SE 3XA3: Test Report
DNA Says

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

December 5, 2016

Contents

1 Functional Requirements Evaluation **1**
- 1.1 User Input . . . 1
- 1.2 Navigation . . . 1
- 1.3 Display . . . 2
- 1.4 Functionality . . . 4
- 1.5 Parallelism . . . 7

2 Nonfunctional Requirements Evaluation **9**
- 2.1 Look and Feel . . . 9
- 2.2 Usability . . . 10
- 2.3 Performance . . . 13
- 2.4 Operational and Environmental . . . 14
- 2.5 Security . . . 15
- 2.6 Cultural . . . 16

3 Comparison to Existing Implementation **16**

4 Unit Testing **17**

5 Changes Due to Testing **21**

6 Automated Testing **22**

7 Trace to Requirements **23**

8 Trace to Modules **25**

9 Code Coverage Metrics **25**

10 Usability Survey Results **25**

List of Tables

1 **Revision History** . . . ii
2 Trace Between Tests and Functional Requirements . . . 23
3 Trace Between Tests and Non-functional Requirements . . . 24
4 Trace Between Tests and Modules . . . 25

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
05/12/16	2.1	Addition of Functional Requirements Evaluation
05/12/16	2.2	Addition of Non-Functional Requirements Evaluation
05/12/16	2.3	Addition of User Survey Results
05/12/16	2.4	Addition of Trace to Requirements
05/12/16	2.5	Addition of Trace to Modules

1 Functional Requirements Evaluation

1.1 User Input

- **Test #1: The user will be able to open the executable file.**
  - Test ID: FT-UI-1
  - Type: Functional, Dynamic, Manual
  - Initial State: Desktop will be open.
  - Input: Mouse click on DNA Says application icon.
  - Output: DNA Says's main menu will appear.
  - Execution: The tester will open the application on a CPU.
  - Result: PASS!
- **Test #2: The user will be able to select one of the three modes to play:**
  - Test ID: FT-UI-2
  - Type: Functional, Dynamic, Manual
  - Initial State: Main menu will be open.
  - Input: Mouse click on a desired mode.
  - Output: Redirection to that modes user interface.
  - Execution: The tester will attempt to open each of the three modes via the main menu.
  - Result: PASS!

1.2 Navigation

- **Test #3: The user will be able to stop the game at any time and go back to the main menu.**
  - Test ID: FT-NAV-1
  - Type: Functional, Dynamic, Manual
  - Initial State: One of the three modes will be open.
  - Input: Mouse click on "Main Menu" icon.

- Output: User will be redirected to the main menu.
- Execution: The tester will play one of the modes and attempt to go back to the main menu via the "Main Menu" icon.
- Result: PASS!

1.3 Display

- **Test #4: The game interface will open in a new window.**
  - Test ID: FT-D-1
  - Type: Functional, Dynamic, Manual
  - Initial State: Desktop will be open.
  - Input: Mouse click on DNA Says application icon.
  - Output: A new window labelled DNA Says will open and the main menu will appear.
  - Execution: The tester will open the application on a CPU.
  - Result: PASS!
- **Test #5: The main menu will display the three different modes.**
  - Test ID: FT-D-2
  - Type: Functional, Dynamic, Manual
  - Initial State: Desktop will be open.
  - Input: Mouse click on DNA Says application.
  - Output: Main menu will open.
  - Execution: The tester will check if the three modes are available via the main menu.
  - Result: PASS!
- **Test #6: Four different coloured buttons will be shown to the user during Shady Says.**
  - Test ID: FT-D-3
  - Type: Functional, Dynamic, Manual

- Initial State: Main menu will be open.
- Input: Mouse click on Shady Says mode.
- Output: Shady Says mode will open and display the user interface.
- Execution: The tester will open the mode "Shady Says" and check if there are four different coloured buttons displayed.
- Result: PASS!

- **Test #7: Each button will light up and produce a different sound when clicked.**
  - Test ID: FT-D-4
  - Type: Functional, Dynamic, Manual
  - Initial State: One of the three modes will be open, and one of the buttons will light up and produce a sound.
  - Input: Mouse click on the specific button that lit up.
  - Output: The specific button will light up and produce a sound.
  - Execution: The tester will play the game and click one of the correct button with respect to the pattern and observe the behaviour.
  - Result: PASS!

- **Test #8: There will be a score box at the top right corner.**
  - Test ID: FT-D-5
  - Type: Functional, Dynamic, Manual
  - Initial State: One of the three modes will be open.
  - Input: No input.
  - Output: No output.
  - Execution: The user will enter one of the three modes and verify that there is a score icon at the top right corner.
  - Result: PASS!

