Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Вычислительная техника
кафедра

Командный проект

Игра «AgarIO» с возможностью подключения ботов
тема проекта

Преподаватель _____ (подпись, дата) Легалов А.И. (фамилия, инициалы)

Студент КИ14-06Б (код (номер) группы) _____ (подпись, дата) Рубан А.Г. (фамилия, инициалы)

Студент КИ14-06Б (код (номер) группы) _____ (подпись, дата) Нагуслаев Н.Т. (фамилия, инициалы)

Студент КИ14-06Б (код (номер) группы) _____ (подпись, дата) Кирилов Н.Э. (фамилия, инициалы)

Студент КИ14-06Б (код (номер) группы) _____ (подпись, дата) Костюченко А.Е. (фамилия, инициалы)

Красноярск 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью командного проекта является разработка сетевой игры «AgarIO», в которой необходимо осуществить возможность подключения ботов.

Для достижения цели были поставлены следующие задачи:

- Проанализировать существующие аналоги игр;
- Разработать клиент-серверное приложение;
- Создать API для связи бота с сервером;
- Написать пример бота на языке Python.

Актуальностью данной работы является необходимость улучшения навыка программирования у первокурсников, которые смогут писать ботов для созданной сетевой игры и делать соревнования между ними. Все это повысит интерес и мотивацию к изучению программирования.

1 Анализ предметной области

Основой для создания сетевой игры было решено взять приложение «AgarIO». Существует несколько реализаций этой игры, известные под такими названиями, как AgarIO, Чашка Петри и другие.

В исходных приложениях главной целью является управлять шариком, который может поедать другие шарики меньшего размера. В свою очередь, в роли шарика может выступать, как еда, так и другие игроки.

Выше представленные аналоги по-своему функционалы очень похожи, в основном отличается только дизайн. Они обладают такими функциями, как управление мышкой и с помощью клавиатуры, существует рейтинговая таблица. В некоторых реализациях есть чат, где игроки могут обмениваться сообщениями с другими пользователями.

Однако, нигде не реализовано подключение ботов в игру. Поэтому есть необходимость это реализовать в разрабатываемом сетевом приложении. Для этого требуется написать API для связи бота с сервером, а также прописать логику поведения бота.

2 Сценарий

Главная цель игры передвигаться по полю, поедая корм и других игроков, которые имеют меньший размер, чем он. После того, как шарик съедает некоторое количество корма или других игроков, за счет чего увеличиваются очки, и шарик начинает расти в диаметре. Основная задача заключается в том, чтобы как можно дольше выжить.

2.1 Компоненты

При заходе в игру пользователь попадает в меню, где может:

- Зарегистрировать и начать игру;

- Просмотреть игровое поле;

Также есть возможность с помощью консоли удаленно подключить к общей игре разработанного бота.

2.2 Серверная часть

Как только сервер будет запущен, он должен ожидать клиентов. Необходимо, чтобы серверная часть отвечала за обработку данных о координатах всех пользователей и могла рассылать всем клиентам обновленные координаты игроков и еды, массу и скорость каждого клиента.

2.3 Клиентская часть

После старта клиента игроку нужно управлять шариком с помощью клавиатуры (↓, →, ↑, ←, s, d, w, a). Поле, по которому движется шарик, ограничено. В режиме демонстрации поля передвижение по карте будет происходить аналогично с помощью стрелок и выше представленными буквами.

Вместо человека в игре могут соревноваться боты. Для этого бот должен обладать простейшим искусственным интеллектом. Клиенты - боты и люди, могут связываться с сервером (для обмена информации между собой) через API.

