Universidade do Minho
Departamento de Informática

Mestrado Integrado em Engenharia Informática

Inteligência Ambiente: Tecnologias e Aplicações

TP1 - Inteligência Ambiente e Sensorização: Monotonização Sonora

---

Grupo 5
a86617 Gonçalo Nogueira
pg42829 Francisco Borges
pg42846 Nuno Pereira
a82529 Carlos Afonso

Braga
Novembro, 2020

Conteúdo

Lista de Figuras

1 Introdução

Quando nos foi apresentado a proposta de utilizar um sensor, o nosso grupo decidiu fazer uma aplicação que trabalhasse com um microfone, a poluição sonora é algo que parece esquecida e com este trabalho tentamos sensibilizar as pessoas para a existência desta, pois não podemos pensar que reduzir a poluição ambiente é o suficiente para termos uma vida melhor.

A Organização Mundial da Saúde (OMS), diz-nos que a poluição sonora é um dos fatores ambientais que provoca mais problemas de saúde. Só na Europa, conforme a Agência Europeia do Meio Ambiente (AEMA), causa 16.600 mortes prematuras por ano e mais de 72.000 hospitalizações.

Assim, com este objetivo definido, começamos a pensar como poderíamos apresentar as nossas preocupações e definir um plano para tal.

2 Objetivos

- Implementar um sistema capaz de recolher e monitorizar leituras de sensores físicos e/ou Virtuais.
- Sensibilizar e motivar a criação de ambientes inteligentes.
- Trabalhar em domínios emergentes como *Internet of things* e/ou *smart cities*.
- Sensibilizar as pessoas sobre a poluição sonora.

3 Abordagem dos objetivos e domínios a tratar

Como o objetivo principal é chamar á atenção as pessoas para a poluição sonora decidimos fazer um programa que capture o som de um microfone.

Para sensibilizar as pessoas pretendemos mostrar o volume do microfone e se esta está num bom sítio ou não.

Outro objetivo proposto pela adenda ao trabalho é captar através de sensores as teclas pressionadas e criar métricas sobre esses valores.

4 Sensores utilizados

Os sensores que utilizamos são:

- Um sensor de áudio (microfone), obtendo os níveis de som através de funções integradas no *Intellij IDEA*.
- Um sensor de teclado captado através da biblioteca *java.awt.event* que implementa tanto um *listener* como um handler para o tratamento de quaisquer eventos que possam ocorrer no teclado de um utilizador.

5 Dados recolhidos e tratamento

5.1 Microfone

Para recolher os dados do microfone usamos a biblioteca *java.x.sound.sample* do java, depois de recolher uma sample do áudio (em Bits), usamos uma fórmula para passar os Bits para um valor, que correspondem aos níveis de decibéis. Quando obtemos estes, apresentamos a cada ciclo os resultados ao usuário e a cada 10 ciclos fazemos uma média (dos 10 ciclos) que também é apresentada como valor final.

Existindo a possibilidade de imprecisão dos valores obtidos, visto que os cálculos não são perfeitos tivemos o cuidado de fazer testes com vista a apresentar os valores o mais próximo da realidade possível, recolhendo valores de várias fontes sonoras comuns (como a buzina de um carro) e comparando com os valores médios observados em artigos.

5.2 Teclado

Relativamente ao teclado, á medida que o utilizador vai pressionando cada tecla na caixa de texto criada para o efeito, os handlers de cada evento recolhem essa informação e usando métodos existentes na biblioteca importada converte os códigos de cada tecla para a sua representação alfanumérica e teclas de ação (Shift, Alt, Enter, Backspace...).

Além disso, em simultâneo é criado um timestamp recorrendo á biblioteca *java.sql.timestamp* que marca o momento exato em que cada evento para o posterior cálculo de métricas.

Em conjunto com a marca temporal, o handler envia uma String referente á tecla pressionada para uma estrutura de dados (*HashMap*) criada com o objetivo de armazenar todas as teclas pressionadas, o número de vezes que cada uma foi pressionada bem como uma lista de marcas temporais em que foi pressionada.