1.4 Functionality

- **Test #9: The game will have three separate modes- Kareem Says, JP Says, and Shady Says**
  - Test ID: FT-FUNC-1
  - Type: Functional, Dynamic, Manual
  - Initial State: Desktop will be open.
  - Input: Mouse click on DNA Says application
  - Output: Main menu will open.
  - Execution: The tester will check if the three modes are available via the main menu.
  - Result: PASS!
- **Test #10: Every time a user passes a level, the score goes up by one point.**
  - Test ID: FT-FUNC-2
  - Type: Functional, Dynamic, Automated
  - Initial State: One of the three modes will be open.
  - Input: Mouse click on the pattern that the computer has displayed to the user.
  - Output: The score icon will increase by 1 point.
  - Execution: The tester will play the game, they will attempt to produce the pattern and observe the score.
  - Result: PASS!
- **Test #11: Every time a user fails a level, the score is reset to zero.**
  - Test ID: FT-FUNC-3
  - Type: Functional, Dynamic, Automated
  - Initial State: One of the three modes will be open.
  - Input: Mouse click on the incorrect button.

  - Output: The score will be reset to zero and the screen will flash three times.
  - Execution: The tester will play the game, they will input the incorrect pattern and observe the score.
  - Result: PASS!

- **Test #12: At level N, a random pattern of N elements will light up and be displayed to the user.**
  - Test ID: FT-FUNC-4
  - Type: Functional, Dynamic, Automated
  - Initial State: One of the three modes will be open.
  - Input: No input
  - Output: One button will light up and produce a sound.
  - Execution: The user will attempt level one, and verify that one button lights up. They can also do this for further levels.
  - Result: PASS!

- **Test #13: The user cannot click on the buttons while the pattern is being displayed.**
  - Test ID: FT-FUNC-5
  - Type: Functional, Dynamic, Manual
  - Initial State: Level one of a mode will start.
  - Input: Mouse click on a button.
  - Output: Sound played.
  - Execution: The tester will attempt to click on a button while the computer is displaying the pattern and observe the behaviour of the program.
  - Result: PASS!

- **Test #14: The user will be able to click on the buttons once the pattern has been displayed.**
  - Test ID: FT-FUNC-6

- Type: Functional, Dynamic, Manual
- Initial State: Level one of a mode will be start.
- Input: Mouse click on a button.
- Output: Correct button click- button will light up and produce a sound. Incorrect button- screen will flash three times and a harmonic sound will play to indicate wrong input.
- Execution: After the pattern has been displayed, the tester will click the correct button and verify that it can be clicked. The tester will also click an incorrect button and verify that it can also be clicked.
- Result: PASS!

- **Test #15: A level is passed if the user repeats the pattern correctly.**
  - Test ID: FT-FUNC-7
  - Type: Functional, Dynamic, Automated
  - Initial State: Level N of a mode will be start.
  - Input: Mouse click on button that are part of the pattern the computer displayed to the user.
  - Output: After each correct click on a button, that specific button will light up and produce a sound. Also the score will increase by one when the correct pattern has been inputted completely. Then the next pattern will be displayed to the user.
  - Execution: The tester will play a level in a one of the three modes and entire the correct pattern.
  - Result: PASS!

- **Test #16: If the user fails, the game will restart- N =1.**
  - Test ID: FT-FUNC-8
  - Type: Functional, Dynamic, Automated
  - Initial State: One of the three modes will be open.
  - Input: Mouse click on an incorrect button.

- Output: The screen will flash white three times, the score will be reset to 0 and the user will be redirected to level 1.
- Execution: The tester will play the game, they will input the incorrect pattern. The user will verify that they are redirected to level 1.
- Result: PASS!

- **Test #17: User clicks exit button while pattern is being played by computer**
  - Test ID: FT-FUNC-9
  - Type: Functional, Dynamic, Manual
  - Initial State: One of the three modes will be open, pattern will be playing.
  - Input: Mouse click on exit button.
  - Output: The user is directed back to main menu.
  - Execution: The tester will play the game, wait until a pattern is being played. Tester will click the exit button while pattern is still being demonstrated. The user will verify that they are redirected back to main menu.
  - Result: PASS!

1.5 Parallelism

In this subsection, tests will be conducted in parallel for existing implementation and our implemented version of the game.

- **Test #18: User will launch both games**
  - Test ID: FT-PAR-1
  - Type: Functional, Dynamic, Manual
  - Initial State: Desktop on standby.
  - Input: User launches both games, the existing implementation and recreated implementation.