3 Архитектура

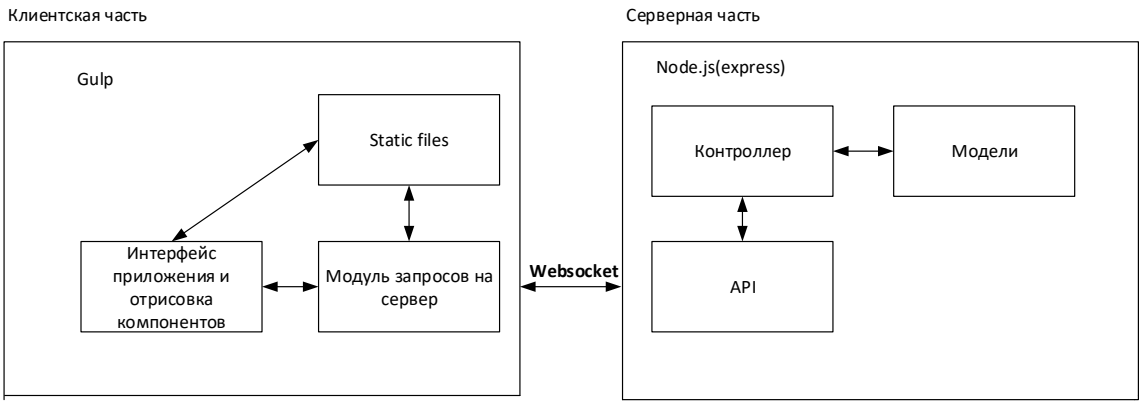

Рисунок 1 — Архитектура приложения

3.1 Клиентская часть

Index.html — отвечает за интерфейс приложения.

App.js — содержит в себе модуль запросов на сервер, а также необходим для отрисовки компонентов в браузере.

Canvas.js — требуется для определения направления движения.

Global.js — является статичным файлом, в котором хранятся основные параметры.

3.2 Серверная часть

Game_controller.js — данный модуль отвечает за логику игры, добавление и удаление еды, передвижение игроков и перерасчет массы игроков.

Player_controller.js — служит для создания нового игрока. В классе имеется конструктор для инициализации игрока со всеми необходимыми параметрами.

Server.js — необходим для инициализации всех компонентов, подключения сокетов и запуска игрового цикла.

Socket.js — для определения всех событий сокетов. В этом модуле отлавливаются сообщения от игроков и отправляются системные сообщения игрокам.

User_controller.js — служит для хранения, добавления и удаления пользователей из массива, а также для получения иноформации о списке игроков из любого файла.

Util.js — содержит различные инструменты для оперирование данными во время игры. Например, проверка никнейма, расчет массы и дистанции.

4 Словарь

При разработке игры «Agar.io» используются следующие понятия:

- Игрок;
- Бот;
- Клавиатура;
- Дисплей;
- Игра;
- Баллы.

4.1 Игрок

Взаимодействует с программой, обеспечивая тем самым выполнение своих целевых функций. Взаимодействие осуществляется через внешние устройства компьютера: клавиатуру и дисплей.

Основные действия игрока можно разделить на:

● Управление ходом игры;

● Регистрация (ввод имени);

● Просмотр игрового поля.

Прямого взаимодействие игрока с программой не происходит. Между ними существует посредник в виде внешних устройств компьютера: клавиатуры, монитора, которые обеспечивают преобразования физических воздействий человека в программные события посредством использования

клавиатуры. Обратная связь осуществляется за счет визуализации изменения состояния программы на экране дисплея.

4.2 Бот

Взаимодействует с сервером через API. Главным отличием от игрока является заложение искусственного интеллекта, в котором будут использованы простейшие алгоритмы передвижения и взаимодействия с другими игроками.

4.3 Игра

Основной программный модуль, решающий целевую задачу. Взаимодействует с клавиатурой, реагируя изменением внутреннего состояния на посылаемые воздействия игрока. Игра является достаточно сложным понятием, которое можно рассматривать как композицию следующих дополнительных понятий:

- модель игры;
- вид игры;
- контроллер игры.

Подобное видение определяется одним из наиболее распространенных в настоящее время подходом к реализации интерактивных приложений на основе концепции модель-вид-контроллер.