Sendo assim, esta *HashMap* contém a *String* como chave que por sua vez redireciona para um par de *ints* (Número de vezes tecla pressionada) e *ArrayList* de *timestamps* (Hora em que tecla foi pressionada) também criado.

```
public class ParIntList {
    private int number;
    private ArrayList<Timestamp> tempo;
```

Figura 1: Par criado constituído por um *int* e um *ArrayList* de *timestamps*

```
public class ChavesPressionadas {

    private HashMap<String, ParIntList> pressionadas;
```

Figura 2: Estrura de dados criada

```java
public void addKeyPressed(String k, Timestamp t) {
    ParIntList a;
    if (pressionadas.containsKey(k)) {

        a = pressionadas.get(k);
        a.addParIntList(t);
    } else {
        ArrayList<Timestamp> b = new ArrayList<~>();
        b.add(t);
        a = new ParIntList( number: 1, b);
    }
    pressionadas.put(k, a);
}
```

Figura 3: Método para adicionar uma tecla pressionada na estrutura

6 Sistema desenvolvido, Arquitetura e Meios de comunicação

6.1 *Firebase*

Depois de calculados os valores guardamo-los para posterior consulta numa base de dados através do *firebase* que esses serviços de armazenamento e facilita uma pesquisa rápida e eficiente dos dados possibilitando a visualização e análise de um histórico com os valores médios de decibéis captados pela aplicação.

Todos os dados do microfone são guardados na *Firebase* usando a *API REST (Representational State Transfer)*. Esta API é uma arquitetura de software usada na criação de *web services*, possibilitando a manipulação e acesso do json na Firebase.

Para escrever na Firebase criamos a função WriteFireBase(), figura 4 linha 105. Depois na linha 108 criámos um objeto *url* com o id da nossa *Firebase*, que no nosso caso é soundreckoning-55568 e anexar .json no final, para podermos escrever no json. A seguir na linha 110 abrimos a conexão, depois enviamos uma solicitação *POST* na linha 114. Definimos uma cabeçalho da solicitação *"content-type"* como *"application / json"* para enviar o conteúdo da solicitação em formato JSON, na linha 116.

```
105  public void WriteFireBase( float soundVal) throws IOException {
106
107      //insert url
108      URL url = new URL ( spec: "https://soundreckoning-55568.firebaseio.com/.json");
109      //Open a Connection
110      HttpURLConnection con = (HttpURLConnection)url.openConnection();
111      //os dados da fire base podem escritos emitindo uma solicitação HTTP PUT
112      //POST create a new class
113      // para enviar uma solicitação POST  to send a POST request, temos que definir a propriedade do método de solicitação para POST
114      con.setRequestMethod("POST");
115      //temos de definir o cabeçalho da solicitação "content-type" como "application / json" para enviar o conteúdo da solicitação em formato JSON.
116      con.setRequestProperty("Content-Type", "application/json;");
117      con.setRequestProperty("Accept", "application/json");
118      //temos de por o setDoOutput a true para podermos escrever data no  output stream:
119      con.setDoOutput(true);
```

Figura 4: Solicitação *POST*

Escrevemos a nossa data num objeto json e enviamos a data para a Firebase na figura 5 da linha 130.

```
121         // converter informação para json
122         JSONObject sendData = new JSONObject();
123         sendData.put("sound",  String.valueOf(soundVal) );
124
125         String jsonInputString = sendData.toJSONString();
126         //send data to firebase
127         System.out.println("string do json=     "+ jsonInputString );
128         try(OutputStream os = con.getOutputStream()) {
129             byte[] input = jsonInputString.getBytes( charsetName: "utf-8");
130             os.write(input,  off: 0, input.length);
131         }
```

Figura 5: Escrever dados na *Firebase*

Para ler os dados criamos a função ReadFireBase2() e em vez de fazermos uma solicitação *POST* fazemos uma solicitação *GET* ao json figura 6 linha 53 e guardamos os dados necessários numa string e depois convertemos numa lista de floats, figura 7 da linha 78 até à 90.

```
50          URL url = new URL ( spec: "https://soundreckoning-55568.firebaseio.com/.json");
51          HttpURLConnection con = (HttpURLConnection)url.openConnection();
52          //os dados da fire base podem ser lidos emitindo uma solicitação HTTP GET
53          con.setRequestMethod("GET");
54          con.setRequestProperty("Content-Type", "application/json;");
55          con.setRequestProperty("Accept", "application/json");
56          con.connect();
```

