csv[11]

});

cities.Add(csv[2]);

}

\_databaseCities = cities.ToArray();

for (int i = 0; i < \_databaseCities.Length; i++)

{

\_databaseCities[i] = \_databaseCities[i].ToLower();

}

}

\_userTimer = new Timer(On\_userTimer, null, 0, 60000);

}

private static void On\_userTimer(object state)

{

foreach (KeyValuePair<string, User> databaseUser in \_databaseUsers)

{

var s = databaseUser.Key.Split(".");

var currentTime = DateTime.Now;

switch (s[0])

{

case "Telegram":

TelegramUserTimerMinute?.Invoke(databaseUser.Key, databaseUser.Value, currentTime);

break;

default:

break;

}

}

}

public static void DeleteUser(string id)

{

\_databaseUsers.Remove(id);

}

public static double NormalRandom(double mu = 0, double sigma = 1, double left = -1.7, double right = 1.3)

{

var u1 = 1.0 - Random.NextDouble();

var u2 = 1.0 - Random.NextDouble();

var rand = Math.Sqrt(-2.0 \* Math.Log(u1)) \* Math.Sin(2.0 \* Math.PI \* u2);

while (!(left <= rand && rand < right))

{

u1 = 1.0 - Random.NextDouble();

u2 = 1.0 - Random.NextDouble();

rand = Math.Sqrt(-2.0 \* Math.Log(u1)) \* Math.Sin(2.0 \* Math.PI \* u2);

}

rand = mu + sigma \* rand;

return (rand - left) / Math.Abs(right - left);

}

/// <param name="city"> Город отправленный пользователем </param>

/// <param name="id"> ID пользователя (в начале советую писать из какого он мессенджера) </param>

/// <param name="cityIsUsed"> Был ли использован этот город пользователем ранее </param>

/// <param name="nextLetter"> Буква на которую должен отвечать пользователь</param>

/// <param name="letterNumberFromEnd"> номер буквы с конца на которую бот возвращает значение (0 = ответ на последнюю букву)</param>

/// <param name="wikiUrl"> Ссылка на википедию города</param>

/// <param name="yandexUrl"> Ссылка на запрос в яндексе по городу</param>

/// <param name="googleUrl"> Ссылка на запрос в гугле по городу</param>

/// <param name="mapUrl"> Ссылка на место на карте</param>

/// <param name="coordinateCity"> Координаты города (latitude - широта, longitude - долгота)</param>

/// <param name="photoUrl"> Ссылка на фото из города</param>

/// <param name="outCity"> null значит города не существует в бд, "" - боту нечего отвечать </param>

/// <param name="onLastLetter"> ответил ли пользователь на последнюю букву предыдущего слова </param>

public static void Get(ref string city, string id, out bool onLastLetter, out bool cityIsUsed, out string outCity, out char nextLetter, out byte letterNumberFromEnd, out string wikiUrl,

out string yandexUrl, out string googleUrl, out string mapUrl, out (decimal latitude, decimal longitude) coordinateCity, out string photoUrl)

{

// изначальные значения

outCity = null;

letterNumberFromEnd = 0;

onLastLetter = true;

cityIsUsed = false;

wikiUrl = null;

yandexUrl = null;

googleUrl = null;

mapUrl = null;

photoUrl = null;

nextLetter = default;

coordinateCity.longitude = default;

coordinateCity.latitude = default;

string[] splittedCities = city.Split();

if (splittedCities.Length > 1)

{

splittedCities[0] = splittedCities[0].ToLower();

splittedCities[1] = splittedCities[1].ToLower();

// если название города состоит из 2 слов

if (DoubleNameCities.Contains(splittedCities[0].ToLower()))

city = splittedCities[0] + " " + splittedCities[1];

else

city = splittedCities[0];

}

else

{

city = splittedCities[0];

}

city = city.ToLower();

if (\_databaseCities.Contains(city))

{

outCity = "";

if (!\_databaseUsers.ContainsKey(id))

\_databaseUsers.Add(id, new User(id)); // Если человека нет в базе пользователей

else

{

if (city[0] != \_databaseUsers[id].NextLetter) // проверка введено ли на правильную букву

{

onLastLetter = false;

return;

}

}

if (\_databaseUsers[id].UsedCities[city[0]].Contains(new City() { Name = city }, CityEqualityComparer)) // проверка на использованность города

{

cityIsUsed = true;