5 Диаграмма прецедентов

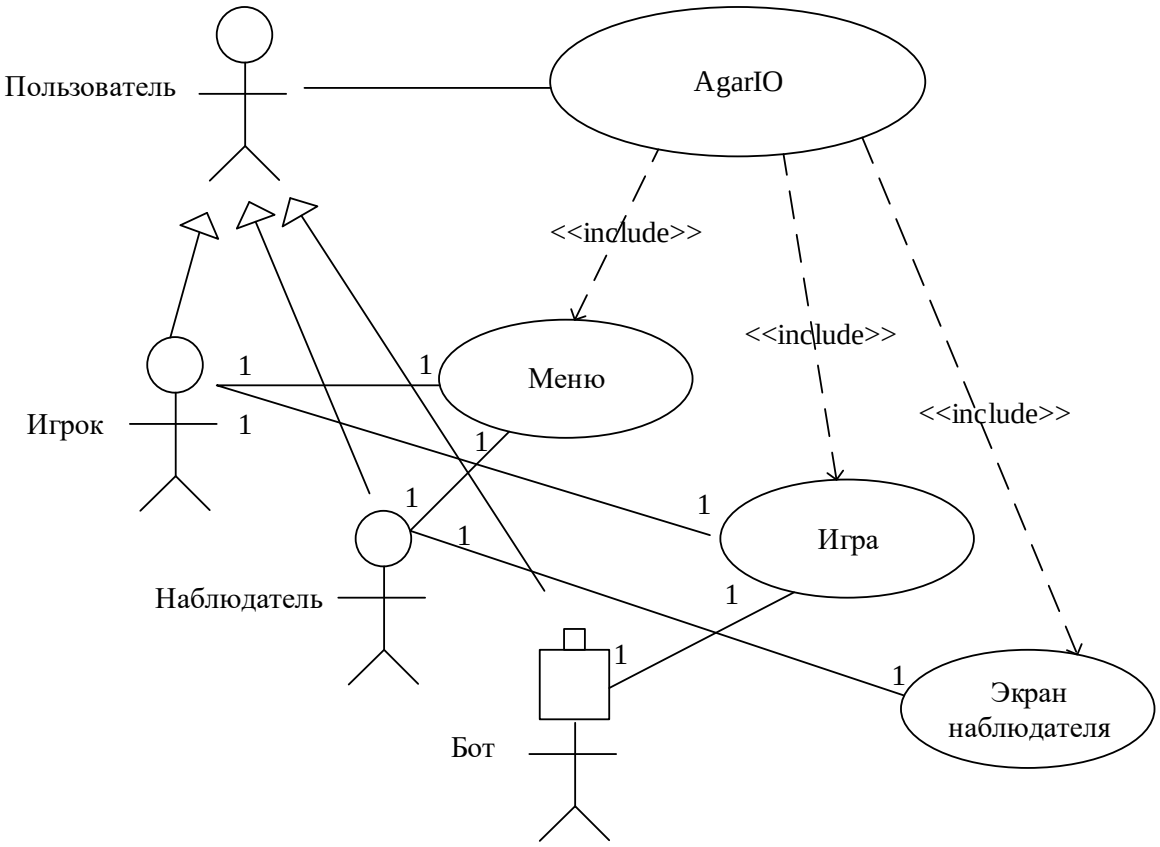

Рисунок 2 — Диаграмма прецедентов

6 Диаграмма классов

Необходимо составить диаграмму классов для сервера и для бота. Uml диаграмма для сервера представлена на рисунке 3, диаграмма для бота — на рисунке 4.