Figura 6: Solicitação *GET*

```
            //retiramos apenas informação pertinente da string e convertemos em float
            List<Float> sList = new LinkedList<Float>();
            String sAux ="";
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < inline.length(); i++) {
                if( inline.charAt(i)=='d' && inline.charAt(i+1)=='"' && inline.charAt(i+2)==':' && inline.charAt(i+3)=='"' ){
                    i=i+4;
                    for (; i < inline.length(); i++){
                        if(inline.charAt(i) == '"') break;
                        sb.append(inline.charAt(i));
                        sAux = sb.toString();
                    }
                    sList.add(Float.parseFloat(sAux));
                    sAux="";
                    sb = new StringBuilder();

                }
                if(i+3> inline.length()) break;
            }
            System.out.println(sList);
        return  sList;
        }
    }
```

Figura 7: Conversão da string *json* para uma lista de floats

6.2 Interface

Relativamente ao sistema desenvolvido desenvolvemos um interface que permite uma interação intuitiva com o utilizador dando-lhe a opção de iniciar a captura do som, uma caixa de texto que permita a captura de eventos do teclado algo que é exclusivo apenas nessa área de texto, a opção de consultar métricas sobre o teclado e por fim a opção de consultar o histórico de toda a informação presente na firebase desde a sua criação.

Figura 8: Menu Principal da Aplicação

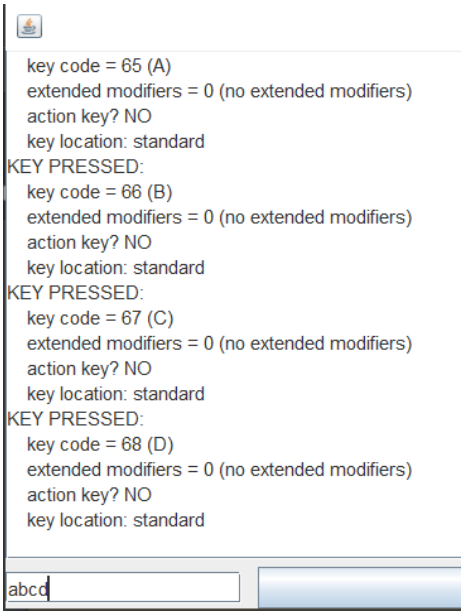

Figura 9: Logger das teclas pressionadas

De cada vez que uma tecla é pressionada é mostrado ao User o código da chave bem como algumas informações adicionais.

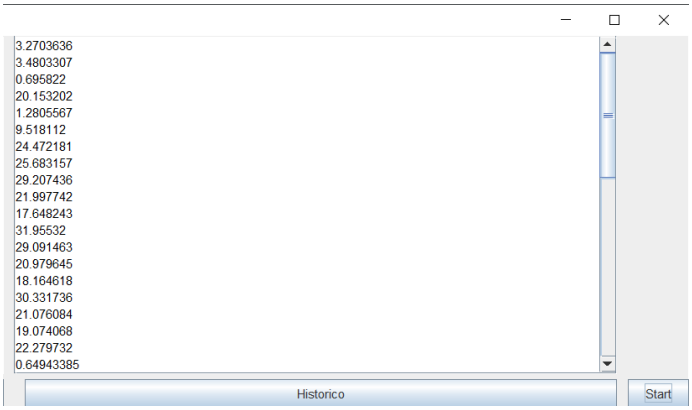

Figura 10: Valores de som capturados

Os valores captados do microfone aparecem na tela ao User desta forma e novas entradas são adicionadas em cima e os valores anteriores vão descendo.