}

else

{

foreach (char c in city.Reverse())

{

if (!\_databaseFind.ContainsKey(c))

{

letterNumberFromEnd++;

continue;

}

List<City> except = \_databaseFind[c]

.Except(\_databaseUsers[id].UsedCities[c], CityEqualityComparer).ToList();

if (except.Count != 0)

{

int numberOfCity =

(int)Math.Round((NormalRandom() - 1d / (except.Count \* 2)) \* except.Count);

outCity = except[numberOfCity].Name; // используется смещенное нормальное распределение чтобы давать более редкие города чаще

\_databaseUsers[id].UsedCities[city[0]].Add(new City() { Name = city });

\_databaseUsers[id].UsedCities[c].Add(new City() { Name = outCity.ToLower() });

bool userWin = true; // проверка на победу + nextLetter

for (int i = outCity.Length - 1; i > 0; i--)

{

nextLetter = outCity[i];

if (\_databaseFind.ContainsKey(nextLetter))

if (\_databaseFind[nextLetter].Except(\_databaseUsers[id].UsedCities[nextLetter], CityEqualityComparer).Any())

{

\_databaseUsers[id].NextLetter = nextLetter;

\_databaseUsers[id].OutCity = outCity;

\_databaseUsers[id].LetterNumberFromEnd = letterNumberFromEnd;

userWin = false;

break;

}

}

if (userWin)

{

outCity = "";

}

else

{

City outCityClass = except[numberOfCity];

wikiUrl = outCityClass.WikiUrl;

yandexUrl = outCityClass.YandexUrl;

googleUrl = outCityClass.GoogleUrl;

mapUrl = outCityClass.MapUrl;

photoUrl = outCityClass.PicUrl;

coordinateCity.longitude = outCityClass.Longitude;

coordinateCity.latitude = outCityClass.Latitude;

}

break;

}

else

{

letterNumberFromEnd++;

}

}

}

}

}

/// <summary>

/// get с поиском по окружности в километрах

/// </summary>

/// <param name="city"> Город отправленный пользователем </param>

/// <param name="id"> ID пользователя (в начале советую писать из какого он мессенджера) </param>

/// <param name="cityIsUsed"> Был ли использован этот город пользователем ранее </param>

/// <param name="onLastLetter"> ответил ли пользователь на последнюю букву предыдущего слова </param>

/// <param name="letterNumberFromEnd"> номер буквы с конца на которую бот возвращает значение (0 = ответ на последнюю букву)</param>

/// <param name="wikiUrl"> Ссылка на википедию города</param>

/// <param name="yandexUrl"> Ссылка на запрос в яндексе по городу</param>

/// <param name="googleUrl"> Ссылка на запрос в гугле по городу</param>

/// <param name="nextLetter"> Буква на которую должен отвечать пользователь</param>

/// <param name="mapUrl"> Ссылка на место на карте</param>

/// <param name="coordinateCity"> Координаты города (latitude - широта, longitude - долгота)</param>

/// <param name="photoUrl"> Ссылка на фото из города</param>

/// <param name="outCity"> null значит города не существует в бд, '' - все города отгаданы</param>

/// <param name="coordinateUser"> latitude - широта, longitude - долгота</param>

/// <param name="searchRadius"> в километрах</param>

public static void Get(ref string city, string id, (decimal latitude, decimal longitude) coordinateUser, double searchRadius,

out bool onLastLetter, out bool cityIsUsed, out string outCity, out char nextLetter, out byte letterNumberFromEnd, out string wikiUrl,

out string yandexUrl, out string googleUrl, out string mapUrl, out (decimal latitude, decimal longitude) coordinateCity, out string photoUrl)

{

// изначальные значения

outCity = null;

letterNumberFromEnd = 0;

onLastLetter = true;

cityIsUsed = false;

wikiUrl = null;

yandexUrl = null;

googleUrl = null;

mapUrl = null;

photoUrl = null;

nextLetter = default;

coordinateCity.longitude = default;

coordinateCity.latitude = default;

string[] splittedCities = city.Split();

if (splittedCities.Length > 1)

{

splittedCities[0] = splittedCities[0].ToLower();

splittedCities[1] = splittedCities[1].ToLower();

// если название города состоит из 2 слов

if (DoubleNameCities.Contains(splittedCities[0].ToLower()))

city = splittedCities[0] + " " + splittedCities[1];

else

city = splittedCities[0];

}

else

{

city = splittedCities[0];

