Проект API. Алиса. Разрабатываем навык

План урока

1. Введение

На этом уроке мы будем писать игру «Угадай город». Алиса показывает пользователю фотографию города, а он его отгадывает.

Важный момент: перед игрой пользователь должен представиться, а Алиса — его поприветствовать. Это добавит нашему навыку немного человечности.

Чтобы сделать такой навык, нам нужно научиться двум вещам:

1. Находить в сообщении от пользователя имя и город, если эти данные там присутствуют. Такие объекты принято называть **именованными сущностями**.
2. Отправлять пользователю картинку.

Итак, приступим.

2. Именованные сущности

На самом деле всё очень просто, поскольку разработчики Алисы уже о нас позаботились и сами выделяют такие сущности.

Под именованными сущностями подразумеваются:

- имена,
- фамилии,
- названия городов и так далее.

Когда пользователь в своём сообщении использует имя, фамилию или город, эти данные попадают в специальный раздел JSON'а, который отправляет нам Алиса.

Рассмотрим пример:


```
{
  "meta": {
    "locale": "ru-RU",
    "timezone": "Europe/Moscow",
    "client_id":
        "ru.yandex.searchplugin/5.80 (Samsung Galaxy; Android 4.4)",
    "interfaces": {
      "screen": { }
    }
  },
  "request": {
    "command": "закажи пиццу на улицу льва толстого, 16 на завтра",
    "original_utterance":
        "закажи пиццу на улицу льва толстого, 16 на завтра",
    "type": "SimpleUtterance",
    "markup": {
      "dangerous_context": true
    },
    "payload": {},
    "nlu": {
      "tokens": [
        "закажи",
        "пиццу",
        "на",
        "льва",
        "толстого",
```

```
      "16",
      "на",
      "завтра"
    ],
    "entities": [
      {
        "tokens": {
          "start": 2,
          "end": 6
        },
        "type": "YANDEX.GEO",
        "value": {
          "house_number": "16",
          "street": "льва толстого"
        }
      },
      {
        "tokens": {
          "start": 3,
          "end": 5
        },
        "type": "YANDEX.FIO",
        "value": {
          "first_name": "лев",
          "last_name": "толстой"
        }
      },
      {
        "tokens": {
          "start": 5,
          "end": 6
        },
        "type": "YANDEX.NUMBER",
        "value": 16
      },
      {
        "tokens": {
          "start": 6,
          "end": 8
        },
        "type": "YANDEX.DATETIME",
        "value": {
          "day": 1,
          "day_is_relative": true
        }
      }
    ]
```

```
    }
  },
  "session": {
    "new": true,
    "message_id": 4,
    "session_id": "2eac4854-fce721f3-b845abba-20d60",
    "skill_id": "3ad36498-f5rd-4079-a14b-788652932056",
    "user_id":
        "AC9WC3DF6FCE052E45A4566A48E6B7193774B84814CE49A922E163B8B29881DC"
  },
  "version": "1.0"
}
```


Обратите внимание на раздел **request** → **nlu** → **tokens**:


```
"tokens": [
        "закажи",
        "пиццу",
        "на",
        "льва",
        "толстого",
        "16",
        "на",
        "завтра"
      ]
```


Можно заметить, что это полученный от пользователя текст, разобранный на отдельные слова. Каждое слово при этом приводится к нижнему регистру.

Это очень удобно, если надо искать вхождения каких-то определённых слов. Например, «да» или «нет». Сегодня мы будем использовать этот раздел как раз для решения подобной задачи.

Второй раздел, на который стоит обратить внимание — это **entities** (то есть сущности):


```
"entities": [
        {
          "tokens": {
            "start": 2,
            "end": 6
          },
          "type": "YANDEX.GEO",
          "value": {
            "house_number": "16",
            "street": "льва толстого"
          }
```

```
        },
        {
          "tokens": {
            "start": 3,
            "end": 5
          },
          "type": "YANDEX.FIO",
          "value": {
            "first_name": "лев",
            "last_name": "толстой"
          }
        },
        {
          "tokens": {
            "start": 5,
            "end": 6
          },
          "type": "YANDEX.NUMBER",
          "value": 16
        },
        {
          "tokens": {
            "start": 6,
            "end": 8
          },
          "type": "YANDEX.DATETIME",
          "value": {
            "day": 1,
            "day_is_relative": true
          }
        }
      ]
```