  - Output: The existing game will open on main menu page which has brief game instructions and one button to start playing game, game has only one mode. In recreated implementation, main menu will open which has no instructions and 3 modes of play to choose from.
  - Execution: The tester will launch the game and observe that the two versions of the game have different menus and one has a single mode of play while the other has 3 modes of play.
  - Result: PASS!

- **Test #19: User will wait 5 seconds after pattern is played in both versions**
  - Test ID: FT-PAR-2
  - Type: Functional, Dynamic, Manual
  - Initial State: One of the three modes will be open, pattern will play
  - Input: User does not repeat the pattern after it is played by the computer for 5 seconds
  - Output: In existing implementation, game does not end and user can get back to the game to repeat the pattern at any time. In our implemented version, user has 4 seconds to start repeating the pattern otherwise level is failed
  - Execution: The tester will open any of the modes and wait for pattern to be played, user will then wait for 5 seconds and observe response of the game, whether game ends or not.
  - Result: PASS!

- **Test #20: User will lose level on both implementations**
  - Test ID: FT-PAR-3
  - Type: Functional, Dynamic, Manual
  - Initial State: One of the three modes will be open in recreated version, pattern will play. In original implementation the game will be running and pattern playing
  - Input: User enters incorrect pattern in both games.

- Output: Both games end when wrong pattern is played and new pattern starts playing right away in both versions.
- Execution: The tester will play a wrong pattern to see whether game goes back to main menu or game just resets to level 1. In both versions game just restarts from level 1.
- Result: PASS!

2 Nonfunctional Requirements Evaluation

2.1 Look and Feel

- **Test #1: The product shall have an appealing colourful appearance.**
  - Test ID: NFT-L1
  - Type: Structural, Static, Manual
  - Initial State: Program installed onto system but not launched.
  - Input: Launch program
  - Output: Menu page should have a colourful appearance.
  - Execution: User launches the application and observes the menu's appearance, then the user satisfaction is recorded by the user.
  - Results: The average rating for appearance from user surveys was 8.2 out of 10, and several users left comments in the user survey about the simplicity and attractiveness of the design. These results imply a pass on this test.
- **Test #2: The buttons must be well designed and coloured**
  - Test ID: NFT-L2
  - Type: Structural, Dynamic, Manual
  - Initial State: Application launched on main menu page.
  - Input: User selects mode to play
  - Output: Game starts in selected mode

- Execution: User presses buttons on menu and again in mode selected to follow pattern, satisfaction regarding button appearance is recorded by the user. Test is succesful if buttons are all same shape in JP and Shady modes and if they are in a piano format in Karim mode. Buttons all be different colours in JP and Shady modes, and white and black in Karim mode.
- Results: The average rating for appearance in the user survey was 8.2 out of 10. The buttons are a part of the appearance. No specific positive or negative comments were made on the buttons so that is a pass for this test. Buttons are all the same shape but different colours in JP and Shady modes and buttons form a piano design in black and white colours in Karim mode. Therefore test is passed.

- **Test #3: The product shall have attractive sound patterns. The associated sounds with buttons must be well constructed and notes must follow harmonically.**
  - Test ID: NFT-L3
  - Type: Structural, Dynamic, Manual
  - Initial State: Program opened on main menu page.
  - Input: Select mode and start playing
  - Output: There will be a note played with every button pressed.
  - Execution: User tries to follow visual/audio pattern shown, and observes sounds made by buttons pressed. Patterns to follow should have notes associated with them them compose a song or a little rhyme. User satisfaction with sounds is recorded. Test is successful if pattern sounds meaningful, if succesive notes have harmony to a certain extent.
  - Results: Results for song rating were exceptionally high with an average of 8.8 out of 10 from user surveys. Several users left comments that they liked the Love Story song in the Karim mode. Test passed.

2.2 Usability

- **Test #4: The product shall be easy to install for all users.**

  - Test ID: NFT-U1
  - Type: Structural, Dynamic, Manual
  - Initial State: User's system active and functional, and game not installed
  - Input: User installs game via USB
  - Output: Game is installed easily onto system
  - Execution: User plugs in USB to transfer game from the USB key to their system, game is transferred with ease. User review is then recorded. Test is succesful if all testers are able to successfully install the game.
  - Results: All the users from the test population were able to quickly install the game on their devices. Test passed.

- **Test #5: The product shall be easy to use for people of all ages, including children.**
  - Test ID: NFT-U2
  - Type: Structural, Dynamic, Manual
  - Initial State: Program opened on main menu page.
  - Input: Select any mode and start playing.
  - Output: A simple one element pattern will be played, user asked to repeat the pattern. Pattern length will keep incrementing
  - Execution: User starts playing the game and following patterns shown, instructions will be provided on every page. Test is succesful if all test population are able to complete at least the first level.
  - Results: The entire test population was able to get past level one, that means everyone understood how the game is supposed to be played. Test passed.