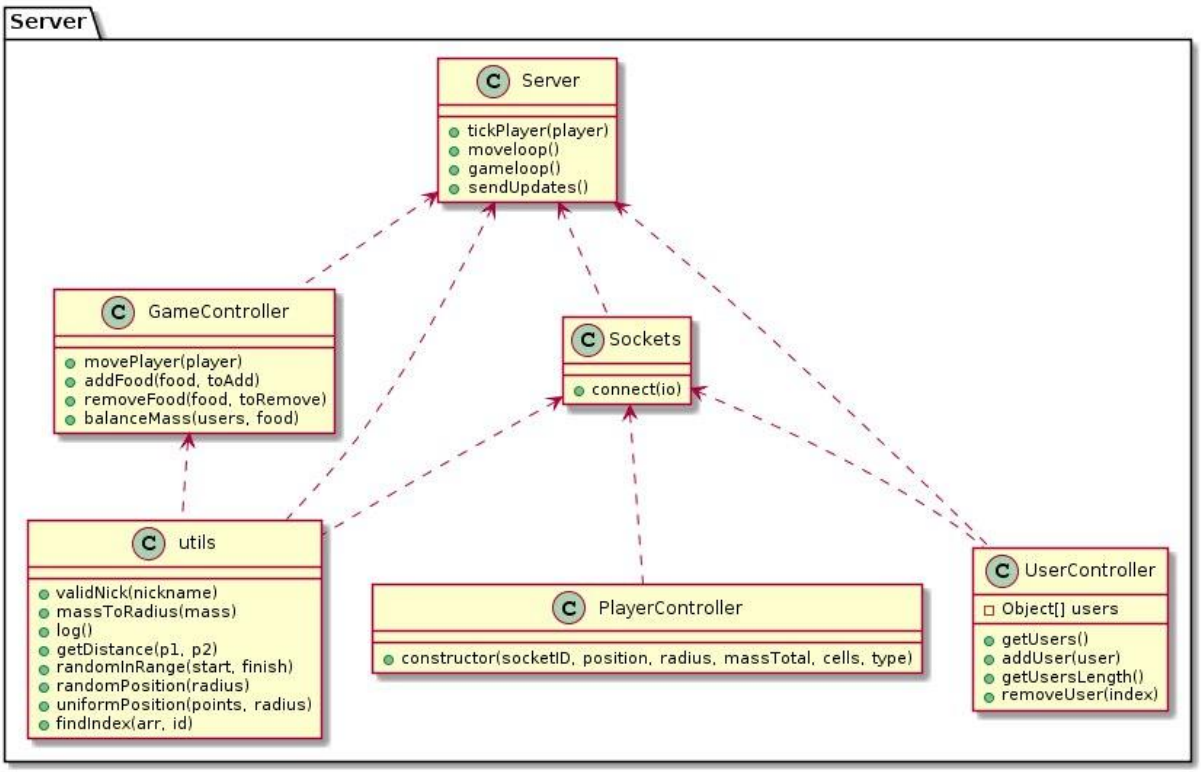

Рисунок 3 — Диаграмма классов. Сервер

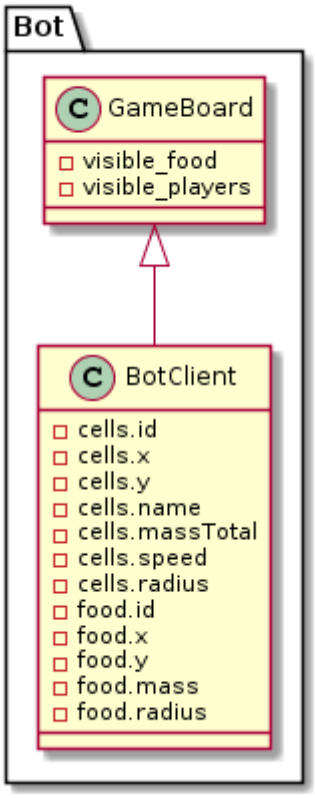

Рисунок 4 — Диаграмма классов. Бот

7 Алгоритм работы программы

Описание состояний продемонстрировано на рисунке 5.

