МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по лабораторной работе №5
“Разработка простого бота для Telegram с использованием языка Python”

Выполнил:
Студент группы ИУ5-35Б
Евдокимов М. С.
Преподаватель:
Гапанюк Ю. Е.

Москва 2025

Цель лабораторной работы

Цель лабораторной работы: изучение разработки ботов в Telegram.

Задание лабораторной работы

Разработайте простого бота для Telegram. Бот должен использовать функциональность создания кнопок.

Листинг программы

Файл bothsides.py

```
import logging
import random
import os
from dotenv import load_dotenv

from telegram import (
    Update,
    InlineKeyboardMarkup,
    InlineKeyboardButton,
    ReplyKeyboardMarkup,
    KeyboardButton,
)
from telegram.ext import (
    Application,
    CommandHandler,
    MessageHandler,
    CallbackQueryHandler,
    ContextTypes,
    filters,
)

load_dotenv()
BOT_TOKEN = os.getenv("BOT_TOKEN")

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s | %(levelname)s | %(name)s | %(message)s",
)
log = logging.getLogger(__name__)

PLAYER = " ❌ "
BOT = " ⭕ "
EMPTY = "  "

reply_kb = ReplyKeyboardMarkup(
    keyboard=[
        [KeyboardButton("🎮 Новая игра")],
        [KeyboardButton("ℹ️ Помощь")],
    ],
    resize_keyboard=True,
)

WIN_LINES = [
    (0, 1, 2),
    (3, 4, 5),
    (6, 7, 8),
    (0, 3, 6),
    (1, 4, 7),
    (2, 5, 8),
    (0, 4, 8),
    (2, 4, 6),
]


def check_winner(board, symbol):
    for a, b, c in WIN_LINES:
        if board[a] == board[b] == board[c] == symbol:
            return True
    return False
```

```
def is_draw(board):
    return all(cell != EMPTY for cell in board)


def ai_move(board):
    for i in range(9):
        if board[i] == EMPTY:
            copy = board.copy()
            copy[i] = BOT
            if check_winner(copy, BOT):
                return i

    for i in range(9):
        if board[i] == EMPTY:
            copy = board.copy()
            copy[i] = PLAYER
            if check_winner(copy, PLAYER):
                return i

    priority = [i for i in (0, 2, 4, 6, 8) if board[i] == EMPTY]
    if priority:
        return random.choice(priority)

    free = [i for i in range(9) if board[i] == EMPTY]
    if free:
        return random.choice(free)

    return None


def board_to_keyboard(board):
    buttons = []
    for row in range(3):
        row_buttons = []
        for col in range(3):
            idx = row * 3 + col
            text = board[idx] if board[idx] != EMPTY else "☐"
            row_buttons.append(
                InlineKeyboardButton(text=text, callback_data=str(idx))
            )
        buttons.append(row_buttons)
    return InlineKeyboardMarkup(buttons)


def reset_game(context):
    context.user_data["board"] = [EMPTY] * 9
    context.user_data["game_active"] = True

    first = random.choice(["player", "bot"])
    context.user_data["first_turn"] = first
    context.user_data["player_turn"] = (first == "player")


async def start(update, context):
    user = update.effective_user.first_name or "игрок"
    text = (
        f"Привет, {user}! 👋\n"
        "Это бот с игрой «Крестики-нолики».\n\n"
        "Каждый раз случайно выбирается, кто ходит первым.\n"
        "Нажми «🎮 Новая игра», чтобы начать."
    )
    await update.message.reply_text(text, reply_markup=reply_kb)
```

```
async def help_cmd(update, context):
    text = (
        "Правила игры:\n\n"
        "• Ты — ❌, бот — ⭕.\n"
        "• В начале партии случайно выбирается, кто ходит первым.\n"
        "• Ходы делаются нажатием на клетки.\n"
        "• Бот играет не идеально — его можно обыграть 😉"
    )
    await update.message.reply_text(text, reply_markup=reply_kb)


async def new_game(update, context):
    reset_game(context)
    board = context.user_data["board"]
    first = context.user_data["first_turn"]

    if first == "player":
        await update.message.reply_text(
            "Вы ходите первыми (❌).\nВыберите клетку:",
            reply_markup=board_to_keyboard(board),
        )
    else:
        msg = await update.message.reply_text(
            "Я хожу первым (⭕)...",
            reply_markup=board_to_keyboard(board),
        )

        bot_move = ai_move(board)
        if bot_move is not None:
            board[bot_move] = BOT

        context.user_data["player_turn"] = True
        await msg.edit_text(
            "Теперь ваш ход (❌). Выберите клетку:",
            reply_markup=board_to_keyboard(board),
        )


async def text_handler(update, context):
    txt = (update.message.text or "").lower()

    if txt == "🎮 новая игра".lower():
        await new_game(update, context)
        return

    if txt in ("🆘 помощь", "/help"):
        await help_cmd(update, context)
        return

    await update.message.reply_text(
        "Нажмите «🎮 Новая игра», чтобы начать.",
        reply_markup=reply_kb,
    )


async def button_handler(update, context):
    query = update.callback_query
    await query.answer()

    if not context.user_data.get("game_active"):
        await query.edit_message_text(
            "Игра уже завершена.\nНажмите «🎮 Новая игра»."
        )
        return
```

```
    board = context.user_data["board"]
    player_turn = context.user_data["player_turn"]
    idx = int(query.data)

    if not player_turn:
        await query.answer("Сейчас ход бота ☺")
        return

    if board[idx] != EMPTY:
        await query.answer("Клетка занята.")
        return

    board[idx] = PLAYER

    if check_winner(board, PLAYER):
        context.user_data["game_active"] = False
        await query.edit_message_text(
            "Вы победили! 🎉",
            reply_markup=board_to_keyboard(board),
        )
        return

    if is_draw(board):
        context.user_data["game_active"] = False
        await query.edit_message_text(
            "Ничья 🤝",
            reply_markup=board_to_keyboard(board),
        )
        return

    context.user_data["player_turn"] = False
    bot_idx = ai_move(board)
    if bot_idx is not None:
        board[bot_idx] = BOT

    if check_winner(board, BOT):
        context.user_data["game_active"] = False
        await query.edit_message_text(
            "Я победил 😎",
            reply_markup=board_to_keyboard(board),
        )
        return

    if is_draw(board):
        context.user_data["game_active"] = False
        await query.edit_message_text(
            "Ничья 🤝",
            reply_markup=board_to_keyboard(board),
        )
        return

    context.user_data["player_turn"] = True
    await query.edit_message_text(
        "Ваш ход ( ✕ ):",
        reply_markup=board_to_keyboard(board),
    )


def main():
    if not BOT_TOKEN:
        raise SystemExit("Ошибка: BOT_TOKEN не найден в .env")

    app = Application.builder().token(BOT_TOKEN).build()

    app.add_handler(CommandHandler("start", start))
```

```
    app.add_handler(CommandHandler("help", help_cmd))
    app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, text_handler))
    app.add_handler(CallbackQueryHandler(button_handler))

    log.info("Бот запущен.")
    app.run_polling()


if __name__ == "__main__":
    try:
        main()
    except (KeyboardInterrupt, SystemExit):
        log.info("Бот остановлен.")
```

Скриншот работы программы