Мы видим, что текст от пользователя подробно разобран и из него выделены все сущности от имени до даты. **entities** — это массив таких сущностей. Рассмотрим одну сущность и подробно распишем, из чего же она состоит:

```
{
"tokens": {
   "start": 3,
   "end": 5
},
   "type": "YANDEX.FIO",
   "value": {
      "first_name": "лев",
       "last_name": "толстой"
```

```
    }
}
```

tokens — обозначение начала и конца именованной сущности в массиве слов (**tokens**, что мы рассматривали выше). Нумерация слов в массиве начинается с 0.

start — первое слово именованной сущности.

end — последнее слово после именованной сущности.

type — тип именованной сущности. Возможные значения:

YANDEX.DATETIME — дата и время;

YANDEX.FIO — фамилия, имя и отчество;

YANDEX.GEO — местоположение (адрес или аэропорт);

YANDEX.NUMBER — число, целое или с плавающей точкой.

value — формальное описание именованной сущности (мы приведём ниже пример для типа YANDEX.FIO. Описание остальных типов ищите в документации).

first_name — имя.

last_name — фамилия.

Пример кода, получающий имя человека из JSON:

```
def get_first_name(req):
    # перебираем сущности
    for entity in req['request']['nlu']['entities']:
        # находим сущность с типом 'YANDEX.FIO'
        if entity['type'] == 'YANDEX.FIO':
            # Если есть сущность с ключом 'first_name',
            # то возвращаем её значение.
            # Во всех остальных случаях возвращаем None.
            return entity['value'].get('first_name', None)
```

3. Картинка в ответе

Теперь научимся показывать картинку в ответе. Например, для нашей игры нужно будет демонстрировать картинку города.

Для начала рассмотрим JSON ответа Алисы, в котором прикреплена картинка:

```
{
  "response": {
    "text": "Здравствуйте! Это мы, хороводоведы.",
    "tts": "Здравствуйте! Это мы, хоров+одо в+еды.",
    "card": {
      "type": "BigImage",
      "image_id": "1027858/46r960da47f60207e924",
      "title": "Заголовок для изображения",
      "description": "Описание изображения.",
      "button": {
        "text": "Надпись на кнопке",
        "url": "http://example.com/",
        "payload": {}
      }
    },
    "buttons": [
      {
        "title": "Надпись на кнопке",
        "payload": {},
        "url": "https://example.com/",
        "hide": true
      }
    ],
    "end_session": false
  },
  "session": {
    "session_id": "2eac4854-fce721f3-b845abba-20d60",
    "message_id": 4,
    "user_id":
        "AC9WC3DF6FCE052E45A4566A48E6B7193774B84814CE49A922E163B8B29881DC"
  },
  "version": "1.0"
}
```


В отличие от примера из предыдущего урока, в ответе появился раздел **card**. Он и содержит в себе картинку. Рассмотрим подробно его содержание:


```
"card": {
      "type": "BigImage",
      "image_id": "1027858/46r960da47f60207e924",
      "title": "Заголовок для изображения",
      "description": "Описание изображения.",
      "button": {
        "text": "Надпись на кнопке",
        "url": "http://example.com/",
```

```
        "payload": {}
}
```

Мы опишем только те поля, которые будем использовать. Про оставшиеся можно посмотреть в документации Алисы.

card — описание карточки — сообщения с поддержкой изображений. Если приложению удаётся отобразить карточку для пользователя, свойство **response.text** не используется.

type — тип карточки. Поддерживаемые значения:

BigImage — одно изображение;

ItemsList — галерея изображений (от 1 до 5).

*Требуемый формат ответа зависит от типа карточки. Мы будем использовать **BigImage***

image_id — идентификатор изображения, который возвращается в ответ на запрос загрузки.