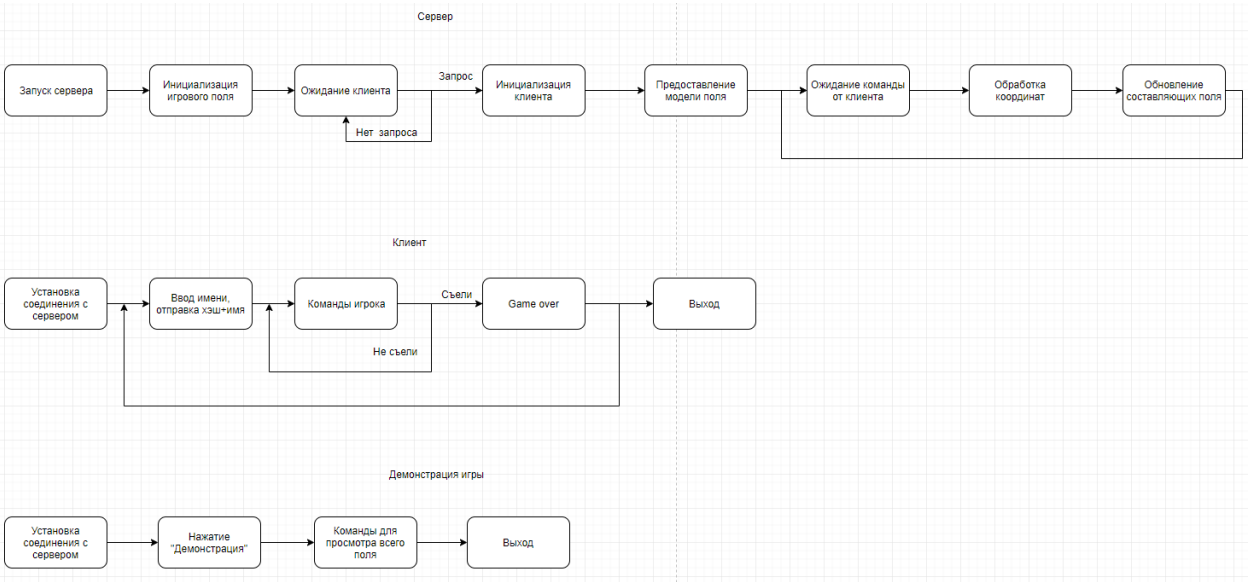

Рисунок 5 — Описание состояний

8 Инструкция по развертыванию системы

8.1 Установка программы и её компонентов

Скопировать репозиторий через ssh или https отсюда: https://gitlab.com/unidev/agario

Библиотека бота находится по адресу: https://gitlab.com/prettyGoo/agario-python-bot

8.2 Инсталлирование программного обеспечения для запуска в локальной системе

Для запуска программного обеспечения на локальной машине существует два варианта инициализации:

1. Для использования программы на локальной системе:

- Необходимо установить nodejs версии не ниже 8: https://nodejs.org/en/ ;
- Установить все зависимости проекта: npm i;
- Запустить сервер: npm start;
- Для запуска в браузере необходимо перейти по адресу: localhost:3000, 0.0.0.0:3000, 127.0.0.1:3000;
- Для остановки сервера необходимо нажать комбинацию клавиш ctrl+c;
- Для деинсталлирования необходимо удалить корневую папку проекта.

2. Использование Docker.

Если вы используете Linux или macOS и не имеете nodejs, установленный локально, то вы можете запустить проект внутри Docker. Для этого необходимо выполнить команду docker-compose up, перед этим убедитесь, что Docker and Docker compose установлены на вашей системе. Вероятность, что докер заработает под Windows достаточно мала (но вы можете попытаться).

Примечание:

Если будет в консоли ошибка, связанная с SIGNIN, то необходимо ввести:

lsof -i tcp:3000

kill -9 <YOUR_PID>

9 Описание бота

9.1 Системные требования

Python 3, pip

Примечание: под Windows установка должна быть осуществлена с правами администратора, чтобы корректно была настроена переменная окружения

9.2 Инсталляция и деинсталляция бота

- Инсталляция из терминала ОС
  pip install agario-bot
- Деинсталляция из терминала ОС
  pip uninstall agario-bot

9.3 API

BotClient(self, botname, speed_rate=1, wait_rate=0.1, host='localhost', port=3000, enable_logging=True) – конструктор, который возвращает объект бота. Принимает следующие аргументы (все аргументы, у которых нет значения по умолчанию, обязательны):

- botname (string) – имя бота;
- speed_rate (float) – время в секундах, в течение которого бот совершает одно свое движение. По умолчанию 1 секунда;
- wait_rate (float) – время, в течение которого бот после каждого запроса к серверу ждет от него ответа. По умолчания 0.1с, не рекомендуется менять.
- host (string) – имя хоста, на котором работает node.js сервер. По умолчанию это localhost.
- port (int) – порт, который слушает node.js сервер. По умолчанию 3000. Важно: это значение должно быть изменено, если сервер начинает слушать на другом порту; значение должно быть None, если сервер работает на настоящем хостинге;

- enable_logging (bool) – вывод логов о работе бота в консоль, по умолчанию True

move_left() – двигает бота влево в течение speed_rate

move_right() - двигает бота вправо в течение speed_rate

move_up() - двигает бота вверх в течение speed_rate

move_down() - двигает бота вниз в течение speed_rate

get_visible_surroundings() – позволяет получить то, что «видит» вокруг себя бот. Возвращает *словарь* с двумя ключами – food и cells.

Значение ключа ‘food’ – это «видимые» боту объекты еды. Это значение является списком из словарей, где каждый словарь – это объект еды, имеющий следующую структуру: id – уникальный идентификатор еды (int), x и y – ее координаты (float), radius – радиус еды (float), mass – масса еды (float).

