Assignment 5 - Iterations

FIT3140 Semester 1, 2017

PAIR ASSIGNMENT

Worth: 25% of final mark.

In this assignment, you will begin to implement "version 0.1" of the Morse Code Interpreter app, using the toolchain you chose earlier, and using Agile practices.

You will complete two iterations and each iteration has its submission link. You will demonstrate your functionality in the Week 12 lab by presenting your work using PowerPoint slides or any format you prefer.

In Assignment 5, you should submit **two iterations** and one **final report**.

Due dates:	
Iteration 1	: Tuesday, May 16th @ 23:55
Iteration 2 and Final Report	: Monday, May 22nd @ 23:55

In this assignment, you need to engineer this properly, including following a design, testing thoroughly, and having a codebase ready for the next set of user stories in the second iteration.

You will also need to do task tracking and time tracking. Task tracking will be done using a Trello board.

What is Morse Code?

Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment. It is named for Samuel F. B. Morse, an inventor of the telegraph. (Wikipedia)

Morse code elements:

- short mark, dot or "dit" (**.**): "dot duration" is one time unit long. (SHORT SIGNAL)
- longer mark, dash or "dah" (**▬**): three time units long (LONG SIGNAL)
- short gap (between letters): three time units long (□□□)
- medium gap (between words): seven time units long (□□□□□□□)

Examples:

-
  - Message : AB
  - Code : SL□□□LSSS
-
  - Message : SOS
  - Code : SSS□□□LLL□□□SSS
-
  - Message : HI CS
  - Code : SSSS□□□SS□□□□□□□LSLS□□□SSS

Where □ can be 1000ms
Note: You can adjust the value of □ based on your experiments.

Data Structure (Morse Code Table)

Different data structures (DS) can be used to implement Morse code table and one of them is the dictionary, which is provided below. However, you can implement your own DS.

```
var morseTable = {
    'SL'   : 'A',
    'LSSS' : 'B',
    'LSLS' : 'C',
    'LSS'  : 'D',
    'S'    : 'E',
    'SSLS' : 'F',
    'LLS'  : 'G',
    'SSSS' : 'H',
    'SS'   : 'I',
    'SLLL' : 'J',
    'LSL'  : 'K',
    'SLSS' : 'L',
    'LL'   : 'M',
    'LS'   : 'N',
    'LLL'  : 'O',
    'SLLS' : 'P',
    'LLSL' : 'Q',
    'SLS'  : 'R',
    'SSS'  : 'S',
    'L'    : 'T',
    'SSL'  : 'U',
    'SSSL' : 'V',
    'SLL'  : 'W',
    'LSSL' : 'X',
    'LSLL' : 'Y',
    'LLSS' : 'Z'
};
```

Morse Decoder Pseudocode

A very high-level description for Morse code decoder is given in the below pseudocode. It shows how to check the words and letters gaps.

```
LET word_gap=7000ms
LET char_gap=3000ms;
LET Δt = Current_time - previous_signal_time


WHILE (there are more signals)
        IF(Δt>=word gap)  // we got a word
            TRANSLATE the current Morse code into a letter
            ADD a space to message
        ELSE IF(Δt>=char_gap) // we got a new character
            TRANSLATE the current Morse code into a letter
        END IF

        ADD signal to current Morse code
END WHILE
```

Note: You can adjust word_gap and char_gap values based on your experiments.

Iteration 1

User stories

The theme for the first of the two iterations is "basic functionality".

The user stories for your first iteration, in order of priority, are as follows:

- As an app user, I want to enter data through a motion sensor.
- As an app user, I want the server to listen to all motion sensor's signals.
- As an app user, I want to read the detected motion signals from the server console.
- **As an app user, I want the server to determine whether the incoming message is long or short.**
- As an app user, I want the client side to receive the motion sensor's signals.
- **As an app user, I want to decode the motion sensor messages to be decoded into English letters based on Morse code coding table.**
- As an app user, I want the client side to print out the decoded messages (i.e. the English letters).

Note: that user interface polish is *not* necessary for the first iteration - just as long as it works, it's fine, it can be polished later

Note: the bold user stories have to be tested.

What you need to do for iteration 1

- Split the user stories up further as appropriate.
- Put the split-up stories onto your Trello board, prioritize them, estimate their size (using T-shirt sizing of small/medium/large is fine), and mark them off once done (including recording time taken).

- Implement the iteration's stories.

Testing User Stories

<u>You are required to</u> perform unit testing for at least two user stories (in bold) within your system. For each user story, you need to develop at least two different test cases (i.e. scenarios).
You might use Mocha and Chai testing framework what are given in week 9. However, feel free to use the one you prefer and fits your requirements.

Tests should include "expected" or "normal" behaviour as well as "error" cases.
Note: In Error cases, we need to test if wrong input or output are acceptable.