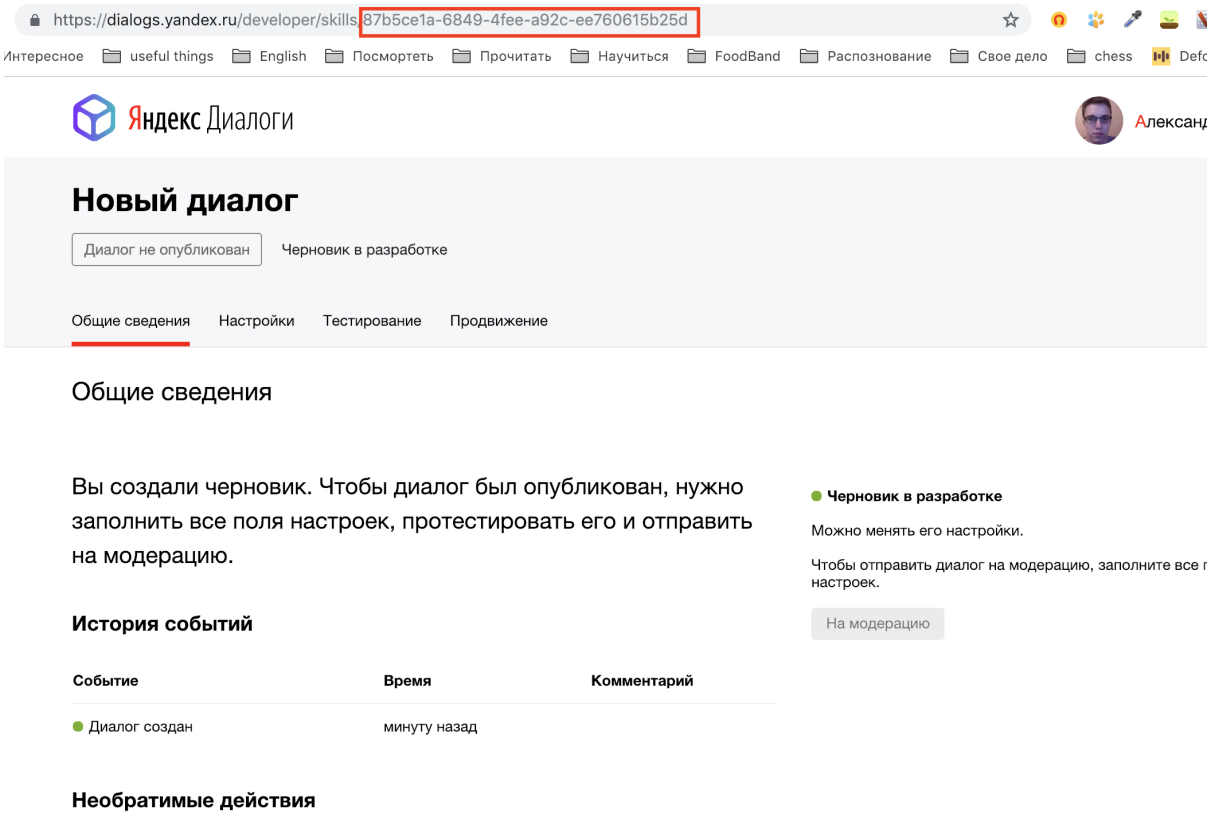
title — заголовок для изображения. Это на самом деле увидит пользователь.

Всё кажется очень понятным, но где же сама картинка? Мы видим лишь какой-то **image_id**. Откуда он взялся? Оказывается, чтобы использовать изображения в навыке, нужно хранить их на серверах Алисы. Как загрузить? Разумеется, через API.

Мы пойдём простым путем и используем для этого программу **Postman**, с которой работали на прошлом уроке для тестирования навыка, однако, конечно же, можно написать код на Python и использовать библиотеку **requests**.

Идём по пунктам:

1. Создадим новый навык. Зайдём в Платформу диалогов и нажмём **создать новый навык**.
2. Скопируем из URL и сохраним его идентификатор (ID):



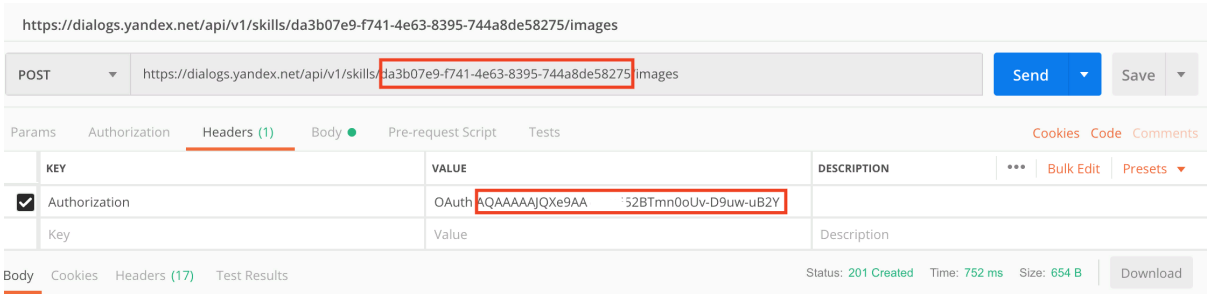
3. Получим **token** по ссылке. Сохраните его тоже. Он вам пригодится!

