SE 3XA3: Test Report
DNA Says

Team 10, DNA
Kareem Abdel Mesih - abdelk2
John-Paul Dakran - dakranj
Shady Nessim - nessimss

December 5, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
01/12/16	1.0	Addition of Comp. to Existing Imp.
01/12/16	1.1	Addition of Automated Testing
02/12/16	1.2	Addition of Unit Testing
02/12/16	1.3	Addition of Changes Due To Testing
02/12/16	2.0	Revision Of The Sections Above

1 Functional Requirements Evaluation

1.1 User Input

1.2 Navigation

1.3 Display

1.4 Functionality

1.5 Parallelism

2 Nonfunctional Requirements Evaluation

2.1 Usability

2.2 Performance

2.3 etc.

3 Comparison to Existing Implementation

The core functionality of DNA Says relies on the original game Simon Says. However, the implementation is vastly different to preserve originality and creativity. The main principle of having the computer play a specific pattern and having the user repeat it is evident in both implementations. It is how the pattern is presented that is modified in DNA Says, rather than having one mode that contains only four buttons. This is what makes DNA Says unique.

DNA Says contains three modes, compared to one, which are Kareem Says, JP Says and Shady Says. In Kareem Says, the user is presented a piano keyboard in which the pattern will be played on, both by the computer and the user. Of course, the colors of the keys match those of a real piano, and each key is assigned one unique tone. When a key is clicked, if it is correct then it will light up in red to indicate the selection and its tone will play. Otherwise, the whole window will flash to indicate the entry of a wrong key and losing the game and a harmonic combination of tones will play. This wil then reset back to level one, and the score to zero.

As for JP Says, there are nine buttons of different colors, arranged as a three-by-three grid in the middle of the screen. Each button is assigned a unique tone that is played every time the button is pressed. To advance with this mode, the user must repeat the pattern that is played by the computer. The pattern is incremented by one element for every level. The new element that is added is completely random every time. Winning and losing conditions follow those of Kareem Says.

Finally, Shady Says. There are four buttons of different colors, arranged as a two-by-two grid in the middle of the screen. For every time that any button is pressed, a completely random tone (out of four in total) will play. To advance with this mode, the user must repeat the pattern that is played by the computer. The pattern is incremented by one element for every level. The new element that is added is completely random every time. Winning and losing conditions follow those of Kareem Says.

4 Unit Testing

The program was split up into modules, each to be tested separately. This unit testing ensured that changes made to the game did not break the basic functionality of the game. The testing conducted is detailed below.

- **Test #1**
  - Test: main
  - Type: manual
  - Description: checks if the program is initialized as required
  - Initial State: program not yet started
  - Input: the program is started
  - Output: Pygame is initialized, fonts and sounds are loaded, and required variables are made global
  - Result: PASS!
- **Test #2**
  - Test: setup

  - Type: automated
  - Description: checks if the screen is refreshed and setup again with every transition
  - Initial State: either on the menu or in any mode
  - Input: if on the menu, then a mode is selected. If in any mode, then the back button is clicked
  - Output: transition to the desired screen is successful, without any left-overs from the previous screen
  - Result: PASS!

- **Test #3**
  - Test: update
  - Type: automated
  - Description: checks if the screen is updated accordingly inside any mode
  - Initial State: in any mode, no changes appear on the screen yet
  - Input: The computer or user plays an element from the pattern
  - Output: the screen updates accordingly and displays that change
  - Result: PASS!

- **Test #4**
  - Test: showInst
  - Type: manual
  - Description: checks if instructions are displayed on screen at the desired location
  - Initial State: the instruction are not shown on the screen
  - Input: showInst() is invoked
  - Output: the instructions appear at the bottom left corner
  - Result: PASS!

- **Test #5**

- Test: showGoBack
- Type: manual
- Description: checks if the back button text is displayed at the desired location
- Initial State: the back button text is not shown on the screen
- Input: showGoBack() is invoked
- Output: the back button text appears at the top left corner
- Result: PASS!