You should have a test plan which documents *exactly* how each test case is to be conducted. This should describe:

- A name for the test.
- A brief rationale for the test.
- Any setup that needs to be done, with enough precision and detail that somebody *other than the developers could repeat the same test.* If the setup procedure is the same for several tests, you can create a "test setup procedure", give it a name, and refer to it by name in your test cases.
- The test procedure, including inputs - be precise and explicit.
- The expected results - again, be precise and explicit!

Stories should be integrated

You will have *one* package containing all the functionality completed to date. Submitting something that does not run at all will be *heavily* penalized (normally, a failing grade for the assignment, whatever its other merits).

Stories should be demonstrable

You will demonstrate your work to your demonstrator in the lab 12 after the submission.

Time Tracking

As well as keeping the project backlog up to date, each team member should again track time, either using Punchtime or a Google Doc spreadsheet, and submitting their time log as part of their submission.

It will be *much* better to do only some of the items, if you do them properly, rather than doing a half-baked incomplete attempt at all of them.

You may ask....what if you get stuck on a horrible bug? Try and make your tasks *small* and as independent as possible, so you can still complete as much as possible. Furthermore, build and test incrementally, so that you can still have something to show for your efforts even if some bits don't work.

Reuse of source code

You can reuse code from previous assignments, but it needs to be modified to be production-quality rather than experimental.

Code quality

You will be assessed on the quality of your code.

This will include:

- Appropriate separation of functionality into modules (whether formal or informal). Maximize cohesion!
- Sensible interfaces for those modules (minimize coupling).
- Sensible choices of data structures and algorithms.
- Error handling.
- Sensible identifier naming.
- Appropriate code Logging. You can use 'Winston' package, or the one you prefer. (in worst case: console.log)
- Appropriate code documentation. Standards will depend on your toolchain, but as a minimum:
    - Each source code file (module) needs header documentation describing its purpose.
    - Each module needs header documentation describing its purpose.
    - Each method needs to be documented to describe its purpose, its arguments, its return values, any preconditions, postconditions and any exceptions that might be thrown.

Final Report

You should only commence working on this when your submission is close to completion. It should be short and punchy! It should be around three A4 pages of 12-point text, with (as a rough guideline) one devoted to each of the three topics below.

Topic 1. Agile software development practices

Review the lectures on Agile software development, and have a look at the Scrum Primer Document (Can be found on Moodle/Assessments / Assignment 5). Compare the way you approached your project to the methods described in these resources.
Examples on agile practices:

- Planning Game
- Design
- Test-Driven Development
- Refactoring
- Continuous Integration
- Scrum daily meetings

Points to consider:

- Which Agile practices did you adopt for your development? How well did they work?
- What features of the academic environment have had the biggest impact on your software development practices?

- Which Agile practices did you *not* adopt? Why not?
- Thinking back over your project, can you think of anything you *should've* done differently?

Topic 2. Working in teams

The constraints on student teams are not the same as the constraints on teams in industry, but teams in each area are expected to face similar challenges. Consider how well you and your partner (or partners) worked together.

Points to consider:

- How did you coordinate with your partner?
- How did you manage communication with your partner?
- How did you allocate tasks? Was this arrangement satisfactory to both of you?
- How would these practices hold up if you were programming in a small company? What would you need to change?

Topic 3. Design

In many other units, a program's architecture comes a distant second to its functionality – a working program is considered much more important than a well-engineered one. But the way a program is put together can have significant ramifications for its maintainability later on, so for programs that are expected to remain in use for a while, the equation is different.

Points to consider:

- If the client wanted to support variations on the Morse Code App - how difficult would this be to add to the design?
- How easy is it to find and fix bugs in your codebase? Can you think of any way that the design could be changed to make errors easier to deal with?
- How happy are you with the visual design of your App? Would you need to change anything in order for it to be suitable for younger children? How easy would it be to add new controls to your user interface while retaining (or improving) its current level of usability?

Can I write more than this?

Of course you can. The "points to consider" are only a guideline to help come up with a page on each topic, and a hint at the sort of things I'm interested in.

Things to submit

After each iteration, you will need to make a submission to Moodle. Your final mark will depend on the contents of *both* submissions, not just the second one!

Each group submission should include:

- Source code of the iteration.
- Documentation. Your project should have a "doc" subdirectory which contains (at a minimum):
    - Any supporting material for the Project Backlog (e.g., diagrams).
    - A README file (in plain text or HTML format) which explains:
        - A very brief overview of the product and project.
        - What has been implemented in this release.
        - A list of known bugs and (non-obvious) limitations, where applicable.
- A copy of your Trello board with tasks - save it to PDF.
- Your time tracking (saved to PDF).
- The test plan(s).

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 should be reduced, rather than extensions granted.

Late submissions

Students must contact the Lecturer if they do not submit the assignment by the deadline. Generally, it will be a much better idea to reduce the scope of the assignment than delay submission.