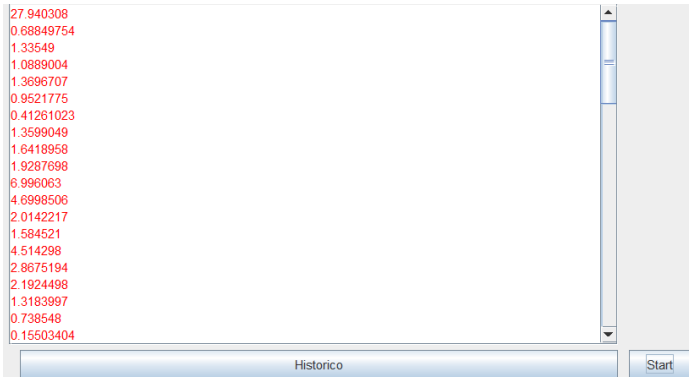

Figura 11: Valores elevados de som capturados

Quando os valores atingem um certo valor, a cor vermelha aparece de forma a sinalizar que naquela altura ultrapassou-se os decibéis considerados perigosos.

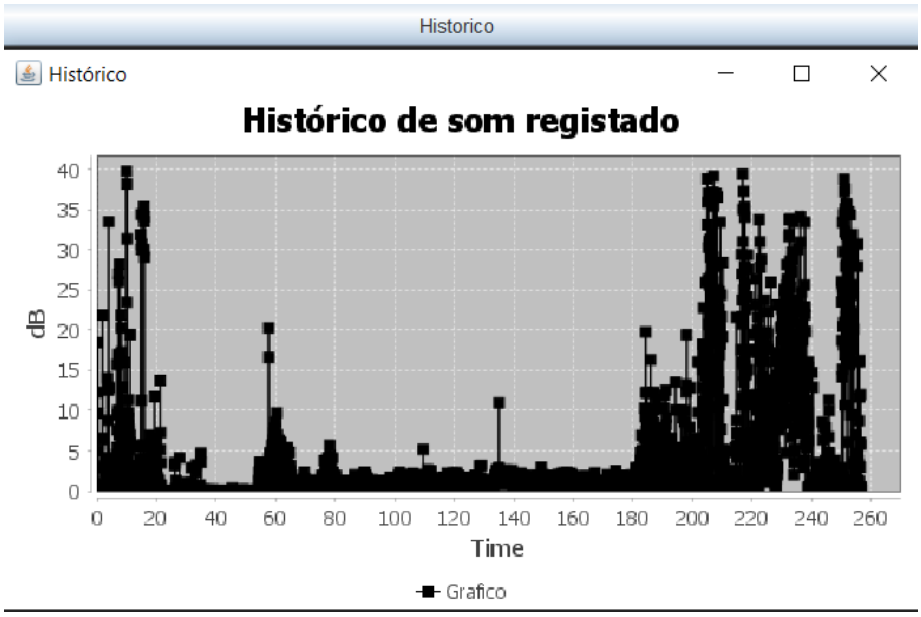

Figura 12: Histórico de todos os valores na firebase

Usando uma libraria do java conhecida por *JFreeChart* conseguimos criar um gráfico que mostra o padrão de sons capturados.

7 Descrição das métricas utilizadas sobre os dados recolhidos do teclado

Para o desenvolvimento de métricas sobre os dados recolhidos do teclado a estrutura de dados que criamos foi já concebida e implementada de forma a tornar eficiente e a facilitar os cálculos das mesmas.

Sendo assim foram criadas duas métricas, a primeira é a tecla mais vezes pressionada até ao instante em que a métrica é chamada e é calculada percorrendo todas as chaves da estrutura verificando qual o maior valor guardado, após toda a estrutura estar analisada devolve a *String* com a tecla obtida.

A segunda é teclas pressionadas por minuto, esta métrica foi a mais complicada de calcular entre as duas. O método encontrado para o cálculo primeiramente viaja por toda a *HashMap* e coleta todas marcas temporais presentes para um *ArrayList*, em seguida, utiliza o tempo em que a métrica foi chamada e o primeiro valor na lista preenchida anteriormente e calcula a sua diferença, no entanto como os valores estão em milissegundos foi preciso primeiro converter esse valor para minutos e finalmente dividindo o tamanho da lista (corresponde ao numero total de teclas pressionadas) pelos minutos chegando assim ao valor final de teclas pressionadas por minuto.

```
public String KeyPressedMore() {
    int count = -1;
    String k = null;
    for (Map.Entry mapElement : pressionadas.entrySet()) {
        ParIntList a = ((ParIntList) mapElement.getValue());
        int c = a.getNumber();
        if (c > count) {
            count = c;
            k = (String) mapElement.getKey();
        }
    }
    return k;
}
```

