Assignment 4 - Design and Benchmarking

FIT3140 Semester 1, 2017

GROUP ASSIGNMENT

Due: Sunday 7th May, 23:00
Worth: 5% of final mark.

In this assignment, you will work in pairs to develop some design documents for the Morse code application, and conduct another spike to determine the best algorithm for implementing an aspect.

What is Morse Code?[1]

Invented by Samuel Morse in 1836, Morse Code is a method for sending and receiving text messages using short and long beeps. Conventionally, a short beep is called a **dot** and a long one is a **dash** (also known, respectively, as a **dit** and a **dah**). Every letter of the alphabet has a unique sequence of dots and dashes.
If you look at the chart below, the letter **A** is beep beeeeeep and the letter **B** is beeeeeep beep beep beep.

A •–	J •–––	S •••	1 •———–
B –•••	K –•–	T –	2 ••–––
C –•–•	L •–••	U ••–	3 •••––
D –••	M ––	V •••–	4 ••••–
E •	N –•	W •––	5 •••••
F ••–•	O –––	X –••–	6 –••••
G ––•	P •––•	Y –•––	7 ––•••
H ••••	Q ––•–	Z ––••	8 –––••
I ••	R •–•	0 –––––	9 ––––•

[1] https://www.raspberrypi.org/learning/morse-code-virtual-radio/worksheet/

- All timings are defined as multiples of one dot length
- A dash is three times the length of a dot
- Each dot or dash has a short gap of silence after it (usually 1 dot length)
- Letters in a word have a slightly longer gap of silence between them (usually 3 dot lengths)
- Words have an even longer gap of silence between them (usually 7 dot lengths)

In our application, dash and dot are the long and short motions respectively

The Morse code application is responsible for:

1. Listen to motion sensor data and classify it into a sequence of long and short motions.
2. Using the Morse code table, interpret the data into letters.
3. The client has to show those letters as a message.

   The Morse code app is quite similar to the Intruder counter app except you compare the input data (motion sequence) with 36 codes instead of one.

Note: Morse code algorithm can be implemented at the client or the server side.
Note: you don't need to implement Morse code app in this assignment.

Your Tasks

You now have some idea about the following things:

- what you're expected to do
- the basics of the toolchain that you're working with..

Based on this knowledge, your next tasks are to come up with a high-level, architectural design for your Morse code interpreter app, and do some experimenting to determine an aspect of the detailed design for your app.

1. Design Documentation - a *sequence and activity* Diagrams

You should now provide some actual design for your system.

Draw a *sequence and activity diagrams* for the proposed Morse code interpreter app.

By a " *sequence and activity diagrams* ", I mean the diagrams should show:

- the objects (entities) of the app,
- the interactions of these objects over time represented as messages

References:

- Lecture notes Week 5
- http://www.sparxsystems.com.au/resources/uml2_tutorial/uml2_sequencediagram.html
- https://www.ibm.com/developerworks/rational/library/3101.html

Tools:

- http://www.umlet.com/
- http://sequencediagram.org/
- https://eclipse.org/papyrus/
- https://www.softwareideas.net/en/features

As well as the diagram, you should write some text explaining each object and its messages.

2. Benchmarking spike

The Morse code application needs to have a data structure to store the Morse codes and their letters. Let's just call it **MorseCodeTable**. Different data structures and comparison algorithms can be used to implement this application. Two methods are proposed, declaring the code as a string and as an array of characters. I'd like to know whether the way I implement them makes any significant performance difference.

Method 1 would be to store the code of each letter and the received data (received from the motion sensor) as strings and the comparison will be between two strings.
Node.js speaking, the data structure could be implemented as:

```
var moreCodeTable = [{
   letter: 'a',
   code  : '1001'
}, {
   letter: 'b',
   code  : '0011'
}, {
   letter: 'c',
   code  : '0101'
}, {
   letter: 'd',
   code  : '1110'
}, {
   letter: 'e',
```

```
    code  : '1111'
}];
```

and the received data can be as simple as

```
inputData='1001';
```

Your Task: compute the total time that is needed to test (interpret) all the input strings (i.e. motion data).