- **Test #6**
  - Test: showScore
  - Type:manual
  - Description: checks if the score is displayed at the desired location
  - Initial State: the score is not shown on the screen
  - Input: showScore() is invoked
  - Output: the score appears at the top right corner
  - Result: PASS!

- **Test #7**
  - Test: flashKeyAnimation
  - Type: automated
  - Description: checks if key animations are error proof with regards to wrong input
  - Initial State: game launched in any mode
  - Input: the computer or user plays an element from the pattern
  - Output: button visuals and associated sound, no output for inputs that are out of range
  - Result: PASS!

- **Test #8**
  - Test: drawKeys

- Type: automated
- Description: checks if the correct buttons are drawn at their correct locations depending on the mode selected
- Initial State: on the menu
- Input: any mode is selected
- Output: the correct buttons appear on the screen along with the back button, at their correct locations
- Result: PASS!

- **Test #9**
  - Test: changeBackgroundAnimation
  - Type: automated
  - Description: checks the changing of the background color after every level change
  - Initial State: game launches in any mode
  - Input: the user beats or loses a level
  - Output: the background changes accordingly
  - Result: PASS!

- **Test #10**
  - Test: gameOverAnimation
  - Type: automated
  - Description: checks if the screen flashes when a user loses at any level
  - Initial State: the computer is waiting for the user's input
  - Input: the user enters a wrong element from the pattern that the computer played
  - Output: the screen flashes
  - Result: PASS!

- **Test #11**

- Test: checkForQuit
- Type: manual
- Description: checks if the program terminates when the quit button is clicked
- Initial State: any screen is displayed
- Input: checkForQuit() is invoked
- Output: the program terminates
- Result: PASS!

The program does not produce any output files. Unit testing is therefore not applicable for output files in this project.

5 Changes Due to Testing

- **Note Release Time**

  After playing the game, it was decided that the release time of the notes should be increased. The piano notes then were re-recorded to accommodate that decision.

- **Button Colors in JP Says**

  Rather than having different shades of a single color present, the team decided to follow a systematic order. It was decided that it would be best to have all seven colors of the rainbow present, along with black and white to have nine visually distinguishable colors/shades to use.

- **Back Button**

  After conducting the survey, it was agreed upon that the back button did not look elegant. Several individuals had suggested to simply display the word Back on the button rather than the left-pointing arrow.

- **Redisplaying Text on Screen**

  As the level changed (incremented or decremented), the text on the screen now flickers to indicate that change along with the change of colors in the background.

- **Endless Playing**

  The game has been changed to be able to reach an infinite level number. The user can continuously play until they lose or choose to stop playing for any circumstances. In addition, the song pattern in Kareem Says has been implemented in a way where the song will loop to the beginning uninterruptedly to accommodate that change.

6 Automated Testing

Generally, automated testing is not the most effective with games as there has not been a fixed artificial intelligence implementation that can mimic a human beings behaviour exactly. However, as a team, automated testing was made a possibility (to an extent). For all three modes, a random selection algorithm was implemented to select a random button to play back to the computer after it displayed the pattern, that is biased towards picking the correct button. This can somewhat identify with the behaviour of a child playing this game. Another short algorithm was developed to then analyze the given input at every level and compare it to the pattern that the computer played. If it matched exactly then the test would advance to the next level. Otherwise, the mode resets back to level one.

This program produces no output files and therefore automated testing for output files is not applicable.

7 Trace to Requirements

8 Trace to Modules

9 Code Coverage Metrics

As an estimate, all tests including manual, automated and unit testing have covered ninety-five percent of the code written in the source file. Each method/module has been tested separately as well as coupled with other appropriate modules. This estimate holds true as all modules have been tested thoroughly with various inputs when applicable, and now the output

of all remains correct as expected. Please refer to the Automated Testing and Unit Testing sections above to further validate the coverage rate estimate.