Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего

образования «Московский государственный технический университет имени

Н.Э. Баумана (национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

**Отчет по лабораторным работам № 5, 6 и домашней работе по курсу**

**Базовые компоненты интернет-технологий**

**«Разработка телеграмм-бота»**

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| студент группы ИУ5-34Б |  |
| Харитонов Андрей |  |
| Подпись и дата:  29.12.21 | Подпись и дата: |

Москва, 2021

**Постановка задачи**

1. Необходимо реализовать телеграмм бота, представляющего конечный автомат, и написать модульные тесты для него.

Файл “bot.py”

from config.Token import token

from botManager import Manager

import config.playersList as plist

import telebot

bot = telebot.TeleBot(token)

managers = {}

@bot.message\_handler(commands = ['start'])

def cmd\_start(mess):

id = mess.chat.id

managers[id] = Manager(bot, id)

managers[id].SetStart()

managers[id].DrawInterface()

@bot.message\_handler(content\_types = ['text'])

def mess\_process(mess):

id = mess.chat.id

try:

managers[id].state.processMess(mess)

managers[id].NextState()

managers[id].DrawInterface()

except KeyError:

bot.send\_message(id, "Введите /start чтобы начать общение с ботом.")

@bot.callback\_query\_handler(func = lambda call : True)

def keyboard\_process(call):

id = call.from\_user.id

try:

managers[id].state.processCallBack(call)

managers[id].NextState()

managers[id].DrawInterface()

except KeyError:

bot.send\_message(id, "Введите /start чтобы начать общение с ботом.")

if \_\_name\_\_ == "\_\_main\_\_":

bot.polling(non\_stop = True)

Файл “botManager.py”

from States.InitStates import InitStates

class Manager:

def \_\_init\_\_(self, bot, chatId):

self.ignore = False

self.bot = bot

self.chatId = chatId

self.states = {}

self.state = None

self.start = "Start"

self.actualname = ""

InitStates(self.states)

def DrawInterface(self):

if not self.ignore:

self.\_TryGetInterface()

else:

self.ignore = False

def NextState(self):

next = self.\_TryGetNext()

if (next == 'Ignore'):

self.ignore = True

elif next != None:

self.\_CheckName(next, "NextState: {}".format(self.actualname))

self.state = self.\_TryCreateState(next, self.state.kwargs)

self.actualname = next

def SetStart(self):

self.\_CheckName(self.start, "SetStart")

self.state = self.\_TryCreateState(self.start)

self.actualname = self.start

def \_CheckName(self, name, funcName):

if name not in self.states.keys():

raise Exception("Invalid name of state: {} in {}.".format(name, funcName))

def \_TryGetInterface(self):

try:

self.state.getInterface()

except KeyError:

raise ValueError("Something wrong with kwargs parametrs.")

def \_TryGetNext(self):

try:

return self.state.getNext()

except KeyError:

raise ValueError("Something wrong with kwargs paranetrs.")

def \_TryCreateState(self, name, kwargs = None):

try:

return self.states[name](self.bot, self.chatId, kwargs)

except KeyError:

raise ValueError("Something wrong with kwargs paranetrs.")

Пакет “States”

Файл “CrListState.py”

from States.AbsState import AbsState

from config.botsSpeeches import speeches

from config.playersList import playersList

class CrListState(AbsState):

def Init(self):

self.ErrNotEnoughPlayers = False

def getInterface(self) -> None:

text = self.\_getText()

self.bot.send\_message(self.chatId, text)

def processCallBack(self, callback) -> None:

self.next = 'Ignore'

def processMess(self, mess) -> None:

names = self.\_parseNames(mess.text)

if(len(names) > 2):

hostName = self.kwargs['creatingName']

self.kwargs['names'] = names

self.kwargs['strNameList'] = mess.text

self.next = 'FinalInfo'

else:

self.next = 'ErrNotEnough'

def getNext(self) -> str:

self.kwargs['althernative'] = 0

return self.next

def \_getText(self):

if self.kwargs['althernative'] == 0:

return speeches['CreateList']

elif self.kwargs['althernative'] > 0:

return speeches['CreateListAlth']

def \_parseNames(self, names):

names = str(names)

namesgroups = names.split(',')

return list( filter(lambda x : x != '', map(lambda x : x.strip(), namesgroups) ))