- **Test #6: The product shall produce a nice animation in the background when user loses or wins a level.**
  - Test ID: NFT-U3
  - Type: Structural, Dynamic, Manual

  - Initial State: Program opened on main menu page.
  - Input: Select any mode and start playing.
  - Output: User will progress through levels. If the user gets the pattern right, the next pattern is played after a change in colour in the background. If user enters wrong pattern screen flashes three times with five notes sounding as well
  - Execution: User plays the game, records satisfaction with game response to patterns entered (both right or wrong patterns). Test is successful if user clearly understands whether they passed a certain level or failed it.
  - Results: Since all users got high scores above 6, that means users understood whether they passed each level or not. No comments were left on confusion regarding level progression. Test passed.

- **Test #7: The product shall produce patterns both visually and auditory so as to accommodate for users with visual or auditory problems that they can use an alternative pattern means.**
  - Test ID: NFT-U4
  - Type: Structural, Dynamic, Manual
  - Initial State: Program opened on main menu page.
  - Input: Mute sound on system. Select any mode and start playing.
  - Output: User will follow pattern shown based only on visual buttons displayed.
  - Execution: User plays the game with no sounds, records ease of game and whether there were challenges following the patterns with no sounds. Test is successful if tester gets through at least 1 level with sounds muted.
  - Results: The three developers as well as two additional testers ran this test and played the game with the sounds muted. All five testers were able to progress through level 1. Test passed.

2.3 Performance

- **Test #8: The application should be able to recognize whether the user has entered the right pattern as soon as they finish pressing the last button.**
  - Test ID: NFT-P1
  - Type: Structural, Dynamic, Manual
  - Initial State: Program launched on main menu page
  - Input: User selects mode and starts playing, enters wrong button not to follow pattern.
  - Output: Screen should flash three times with five notes played.
  - Execution: User observes how long the game takes to detect wrong pattern entered, user records satisfaction with game response. Test is successful if user is aware if they lost the level or else they know they passed the level.
  - Results: No users reported any confusion or misunderstanding in user survey. Ease of use rating averaged 9 out of 10 so it is clear users did not face difficulties with understanding the game's response to their input. Test passed.
- **Test #9: The application must be specific to each button press. Button press confusion or mistake must not be tolerated**
  - Test ID: NFT-P2
  - Type: Structural, Dynamic, Manual
  - Initial State: Application launched on any mode.
  - Input: User follows all notes of a pattern except the last one, presses a wrong button
  - Output: Game should immediately detect wrong pattern input and flash three times.
  - Execution: User plays game until reaching a level with a long pattern, the user follows the pattern correctly except for the last note, records whether the game detects the incorrect note or regards the pattern as correct.

- Results: The three developers ran this test. Upon reaching level 10, all buttons in the pattern were pressed correctly except the last one. The system detected the wrong input and game was reset to level 1. Test passed.

- **Test #10: The application must be able to produce and receive patterns of infinite length.**
  - Test ID: NFT-P3
  - Type: Structural, Dynamic, Manual
  - Initial State: Program opened on main menu page.
  - Input: Select mode and start playing, reaches level 30, completes level.
  - Output: The game will be able to carry on normally.
  - Execution: User reaches level 30 of any mode which consists of a pattern of length 30 buttons.
  - Results: The three developers collaborated to reach level 30. Each developer memorized 10 buttons so together they were able to reach level 30. Following level 30, the system played a pattern of length 31, therefore level 31 was reached. Game has infinite levels. Test passed.

2.4 Operational and Environmental

- **Test #11: The product is to be used on laptops and desktops.**
  - Test ID: NFT-O1
  - Type: Structural, Dynamic, Manual
  - Initial State: Laptop and desktop devices available for use.
  - Input: Launch program on laptop and desktop
  - Output: Game should launch on both laptop and desktop.
  - Execution: User launches the application on different devices, laptops and desktops. User records whether application launches on every device or not.

  - Results: Game was launched succesfully on various laptops and desktops.

- **Test #12: The product shall run on Windows, Linux and Mac OS X environments.**
  - Test ID: NFT-O2
  - Type: Structural, Dynamic, Manual
  - Initial State: OS ready for use with game installed.
  - Input: User launches application.
  - Output: Game starts; main menu page appears.
  - Execution: User tests application on three different operating systems: Windows, Linux, and Mac OS X and records whether errors occur on any of the environments.
  - Results: Game was installed and launched on Mac OS, Windows and Linux computers succesfully. Test passed.