4. Теперь запускаем **Postman**.

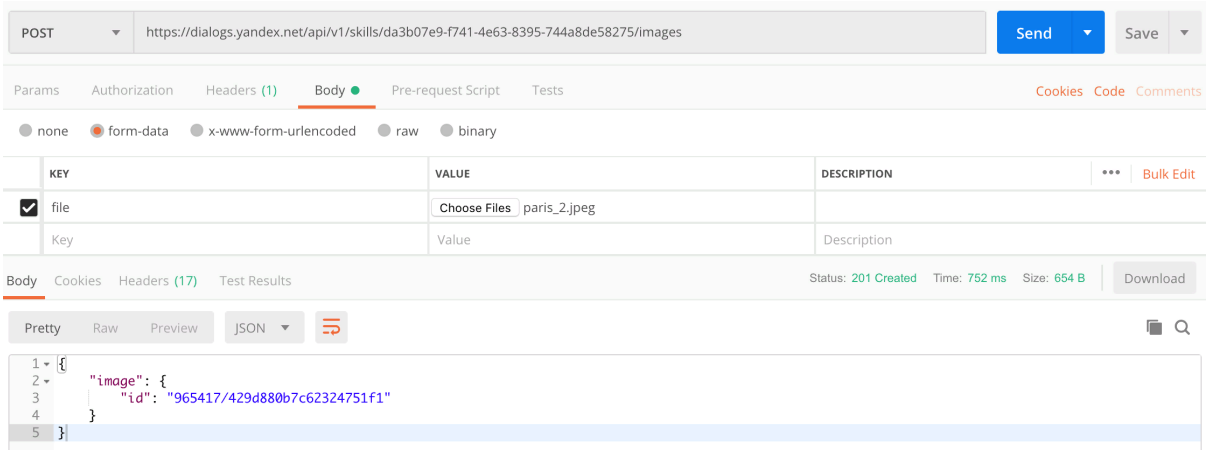
Заполняем поля на вкладке **Headers**.

Нужно добавить параметр с названием **Authorization**. Для этого просто в колонке **KEY** пишем Authorization, а в колонке **VALUE** — «OAuth Token» (OAuth пробел token, который вы получили).

В поле ссылки добавляем ссылку https://dialogs.yandex.net/api/v1/skills/id_навыка/images (заменить id_навыка на id из пункта 2).



Теперь на вкладке **Body** выбираем радио-кнопку **form-data** и добавляем файл. Надо в колонке **key** написать «file», в колонке **value** нажать «choose file» и выбрать файл с вашего компьютера. Далее нажимаем **Send**.



Если всё сделано правильно, то в ответе вы получите идентификатор картинки, который мы и будем потом использовать.

Загрузите 6 картинок для разных городов и запомните полученные id для каждого города, который будете использовать.

Мы собрали для вас картинки, которые надо загрузить в этот архив.

4. Навык, который знакомится

Напишем простой навык, который ведёт следующий диалог:

— Привет! Назови свое имя!

— Саша.

— Приятно познакомиться, Саша. Я — Алиса. Какой город хочешь увидеть?

— Москва (Нью-йорк, Париж).

— Этот город я знаю. *и показывает фото города*

Если вдруг пользователь не назвал имя, Алиса должна сказать:

— Не расслышала имя. Повтори, пожалуйста!

Если пользователь не назвал город или назвал город, для которого нет картинки, то Алиса должна сказать:

— Первый раз слышу об этом городе. Попробуй ещё разок!