Figura 13: Método que calcula a tecla mais vezes pressionada

```java
public int KeysPerMinute(Timestamp t) {
    ArrayList<Timestamp> aux;
    ArrayList<Timestamp> both = new ArrayList<>( initialCapacity: 10000);

    for (Map.Entry mapElement : pressionadas.entrySet()) {
        ParIntList a = ((ParIntList) mapElement.getValue());
        aux = a.getTempo();
        both.addAll(aux);
    }
    long l1=t.getTime();
    long l2=both.get(0).getTime();
    long l3=l1-l2;
    int minutes = (int) ((l3 / (1000*60)) % 60);
    int seconds = (int) (l3 / 1000) % 60 ;
    float z = (float) (minutes + (Math.abs(seconds) / Math.pow(10, Math.floor(Math.log10(Math.abs(seconds)) + 1))));
    if(z!=0) {
        return (int) (both.size() / z);
    }else{
        return 0;
    }
}
```

Figura 14: Método que calcula quantas teclas são pressionadas em média por minuto

8 Sumário dos resultados e análise critica

Relativamente aos resultados dos microfone não foram completamente exatos, apercebemo-nos que algo nos cálculos não está perfeito, embora os cálculos matemáticos usados teoricamente estejam corretos, reparamos no entanto que o nosso ambiente normal de casa não está perto da considerada "poluição sonoro", o que é excelente pois mostra uma boa responsabilidade da nossa parte, no entanto, quando usado enquanto se ouve música em condições não preparadas (correr o programa sem avisar os restantes membros da família) notamos que os valores captados podem por vezes se aproximar dos valores de poluição sonora.

Para o teclado consideramos que os valores são de forma geral obtidos de forma correta e estamos bastante satisfeitos com a estrutura criada para o armazenamento da informação pertinente e as métricas apesar de poucas estão bem calculadas e corretamente apresentadas o que nos permitiu por exemplo comparar a velocidade de escrita de cada membro do grupo.

De forma geral, a parte em que tivemos mais dificuldade e foi a que talvez ficou um pouco menos bem feita foi a parte do front-end devido á falta de experiência de todos os elementos com o javax.swing.

9 Sugestões de melhoria

Para melhorar a nossa aplicação temos de melhorar os nossos cálculos matemáticos para assim apresentar um resulta que seja o mais próximo possível da realidade, integrar a capacidade de recolher dados da localização do utilizador bem como a implementação de uma base de dados universal dando assim a hipótese de ter dados de vários locais, possibilitando a emissão de alertas para evitar certas zonas com bastante poluição sonora.

Relativamente ao teclado, o ponto que seria possível melhorar seria porventura a construção de mais métricas sobre os dados como por exemplo se se pressiona mais teclas do lado direito ou do lado esquerdo. Uma melhoria também fácil de apontar seria a captura de dados através de sensores sobre os cliques e arrastos do rato, algo que consideramos implementar no nosso projeto, mas mais tarde acabamos por decidir não o fazer devido a falta de tempo.

10 Conclusão

De facto, ainda falta algum caminho a percorrer para a aplicação atingir o estado ideal para o lançamento ao público, no entanto, temos um bom protótipo para verificar os níveis de ruído que pode ser usada para sensibilizar as pessoas.

Sendo assim, cumprimos com a maioria dos nossos objetivos iniciais como a captura e apresentação dos valores obtidos quer do microfone quer do rato e algumas métricas.

De salientar, que mesmo com alguns elementos do grupo com uma familiarização ténue acerca da linguagem de programação utilizada, a ajuda dos restantes elementos fortaleceu não só o conhecimento de ambas as partes como também fomentou o trabalho em equipa e entreajuda.

11 Bibliografia

https://www.iberdrola.com/meio-ambiente/o-que-e-poluicao-sonora-causas-consequencias-solucoes