}

city = city.ToLower();

if (\_databaseCities.Contains(city))

{

outCity = "";

if (!\_databaseUsers.ContainsKey(id))

\_databaseUsers.Add(id, new User(id)); // Если человека нет в базе пользователей

else

{

if (city[0] != \_databaseUsers[id].NextLetter) // проверка введено ли на правильную букву

{

onLastLetter = false;

return;

}

}

if (\_databaseUsers[id].UsedCities[city[0]].Contains(new City() { Name = city }, CityEqualityComparer)) // проверка на использованность города

{

cityIsUsed = true;

}

else

{

foreach (char c in city.Reverse())

{

if (!\_databaseFind.ContainsKey(c))

{

letterNumberFromEnd++;

continue;

}

List<City> except = \_databaseFind[c]

.Except(\_databaseUsers[id].UsedCities[c], CityEqualityComparer).ToList();

foreach (City city1 in except)

{

double alpha2 = Math.Pow((searchRadius \* 180d) / (Math.PI \* 6371d), 2);

if (Math.Pow((double)(city1.Latitude - coordinateUser.latitude), 2) +

Math.Pow((double)(city1.Longitude - coordinateUser.longitude), 2) > alpha2)

{

except.Remove(city1);

}

}

if (except.Count != 0)

{

int numberOfCity =

(int)Math.Round((NormalRandom() - 1d / (except.Count \* 2)) \* except.Count);

outCity = except[numberOfCity].Name; // используется смещенное нормальное распределение чтобы давать более редкие города чаще

\_databaseUsers[id].UsedCities[city[0]].Add(new City() { Name = city });

\_databaseUsers[id].UsedCities[c].Add(new City() { Name = outCity.ToLower() });

bool userWin = true; // проверка на победу + nextLetter

for (int i = outCity.Length - 1; i > 0; i--)

{

nextLetter = outCity[i];

if (\_databaseFind.ContainsKey(nextLetter))

if (\_databaseFind[nextLetter].Except(\_databaseUsers[id].UsedCities[nextLetter], CityEqualityComparer).Any())

{

\_databaseUsers[id].NextLetter = nextLetter;

\_databaseUsers[id].OutCity = outCity;

\_databaseUsers[id].LetterNumberFromEnd = letterNumberFromEnd;

userWin = false;

break;

}

}

if (userWin)

{

outCity = "";

}

else

{

City outCityClass = except[numberOfCity];

wikiUrl = outCityClass.WikiUrl;

yandexUrl = outCityClass.YandexUrl;

googleUrl = outCityClass.GoogleUrl;

mapUrl = outCityClass.MapUrl;

photoUrl = outCityClass.PicUrl;

coordinateCity.longitude = outCityClass.Longitude;

coordinateCity.latitude = outCityClass.Latitude;

}

break;

}

else

{

letterNumberFromEnd++;

}

}

}

}

}

public static string DatabaseFindToString()

{

string s = "";

foreach (KeyValuePair<char, List<City>> keyValuePair in \_databaseFind)

{

bool first = true;

s += keyValuePair.Key + ": ";

foreach (City city in keyValuePair.Value)

{

if (first)

{

s += city.Name + " " + city.Population + "\n";

first = false;

}

else

{

s += " " + city.Name + " " + city.Population + "\n";

}

}

}

return s;

}

public static string DatabaseCitiesToString()

{

string s = "";

foreach (string city in \_databaseCities)

{

s += city + ", ";

}

return s;

}

}

}

## Класс Logging

using System;

using System.IO;

namespace Database

{

public class Logging

{

public enum LevelLogging : byte

{

DEBUG = 0,

INFO = 1,

WARNING = 2,

ERROR = 3,

FATAL = 4,

NOLOGGING = 5,

}

public Logging(LevelLogging levelLogging, string pathToLogFile, bool writeInConsole = false)

{

\_levelLogging = levelLogging;

FileInfo fileInf = new FileInfo(pathToLogFile);

if (!fileInf.Exists)

fileInf.Create();

\_pathToLogFile = pathToLogFile;

\_writeInConsole = writeInConsole;

}

private readonly string \_pathToLogFile;

private readonly LevelLogging \_levelLogging;

private readonly bool \_writeInConsole;

public void DEBUG(string message = "")

{

if (\_levelLogging <= LevelLogging.DEBUG)