Код:

```
from flask import Flask, request
import logging
import json
```

```
import random

app = Flask(__name__)

logging.basicConfig(level=logging.INFO)

# создаём словарь, в котором ключ — название города, а значение — массив,
# где перечислены id картинок, которые мы записали в прошлом пункте.
cities = {
    'москва': [
        '1540737/daa6e420d33102bf6947', '213044/7df73ae4cc715175059e'
    ],
    'нью-йорк': [
        '1652229/728d5c86707054d4745f', '1030494/aca7ed7acefde2606bdc'
    ],
    'париж': [
        '1652229/f77136c2364eb90a3ea8', '3450494/aca7ed7acefde22341bdc'
    ]
}

# создаём словарь, где для каждого пользователя мы будем хранить его имя
sessionStorage = {}


@app.route('/post', methods=['POST'])
def main():
    logging.info('Request: %r', request.json)
    response = {
        'session': request.json['session'],
        'version': request.json['version'],
        'response': {
            'end_session': False
        }
    }
    handle_dialog(response, request.json)
    logging.info('Response: %r', response)
    return json.dumps(response)


def handle_dialog(res, req):
    user_id = req['session']['user_id']

    # если пользователь новый, то просим его представиться.
    if req['session']['new']:
        res['response']['text'] = 'Привет! Назови свое имя!'
        # созда\м словарь в который в будущем положим имя пользователя
        sessionStorage[user_id] = {
```

```
            'first_name': None
        }
        return

    # если пользователь не новый, то попадаем сюда.
    # если поле имени пустое, то это говорит о том,
    # что пользователь ещё не представился.
    if sessionStorage[user_id]['first_name'] is None:
        # в последнем его сообщение ищем имя.
        first_name = get_first_name(req)
        # если не нашли, то сообщаем пользователю что не расслышали.
        if first_name is None:
            res['response']['text'] = \
                'Не расслышала имя. Повтори, пожалуйста!'
        # если нашли, то приветствуем пользователя.
        # И спрашиваем какой город он хочет увидеть.
        else:
            sessionStorage[user_id]['first_name'] = first_name
            res['response'][
                'text'] = 'Приятно познакомиться, ' + first_name.title()\
                          + '. Я - Алиса. Какой город хочешь увидеть?'
            # получаем варианты buttons из ключей нашего словаря cities
            res['response']['buttons'] = [
                {
                    'title': city.title(),
                    'hide': True
                } for city in cities
            ]
    # если мы знакомы с пользователем и он нам что-то написал,
    # то это говорит о том, что он уже говорит о городе, что хочет увидеть.
    else:
        # ищем город в сообщение от пользователя
        city = get_city(req)
        # если этот город среди известных нам,
        # то показываем его (выбираем одну из двух картинок случайно)
        if city in cities:
            res['response']['card'] = {}
            res['response']['card']['type'] = 'BigImage'
            res['response']['card']['title'] = 'Этот город я знаю.'
            res['response']['card']['image_id'] = random.choice(
                cities[city])
            res['response']['text'] = 'Я угадал!'
        # если не нашел, то отвечает пользователю
        # 'Первый раз слышу об этом городе.'
        else:
            res['response']['text'] = \
                'Первый раз слышу об этом городе. Попробуй еще разок!'
```