Steps

1. Implement the above data structure (five letters are enough)
2. Synthetically, generate an array of 50000 of motion data (let's call it myData) as shown below:

   ['1001',
   '0000',
   '1110',
   '0000',
   '0100',
   '1010',
   …
   …
   …]

3. For each item in myData (i.e. for each input string), loop through the Morse code table and print out the letter if the item matches the code; print null otherwise.
4. Calculate the execution time of point (3)

Method 2 would be to implement the codes and the received data as arrays of characters.
Node.js speaking, the data structure could be implemented as:

```
var morseCodeTableArray = [{
    letter: 'a',
    code  : ['1', '0', '0', '1']
}, {
    letter: 'b',
    code  : ['0', '0', '1', '1']
}, {
    letter: 'c',
    code  : ['0', '1', '0', '1']
}, {
    letter: 'd',
    code  : ['1', '1', '1', '0']
}, {
```

```
  letter: 'e',
  code  : ['1', '1', '1', '1']
}];
```

and the received data could be:

```
inputData =['1','0','0','1'];
```

Your Task: compute the total time that is needed to test (interpret) all the input strings (i.e. motion data).
Steps:

1. Implement the above data structure (five letters are enough)
2. Synthetically, generate an array of 50000 of motion data (let's call it myData) as shown below:

   [['1', '1', '1', '1'],
   ['1', '0', '0', '1'],
   ['1', '0', '0', '1'],
   ['1', '0', '0', '1'],
   ['0', '1', '0', '0'],
   ['1', '0', '0', '1'],
   ['0', '0', '0', '0'],
   ['0', '1', '0', '1'],
   …..
   ……
   ……
   ]

3. For each item in myData (i.e. for each input string), loop through the Morse code table and print out the letter if the item matches the code; print null otherwise.
4. Calculate the execution time of point (3)

Note:

- You should do planning for this spike but you don't have to submit a written spike plan.

3. Benchmarking report

Write a report about your benchmarking, clearly explaining:

- how you measured the performance of the two options, in sufficient detail that somebody else could repeat your experiments and get the same results
- The results of your benchmarking. You can use tables or graphs if you think they are helpful. Try with different number of motion data. In other words, the size of myData can be 10000, 50000, and 100000.

- your conclusions about the most appropriate data structures to use in the app, based on your benchmarking.

Your report should be clearly understandable to somebody who is not intimately familiar with your particular code base - such as your boss's boss, if you were doing this at work.

Your report should be a maximum of five pages of text, but I will allow a couple of extra pages for images. There is no penalty for writing less than the maximum amount, providing you have fully addressed all the points listed above: do not pad your report.

Task planning and time tracking

You must split up the tasks for this assignment, and keep track of how they are going (whether they are not started, incomplete, or complete. You can use Trello (https://trello.com/) for this, or a Google Doc spreadsheet, or some other form.

You should also keep a time log recording when, and for how long, each partner works on each task. If you use Trello, you can use the Punchtime extension to record it.

Either way, you must create and submit a record of how long is spent by each partner, on each task.

If you use Punchtime, you must print your work log to PDF and submit it.

Special consideration

If a team member faces exceptional circumstances (serious illness or injury, family emergency etc) that prevent them completing the assignment, they may apply for special consideration according to the policy and procedure on the Faculty website:

http://www.infotech.monash.edu.au/resources/student/equity/special-consideration.html

Generally, for this assignment if one individual is unable to fully contribute, the scope of work will be reduced, rather than extensions granted.

Working as a pair

You are expected to work as a pair on this and subsequent assignments, and contribute roughly equal amounts of work to each, and very close to equal amounts over the course of semester. This should be tracked using Trello and a time log. In most cases, if this is followed, students will receive equal marks on coursework.