{

string mes = $"{DateTime.Now.ToShortDateString()} {DateTime.Now.ToShortTimeString()} DEBUG: {message}";

File.AppendAllText(\_pathToLogFile, mes);

if (\_writeInConsole)

Console.WriteLine(mes);

}

}

public void INFO(string message = "")

{

if (\_levelLogging <= LevelLogging.INFO)

{

string mes = $"{DateTime.Now.ToShortDateString()} {DateTime.Now.ToShortTimeString()} INFO: {message}";

File.AppendAllText(\_pathToLogFile, mes);

if (\_writeInConsole)

Console.WriteLine(mes);

}

}

public void ERROR(string message = "")

{

if (\_levelLogging <= LevelLogging.ERROR)

{

string mes = $"{DateTime.Now.ToShortDateString()} {DateTime.Now.ToShortTimeString()} ERROR: {message}";

File.AppendAllText(\_pathToLogFile, mes);

if (\_writeInConsole)

Console.WriteLine(mes);

}

}

public void WARNING(string message = "")

{

if (\_levelLogging <= LevelLogging.WARNING)

{

string mes = $"{DateTime.Now.ToShortDateString()} {DateTime.Now.ToShortTimeString()} WARNING: {message}";

File.AppendAllText(\_pathToLogFile, mes);

if (\_writeInConsole)

Console.WriteLine(mes);

}

}

public void FATAL(string message = "")

{

if (\_levelLogging <= LevelLogging.FATAL)

{

string mes = $"{DateTime.Now.ToShortDateString()} {DateTime.Now.ToShortTimeString()} FATAL: {message}";

File.AppendAllText(\_pathToLogFile, mes);

if (\_writeInConsole)

Console.WriteLine(mes);

}

}

}

}

# Тестирование

Так как все реализованные функции задействованы в боте, отдельное тестирование данных классов не требуется.

# Заключение

В результате работы был создан чат-бот, который способен вести с пользователем игру в Города[[1]](#footnote-1), также бот способен предоставлять пользователю возможность получения справки из электронной библиотеки «Википедия» с подробной информацией о городе, фотографию города, которая опубликована в статье о городе в электронной библиотеке «Википедия», предоставлять ссылки в поисковые системы «Google» и «Яндекс» по запросу данного города, предоставлять пользователю геолокацию города с местоположением на карте (не поддерживается в веб-версии) и возможностью построить маршрут до данного города.

Вся система работает согласно техническому заданию, все задуманные возможности реализованы.

# Список источников

**A Fast CSV Reader** [В Интернете] // LiumenWorks. - 2021 г.. - https://www.codeproject.com/Articles/9258/A-Fast-CSV-Reader.

**API Documentation** [В Интернете] // OpenStreetMap. - 2021 г.. - https://www.openstreetmap.org/.

**C# Documentation** [В Интернете] // Microsoft Docs. - Microsoft. - 2021 г.. - https://docs.microsoft.com/ru-ru/dotnet/csharp/.

**GeoPy’s documentation** [В Интернете] // GeoPy. - 2021 г.. - https://geopy.readthedocs.io/en/stable/.

**lxml - XML and HTML with Python** [В Интернете] // lxml. - 2021 г.. - https://lxml.de/.

**openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files**   
[В Интернете] // openpyxl. - Eric Gazoni, Charlie Clark. - 2021 г.. - https://openpyxl.readthedocs.io/en/stable/.

**pandas documentation** [В Интернете] // Pydata. - NumFOCUS. - 2021 г.. - https://pandas.pydata.org/docs/.

**Pgeocode** [В Интернете] // Pgeocode. - 2021 г.. - https://pgeocode.readthedocs.io/en/latest/.

**Python 3.9.6 documentation** [В Интернете] // Python 3.9.6 documentation. - 2021 г.. - https://docs.python.org/3/index.html.

**Requests: HTTP for Humans** [В Интернете] // Requests Documentation. - 2021 г.. - https://docs.python-requests.org/en/master/.

**Wikipedia API** [В Интернете] // Wikipedia API. - 2021 г.. - https://wikipedia-api.readthedocs.io/.

**Электронная энциклопедия Wikipedia** [В Интернете] // Электронная энциклопедия Wikipedia. - 2021 г.. - https://ru.wikipedia.org/.

1. Города́ — игра, в которой каждый участник в свою очередь называет реально существующий город России, название которого начинается на ту букву, которой оканчивается название города предыдущего участника. [↑](#footnote-ref-1)