Пример получения доступа к данным о первом объекте еды в словаре

```
surroundings = b.get_visible_surroundings()
surroundings['food'][0]['id']
surroundings['food'][0]['x']
surroundings['food'][0]['y']
surroundings['food'][0]['mass']
surroundings['food'][0]['radius']
```

Значение ключа ‘cells’ – это «видимые» боту игроки, включая самого бота. Это значение является списком из словарей, где каждый словарь – это объект игрока, имеющий следующую структуру: id – уникальный идентификатор игрока, совпадает с id сокета, через который установлено клиент-серверное соединение (string), x и y –координаты игрока (float), massTotal – масса игрока (float), name – имя игрока (string), cells – список из

одного словаря, из которого могут быть получены так же radius – радиус игрока (float), speed – скорость игрока

Пример получения доступа к данным о первом объекте игрока в словаре:

surroundings = b.get_visible_surroundings()

surroundings['cells'][0]['id']

surroundings['cells'][0]['x']

surroundings['cells'][0]['y']

surroundings['cells'][0]['name']

surroundings['cells'][0]['massTotal']

surroundings['cells'][0]['cells'][0]['speed']

surroundings['cells'][0]['cells'][0]['radius']

Своего бота можно определить в списке игроков следующим образом – для него в словаре отсутствует ключ id

9.4 Примеры

Ниже представлен пример простейшего бота, который двигается по периметру квадрата.

 # main.py

```
from agario_bot.bot import BotClient

b = BotClient('prettygoo', wait_rate=0.1)

surroundings = b.get_visible_surroundings()

while True:

    b.move_left()
```

```
surroundings = b.get_visible_surroundings()
print(surroundings['cells'])
print(surroundings['food'])

b.move_up()
surroundings = b.get_visible_surroundings()
print(surroundings['cells'])
print(surroundings['food'])

b.move_right()
surroundings = b.get_visible_surroundings()
print(surroundings['cells'])
print(surroundings['food'])
b.move_down()
surroundings = b.get_visible_surroundings()
print(surroundings['cells'])
print(surroundings['food'])
```

Также имеется пример готового, более сложного бота (https://gitlab.com/prettyGoo/agario-python-bot/blob/dev/agario_bot/examples/scary_bot.py), который убегает от всех

```
# main.py
from agario_bot.examples.scary_bot import run_scary_bot
run_scary_bot()
```

9.5 Запуск из терминала ОС

1. Перейти в папку, где находится файл (например, main.py) с ботом
2. Выполнить python main.py

9.6 Внесение изменений в библиотеку

- скачать исходный код **https://gitlab.com/prettyGoo/agario-python-bot**
- изменить setup.py (хотя бы название библиотеки)
- затем создать аккаунт на pypi.org, добавить логин и пароль в соотвествующий файл на системе для облегчения деплоя (о том, как это сделать, можно найти на сайте pypi)
- Внести ваши изменения в библиотеку
- выполнить команду make deploy.

10 Список выполненных работ

В таблице 1 представлен список выполненных работ каждого члена команды:

Таблица 1 — Список работ

Рубан А.Г.	Нагуслаев Н.Т.	Кирилов Н.Э.	Костюченко А.Е.
Документация	Документирование кода клиента	Документирование бота	Документирование кода сервера
Frontend		Backend	
Модальное окно с инструкцией	Отрисовка меню	Функции по работе с сокетами	Настройка сборщика проекта Gulp
Валидация имени	Отрисовка поля	Модуль GameController для управления поведением игровых элементов на поле	Функции по управлению игровых циклов
Отрисовка героя (прием данных)	Отрисовка сетки	Конфигурационный файл	Вынесение работы с сокетами в отдельный модуль
Принятие команд (↓→↑←awsd) и изменение направления	Отрисовка еды (прием данных)	Инициализация класса PlayerController	Инициализация класса UsersController

Продолжение таблицы 1

Общение с сервером во время игры (принятие и отправка координат)	Отправка данных (имени) и принятия начальных данных	Утилиты	
Изменение размера поля	Расчет массы, скорости и радиуса.		
Логика собственного бота	Логика собственного бота	API для бота	Тестирование