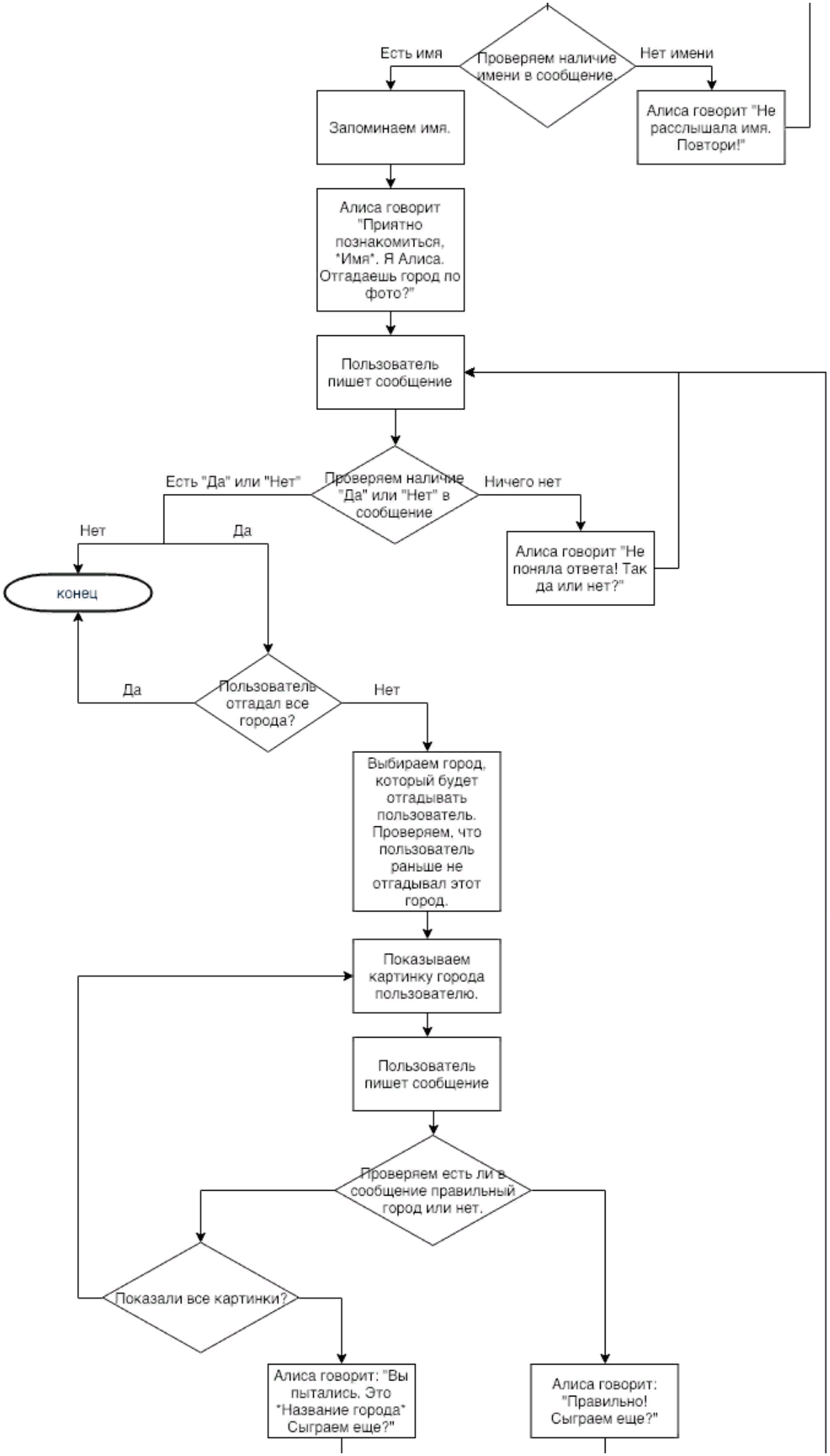
```
def get_city(req):
    # перебираем именованные сущности
    for entity in req['request']['nlu']['entities']:
        # если тип YANDEX.GEO то пытаемся получить город(city),
        # если нет то возвращаем None
        if entity['type'] == 'YANDEX.GEO':
            # возвращаем None, если не нашли сущности с типом YANDEX.GEO
            return entity['value'].get('city', None)


def get_first_name(req):
    # перебираем сущности
    for entity in req['request']['nlu']['entities']:
        # находим сущность с типом 'YANDEX.FIO'
        if entity['type'] == 'YANDEX.FIO':
            # Если есть сущность с ключом 'first_name',
            # то возвращаем ее значение.
            # Во всех остальных случаях возвращаем None.
            return entity['value'].get('first_name', None)


if __name__ == '__main__':
    app.run()
```

Загружаем этот код в PythonAnyWhere, заполняем данные в Алисе и отправляемся тестировать!

5. Игра

Доработаем предыдущую программу до конца.

Рассмотрим блок-схему, чтобы понять логику общения нашего навыка с пользователем.


Проверяем наличие имени в сообщение.
Есть имя
Нет имени
Запоминаем имя.
Алиса говорит "Не расслышала имя. Повтори!"
Алиса говорит "Приятно познакомиться, *Имя*. Я Алиса. Отгадаешь город по фото?"
Пользователь пишет сообщение
Проверяем наличие "Да" или "Нет" в сообщение
Есть "Да" или "Нет"
Ничего нет
Алиса говорит "Не поняла ответа! Так да или нет?"
Нет
Да
конец
Пользователь отгадал все города?
Да
Нет
Выбираем город, который будет отгадывать пользователь. Проверяем, что пользователь раньше не отгадывал этот город.
Показываем картинку города пользователю.
Пользователь пишет сообщение
Проверяем есть ли в сообщение правильный город или нет.
Показали все картинки?
Алиса говорит: "Вы пытались. Это *Название города* Сыграем еще?"
Алиса говорит: "Правильно! Сыграем еще?"

Пример кода приведён в файле.