Файл “ErrCrName.py”

from States.AbsState import AbsState

from config.botsSpeeches import speeches

from telebot.types import InlineKeyboardButton, InlineKeyboardMarkup

class ErrCrNameState(AbsState):

def getInterface(self) -> None:

text = speeches['ErrNameAlreadyExist']

button = InlineKeyboardButton("Хорошо", callback\_data = "ErrCrName.CreateHost")

self.bot.send\_message(self.chatId, text, reply\_markup = InlineKeyboardMarkup().add(button))

def getNext(self) -> str:

self.kwargs['althernative'] = 1

return self.next

def processCallBack(self, callback) -> None:

self.GetNextFromCallback('ErrCrName', callback.data)

def processMess(self, mess) -> None:

if(mess.text == 'Хорошо'):

self.next = 'CreateHost'

else:

self.next = 'Ignore'

Файл “ErrNotEnough.py”

from States.AbsState import AbsState

from config.botsSpeeches import speeches

from telebot.types import InlineKeyboardButton, InlineKeyboardMarkup

class ErrNotEnough(AbsState):

def getInterface(self) -> None:

text = speeches['ErrNotEnoughPlayers']

button1 = InlineKeyboardButton("Я введу игроков занаво", callback\_data = "ErrNotEnough.CreateList")

button2 = InlineKeyboardButton(":(", callback\_data = "ErrNotEnough.Start")

markup = InlineKeyboardMarkup().add(button1, button2)

self.bot.send\_message(self.chatId, text, reply\_markup = markup)

def getNext(self) -> str:

self.kwargs['althernative'] = 1

return self.next

def processCallBack(self, callback) -> None:

self.GetNextFromCallback('ErrNotEnough', callback.data)

def processMess(self, mess) -> None:

self.next = 'Ignore'

Файл “FinalInfo.py”

from States.AbsState import AbsState

from config.botsSpeeches import speeches

from config.playersList import playersList

from telebot.types import InlineKeyboardMarkup, InlineKeyboardButton, KeyboardButton

class FinalInfo(AbsState):

def getInterface(self) -> None:

text = self.\_\_getText()

keyboard = self.\_\_getKeyboard()

self.bot.send\_message(self.chatId, text, reply\_markup = keyboard)

def processCallBack(self, callback) -> None:

if(callback.data == "FinalInfo.Start"):

playersList[self.kwargs['creatingName'] ] = self.kwargs['names']

self.GetNextFromCallback('FinalInfo', callback.data)

def processMess(self, mess) -> None:

self.next = 'Ignore'

def getNext(self) -> str:

self.kwargs['althernative'] = 2

return self.next

def \_\_getText(self):

text = speeches['FinalInfo']

text += '\n\nИмя организация: ' + self.kwargs['creatingName'] + '\n'

text += '\nСписок игроков: ' + self.kwargs['strNameList'] + '\n'

return text

def \_\_getKeyboard(self):

button1 = InlineKeyboardButton("Меня всё устраивает", callback\_data = "FinalInfo.Start")

button2 = InlineKeyboardButton("Я хочу изменить имя организатора", callback\_data = "FinalInfo.CreateHost")

button3 = InlineKeyboardButton("Я хочу изменить список игроков", callback\_data = "FinalInfo.CreateList")

keyboard = InlineKeyboardMarkup()

keyboard.add(button1)

keyboard.add(button2)

keyboard.add(button3)

return keyboard

Файл “InitStates.py”

from States.StartState import StartState

from States.CrHostState import CrHostState

from States.ErrCrNameState import ErrCrNameState

from States.CrListState import CrListState

from States.ErrNotEnugh import ErrNotEnough

from States.FinalInfo import FinalInfo

def InitStates(states):

states['Start'] = StartState

states['CreateHost'] = CrHostState

states['ErrCrName'] = ErrCrNameState

states['CreateList'] = CrListState

states['ErrNotEnough'] = ErrNotEnough

states['FinalInfo'] = FinalInfo

Файл “StartState.py”

from States.AbsState import AbsState

from config.botsSpeeches import speeches

from telebot.types import InlineKeyboardButton, InlineKeyboardMarkup, VoiceChatScheduled

class StartState(AbsState):

def Init(self):

self.ErrNoMessProc = False

self.kwargs = {'althernative' : 0}

def getInterface(self) -> None:

text = self.\_\_getTextInterface()

keyboard = self.\_\_getKeyboard()

self.bot.send\_message(self.chatId, text, reply\_markup = keyboard)

def processCallBack(self, callback) -> None:

self.GetNextFromCallback('Start', callback.data)

def processMess(self, mess) -> None:

self.ErrNoMessProc = True

return None

def getNext(self) -> str:

return self.next

def \_\_getKeyboard(self):

button1 = InlineKeyboardButton("Создать игру", callback\_data = "Start.CreateHost")

button2 = InlineKeyboardButton("Найти игру", callback\_data = "Start.FindHost")

return InlineKeyboardMarkup().add(button1, button2)

def \_\_getTextInterface(self):

if(self.ErrNoMessProc):

return speeches['ErrNoProcessOfText']

else:

return speeches['Start']

Файл “AbsState.py”

from abc import ABC, abstractclassmethod

from telebot.util import CustomRequestResponse

class AbsState(ABC):

def \_\_init\_\_(self, bot, chatId, kwargs = None):

self.bot = bot

self.chatId = chatId

self.kwargs = kwargs

self.next = None

self.Init()

def Init(self):

"""Расширяет конструктор класса."""

pass

def GetNextFromCallback(self, stateName, callback):

try:

self.next = self.CutCallBack(stateName, callback)

except ValueError:

self.next = 'Ignore'

def CutCallBack(self, stateName, callback) -> str:

words = callback.split('.')

if (len(words) != 2):

raise Exception("Uncorrect callback")

elif (words[0] != stateName):

raise ValueError("Callback was from another state")

else:

return words[1]

@abstractclassmethod

def getInterface() -> None:

"""Возвращает интерфейс взаимодействия с пользователем."""

pass

@abstractclassmethod

def processCallBack(callback) -> None:

"""Обрабатывает нажатие кнопки и возвращает следующее состояние."""

pass

@abstractclassmethod

def processMess(mess) -> None:

"""Обрабатывает текстовое сообщение и возвращает следующее состояние."""

pass

@abstractclassmethod

def getNext() -> str:

"""Возвращает значение следующего состояния. Если значение None, значит состояние остаётся текущим"""

Pass

Файл “CrHostName.py”

from States.AbsState import AbsState

from config.botsSpeeches import speeches

from config.playersList import playersList

class CrHostState(AbsState):

def Init(self):

self.ErrNameAlreadyExist = False

def getInterface(self) -> None:

text = self.\_getText()

self.bot.send\_message(self.chatId, text)

def processCallBack(self, callback) -> None:

self.next = 'Ignore'

def processMess(self, mess) -> None:

name = mess.text

if name in playersList.keys():

self.ErrNameAlreadyExist = True

self.next = 'ErrCrName'

else:

self.kwargs['creatingName'] = name

if(self.kwargs['althernative'] == 2):

self.next = 'FinalInfo'

else:

self.next = 'CreateList'

def getNext(self) -> str:

self.kwargs['althernative'] = 0

return self.next

def \_getText(self):

if self.kwargs['althernative'] == 0:

return speeches['CreateHost']

elif self.kwargs['althernative'] > 0:

return speeches['CreateHostAlth']

Пакет configs

Файл botSpeeches.py

speeches = {

'Start' :

'Привет! Я организую игры в тайного санту) Если ты хочешь организовать ' +

'игру, то нажми на кнопку Создать игру. Если ты пришёл сюда по приглашению, ' +

'то можешь попытаться найти свою игру по одноимённой кнопке :>',

'ErrNoProcessOfText' :

'Прости, но я не понимаю человеческую речь и могу общаться '+

'с тобой только посредством кнопок :(\n' +

'Вот варианты взаимодействия со мной.',

'CreateHost' :

'Прекрасно! Теперь напиши мне своё имя, чтобы мы были знакомы) ' +

'Но будь внимателен, так как по этому имени твои друзья будут искать игру.',

'CreateHostAlth' :

'Ладно, попробуй ещё раз. Напомню, что по этому имени твои друзья будут искать игру.',

'ErrNameAlreadyExist' :

'Прости, но это имя уже занято :(\n' +

'Я не могу тебя так назвать, так как иначе я буду путать тебя с другим моим гостем. '+

'Попробуй придумать другое имя.',

'CreateList' :

'Хорошо. А сейчас напиши мне, пожалйста, список игроков через запятую.\n'+

'Например: Миша, Ваня, Таня Синицина, Таня Воробьёва',

'CreateListAlth' :

'Напиши список игроков через запятую. Но учти, что их должно быть больше 2ух ;)\n' +

'Пример ввода: Миша, Ваня, Таня Синицина, Таня Воробьёва',

'ErrNotEnoughPlayers' :

'Ой... Играть в тайного санту вдвоём не получится( Иначе смысл в игре теряется.\n' +

'Может ты найдёшь ещё кого-то из друзей?',

'FinalInfo' :

'Супер! Здесь ниже представленны введённые тобой данные. Посмотри внимательно и измени ' +

'если что то не нравится:'

}

Файл “plauersList”

import json

playersList = {}

def ReadList(data):

with open("playersList.json", "r") as plist:

data = json.load(plist)

def SaveList(data):

with open("playersList.json", "w") as plist:

json.dumps(data)

Пакет “Tests”

Файл Manager\_tests.py

import unittest

import telebot

from States.AbsState import AbsState

from botManager import Manager

from States.StartState import StartState

from config.Token import token

class MockState(AbsState):

test1 = True

test2 = True

test3 = True

test4 = True

interfaceText = "no-interface"

def getNext(self) -> str:

if self.test1:

return "AbraCadabra"

elif self.test2:

return None

elif self.test3:

return "Start"

elif self.test4:

return "Ignore"

def getInterface(self) -> None:

self.interfaceText = "interface"

def processCallBack(self, callback) -> None:

pass

def processMess(self, mess) -> None:

pass

class TestManager(unittest.TestCase):

def setUp(self):

self.manager = Manager(telebot.TeleBot(token), 666)

def test\_setStart\_1(self):

self.manager.start = "Start"

self.manager.SetStart()

Mock = StartState(telebot.TeleBot(token), 666)

self.assertEqual(self.manager.state.chatId, Mock.chatId)

def test\_setStart\_2(self):

self.manager.start = ""

try:

self.manager.SetStart()

except Exception as e:

self.assertEqual(e.args[0], "Invalid name of state: in SetStart.")

def test\_nextState\_1(self):

self.manager.state = MockState(None, 111)

self.manager.actualname = "mock"

try:

self.manager.NextState()

except Exception as e:

self.assertEqual(e.args[0], "Invalid name of state: AbraCadabra in NextState: mock.")

def test\_nextState\_2(self):

self.manager.state = MockState(None, 111)

self.manager.state.test1 = False

state = self.manager.state

self.manager.NextState()

self.assertEqual(self.manager.state, state)

def test\_nextState\_3(self):

self.manager.state = MockState(None, 111)

self.manager.state.test1 = False

self.manager.state.test2 = False

#self.manager.NextState()

#self.assertEqual(self.manager.actualname, "Start")

def test\_IgnoreMessage\_1(self):

self.manager.state = MockState(None, 222)

self.manager.state.test1 = False

self.manager.state.test2 = False

self.manager.state.test3 = False

self.manager.NextState()

self.manager.DrawInterface()

self.assertEqual(self.manager.state.interfaceText, "no-interface")

def test\_IgnoreMessage\_2(self):

self.manager.state = MockState(None, 333)

self.manager.state.test1 = False

self.manager.NextState()

self.manager.DrawInterface()

self.assertEqual(self.manager.state.interfaceText, "interface")