2.5 Security

- **Test #13: The application shall not store, transmit, or upload any user data.**
  - Test ID: NFT-S1
  - Type: Structural, Dynamic, Manual
  - Initial State: Application launched in any game state.
  - Input: Follow pattern.
  - Output: Pattern displayed, flashing with wrong input.
  - Execution: Developer plays game, checks if application has stored any information from user.
  - Results: All three developers confirm that game does not store, transmit, or upload any personal user data. Test passed.

2.6 Cultural

- **Test #14: The application shall not contain any images or text that can be reasonably foreseen as potentially offensive to users of all cultures, backgrounds and ethnicities.**
  - Test ID: NFT-C1
  - Type: Structural, Dynamic, Manual
  - Initial State: Application launched in any game state.
  - Input: Follow pattern.
  - Output: Patterns displayed, flashing with wrong input.
  - Execution: A test group comprising of different cultural backgrounds or at least knowledge of different cultural backgrounds play the game and record whether any component of the game is deemed offensive to any known culture or religion to test group.
  - Results: Friendliness rating averaged 9 out of 10 from user surverys. No user commented on any offensive material. Test passed.

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

Requirement	Tests
Functional Requirements Testing	
Requirement #1	FT-UI-1
Requirement #2	FT-D-1
Requirement #3	FT-FUNC-1
Requirement #4	FT-UI-2
Requirement #5	FT-D-2
Requirement #6	FT-D-3
Requirement #7	FT-D-4
Requirement #8	FT-NAV-1, FT-FUNC-9
Requirement #9	FT-FUNC-2
Requirement #10	FT-FUNC-3
Requirement #11	FT-D-5
Requirement #12	FT-FUNC-4
Requirement #13	FT-FUNC-5
Requirement #14	FT-FUNC-6
Requirement #15	FT-FUNC-7
Requirement #16	FT-FUNC-8, FT-PAR-3

Table 2: Trace Between Tests and Functional Requirements

Requirement	Tests
Non-functional Requirements Testing	
Requirement #1	NFT-LI, NFT-L2
Requirement #2	NFT-L1, NFT-L2
Requirement #3	NFT-L3
Requirement #4	NFT-L1, NFT-L2
Requirement #5	NFT-U2
Requirement #6	NFT-U1
Requirement #7	NFT-U2
Requirement #8	NFT-U1, NFT-U2
Requirement #9	NFT-L3
Requirement #10	NFT-U3
Requirement #11	NFT-U4
Requirement #12	NFT-P1
Requirement #13	NFT-P2
Requirement #14	NFT-01, NFT-02
Requirement #15	NFT-P3
Requirement #16	NFT-01
Requirement #19	NFT-02
Requirement #20	NFT-S1
Requirement #21	NFT-C1

Table 3: Trace Between Tests and Non-functional Requirements

8 Trace to Modules

Module	Test
Module #1	FT-UI-1
Module #2	FT-D-1, FT-UI-2
Module #3	FT-D-1, FT-FUNC-1
Module #4	FT-FUNC-1, FT-D-2, FT-D-3, FT-FUNC-4
Module #5	FT-FUNC-5, FT-FUNC-7, FT-FUNC-8, FT-PAR-3
Module #6	FT-FUNC-1, FT-D-2, FT-D-3, FT-FUNC-4
Module #7	FT-UI-2, FT-D-2
Module #8	FT-D-3
Module #9	FT-FUNC-2
Module #10	FT-FUNC-2, FT-FUNC-3, FT-D-5
Module #11	FT-D-5
Module #12	FT-FUNC-1, FT-UI-2, FT-D-2, FT-D-3
Module #13	FT-D-3, FT-D-4, FT-FUNC-4, FT-FUNC-6
Module #14	FT-FUNC-8, FT-PAR-3
Module #15	FT-FUNC-8, FT-PAR-3
Module #16	FT-FUNC-5

Table 4: Trace Between Tests and 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.

10 Usability Survey Results

Calculated averages for user surveys:

User Experience Survey

The following survey should be filled out after playing the game for at least 15 minutes.

Time spent playing: 20 minutes

Please provide a ranking between 0 and 10 in each of the following categories, 0 being most boring, 10 being most fun. Please include notes on the things you liked and the ones you did not like. Please also include suggestions to improve the game.

Entertainment:
7.5

Friendliness:
9

Ease of use:
9

Game Difficulty:
7

Appearance:
8.2

Sounds:
8

Song:
8.8

High Score:
9

Comments: