SE 3XA3: Test Plan
DNA Says

Team #10, Team Name: DNA
Kareem Abdel Mesih (abdelk2)
John-Paul Dakran (dakranj)
Shady Nessim (nessimss)

October 31, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
2016/10/25	1.0	Addition of General Information section
2016/10/29	1.1	Addition of Comparison to Existing Implementation section
2016/10/30	1.2	Addition of Tests for Proof of Concept
2016/10/30	1.3	Addition of Tests for Functional Requirements

1 General Information

1.1 Purpose

In the engineering process, verification and validation of the requirements outlined in the Software Requirements Specification (SRS) document is essential. This process is executed through a series of tests executed on the requirements to prove that the functionality of the game is correct. This document serves the purpose of outlining how the requirements will be validated and verified.

The implementation of the game DNA Says consists of numerous functional capabilities. These functional capabilities range from detecting user input to outputting a correct sound at a precise given time. The complete set of requirements will be broken down into specific and simple tests to prove the functionality of each specific requirement.

1.2 Scope

The main objective of this document is to outline an agreed upon set of tests that will be performed on the software system to validate its functionality. The scope of the testing for this game includes testing the animations, sounds, buttons, integration of the system, and all functional and non-functional requirements outline in the Software Requirements Specification (SRS) document.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
SRS	Software Requirement Specification
PoC	Proof of Concept
GUI	Graphical User Interface
CPU	Central Processing Unit
OS	Operating System
USB	Universal Serial Bus

Table 3: **Table of Definitions**

Term	**Definition**
Mode	Different subsections of the game
Gantt Chart	Chart outlining the timeline of the project.
Functional	Focused on results not with how those results are achieved.
Structural	Focused on testing the actual processes - how results are achjeived.
Dynamic	Test cases are run and checked against expected behaviour.
Static	Testing syntax and requirements.
Automated	Testing done automatically via the computer.
Manual	Testing done manually via a human user.
Windows	An operating system.
Linux	An operating system.
Mac OS X	An operating system.

1.4 Overview of Document

This document outlines a collection of information about the software system - DNA Says - that is in the process of creation. The test plan document begins by describing the software system and its functionality. It then proceeds to introducing the reader with the test team and the plan for testing - I.e. Testing tools and the testing schedule.

Next, the tests for functional and non-functional requirements will be described. Each test will have a type, initial state, input, output, and description of how the test will be performed. The same format will be used for the next section which outlines the tests for the proof of concept.

Proceeding, the reader will be introduced to a concise comparison between the original implementation and the implementation that is currently in the process of creation. Next the unit testing plan will be revealed to the reader which describes the unit testing of the internal functions and output files. The test plan document will be concluded by the appendix which will hold a list of symbolic parameters and survey questions for user testing.

2 Plan

This section provides information about the plan for testing. It includes description of software, testing members as well as other details on tools and automation for testing. Provided at the end is a link to testing schedule on the Gantt chart.

2.1 Software Description

This software is a game adapted from the popular game Simon Says. Three modes are provided, each with a different style of pattern matching, user shall observe visual/audio pattern and reproduce the pattern, then the pattern will be extended and so on.

2.2 Test Team

The test team consists of:

- Kareem Abdel Mesih
- John-Paul Dakran
- Shady Nessim

Family and friends will also be involved in the testing process through user surveys.

2.3 Automated Testing Approach

Automated testing will be used to test the basic functionality of the system. A random button will be pressed to examine the system's response to correct/incorrect input. Unit testing will also be applied to test functions and methods.

2.4 Testing Tools

2.5 Testing Schedule

See Gantt Chart at the following url ...

Table 4: **Testing Tools**

Tool	Description	Use
unittest	Unit Testing Framework	Unit Testing

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 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 main menu will appear.
  - Execution: The tester will open the application on a CPU.
- **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 desire mode.
  - Output: Redirection to that modes user interface.
  - Execution: The tester will attempt to open each of the three modes via the main menu.

3.1.2 Navigation

- **Test #3: The user will be able to exit 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.

3.1.3 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.
- **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.
- **Test #:6 Four different coloured disks 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 disks displayed.

- **Test #7: Each disk/key 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 disks/keys will light up and produce a sound.
  - Input: Mouse click on the specific disk that lit up.
  - Output: The specific disk/key will light up and produce a sound.
  - Execution: The tester will play the game and click one of the correct disks/keys with respect to the pattern and observe the behaviour.

- **Test #8: There will be a score icon in the bottom 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 in the bottom right corner.

3.1.4 Functionality

- **Test #9: The game will have 3 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 3 modes are available via the main menu.

- **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.

- **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 pattern.
  - 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.

- **Test #12: At level N, a random pattern of N disks 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 disk/key will light up and produce a sound.
  - Execution: The user will attempt level one, and verify that one key/disk lights up. They can also do this for further levels.

- **Test #13: The user cannot click the disks while the pattern is being displayed.**
  - Test ID: FT-FUNC-5
  - Type: Functional, Dynamic, Manual
  - Initial State: Level one of a mode will be open.
  - Input: Mouse click on a key/disk.
  - Output: No output.
  - Execution: The tester will attempt to click a key/disk while the computer is displaying the pattern and observe the behaviour of the program.

- **Test #14: The user will be able to click the disks once the pattern has been displayed.**
  - Test ID: FT-FUNC-6
  - Type: Functional, Dynamic, Manual
  - Initial State: Level one of a mode will be open.
  - Input: Mouse click on a key/disk.
  - Output: Correct key/disk click - Key/disk will light up and produce a sound. Incorrect key/disk - Screen will flash threee times.
  - Execution: After the pattern has been displayed, the tester will click the correct disk and verify that it can be clicked. The tester will also click an incorrect disk and verify that it can also be clicked.

- **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 open.
  - Input: Mouse click on key/disk that are part of the pattern the computer displayed to the user.
  - Output: After each correct click on a key/disk, that specific key/disk 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.
- **Test #16: If the user fails, the game will restart - I.e. 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 disk/key.
  - Output: The screen will flash white three times, the score will be reset to 0 and the user will be redirected to level 0.
  - Execution: The tester will play the game, they will input the incorrect pattern. The user will verify that they are redirected to level 1.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

- **Test #1: The product shall have an appealing colorful 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 be displayed with colorful appearance.
- Execution: User launches the application and observes menu appearance, user satisfaction is recorded by user.

- **Test #2: The buttons must be well designed and colored**
  - 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, satisafaction regarding button appearance is recorded by the user.

- **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 satisafaction with sounds is recorded.

3.2.2 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, game not installed
  - Input: User installs game from USB transfer
  - Output: Game is installed easily onto system
  - Execution: User plugs in USB to transfer game from USB to their system, game is transfered with ease. User review recorded.

- **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 note/one button 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, no instructions page will be necessary to clarify the game.

- **Test #6: The product shall produce friendly messages 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 user gets pattern right, next pattern is played without a message stating the user got it right. If user enters wrong pattern screen flashes three times with three notes sounding with the flashing
  - Execution: User plays the game, records satisfaction with game response to patterns entered (both right or wrong patterns)

- **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.

3.2.3 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 three notes.
  - Execution: User observes how long the game takes to detect wrong pattern entered, user records satisfacrtion with game response.
- **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 crrectly except for the last note, records whether the game detects the incorrect note or regards the pattern as correct.

- **Test #10: The application must be able to produce and receive patterns as long as 25 buttons.**
  - Test ID: NFT-P3
  - Type: Structural, Dynamic, Manual
  - Initial State: Program opened on main menu page.
  - Input: Select mode and start playing, reaches level 25, completes level.
  - Output: There will be a winning message/sign displayed.
  - Execution: User reaches level 25 of any mode which consists of a pattern of length 25 butons/notes. User should recieve a winning message upon completion of mode.

3.2.4 Operational and Environmental

- **Test #11: The product should be able to be used on laptops and desktops.**
  - Test ID: NFT-O1
  - Type: Structural, Dynmic, 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.

- **Test #12: The product shall run in 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.

3.2.5 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: Patterns displayed, flashing with wrong input.
  - Execution: Developer plays game, checks if application has stored any information from user.

3.2.6 Cultural

- **Test #14: The application shall not contain any imagery 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.

4 Tests for Proof of Concept

4.1 User Input

- **Test #1: Opening the DNA Says application**
  - Test ID: POC-UI-1
  - Type: Functional, Dynamic, Manual
  - Initial State: Desktop will be open.
  - Input: Mouse click on DNA Says application icon.
  - Output: New window labelled DNA Says opens.
  - Execution: The tester will open the application on a CPU.
- **Test #2: Clicking a correct key**
  - Test ID: POC-UI-2
  - Type: Dynamic, Automated
  - Initial State: Kareem Says mode will be open.
  - Input: Mouse click on a key that is part of the pattern given by the computer.
  - Output: Key will light up and make a sound.
  - Execution: The tester will click on a correct key and observe the behaviour of the game.
- **Test #3: Clicking an incorrect key**
  - Test ID: POC-UI-2
  - Type: Dynamic, Automated
  - Initial State: Kareem Says mode will be open.

- Input: Mouse click on a key that is not part of the pattern given by the computer.
- Output: Screen will flash white three times and a sound will be outputted indicating incorrect key.
- Execution: The tester will click on an incorrect key and observe the behaviour of the game.

4.2 Display

- **Test #4: Viewing mode - "Kareem Says"**
  - Test ID: POC-D-1
  - Type: Functional, Dynamic, Manual
  - Initial State: Desktop will be open.
  - Input: Mouse click on the DNA Says application icon.
  - Output: Kareem Says mode will open displaying a piano.
  - Execution: The tester will open the application on a CPU.
- **Test #5: Failure signal**
  - Test ID: POC-D-2
  - Type: Functional, Dynamic, Manual
  - Initial State: Kareem Says mode will be open.
  - Input: Mouse click on an incorrect key.
  - Output: The screen will flash white three times and a sound indicating incorrect key will be outputted.
  - Execution: The tester will click on an incorrect key.

4.3 Functionality

- **Test #6: Correct score increment**
  - Test ID: POC-FUNC-1
  - Type: Functional, Dynamic, Automated
  - Initial State: Kareem Says mode will be open.

- Input: Mouse click on a correct key.
- Output: The key will light up and a sound will be outputted. The score increment will be incremented by one.
- Execution: The tester will click on a correct key and view the score icon.

- **Test #7: Correct pattern increment**
  - Test ID: POC-FUNC-1
  - Type: Functional, Dynamic, Automated
  - Initial State: Kareem Says mode will be open.
  - Input: Mouse click on correct keys that make up the pattern.
  - Output: Each key will light up and produce its sound. At each level, one additional key will be appended to the pattern.
  - Execution: The tester will play the game and observe that one additional key is added to the pattern at each level.

5 Comparison to Existing Implementation

The implementation of DNA Says and Simon Says have many similarities and differences that will be outlined in this section. The main principle of the game stays constant however there are many improvements and modifications that have been added to truly make DNA Says unique.

The first significant difference between DNA Says and Simon Says is the fact that DNA Says has three modes. The original version of Simon Says only has one mode or one level which consists of 4 buttons that display the pattern to the user. In contrast DNA Says has three levels or three modes - each with a different implementation but overall the same theory from the Simon Says implementation. The first mode - Kareem Says, consists of piano keys that will display the pattern to the user. The second mode - Shady Says, consists of 4 different coloured buttons that will display the pattern to the user. The final mode - JP Says, consists of 9 different coloured buttons that will display the pattern to the user.

The second critical difference between the two implementations is the 4 second timeout. The original implementation - Simon Says, has no timeout. This means when the user clicks a button and does not click the next button, the program will wait for the user to enter the next button. In contrast, DNAY Says has a 4 second timeout that is implemented. When a user clicks a button/key and does not click the next button within 4 seconds, the player loses.

The third significant difference between DNA Says and Simon Says is the protocol the program displays to the user when an incorrect button/key is inputted. The original version - Simon Says - displays the correct button that should have been clicked along with a sound that indicates an incorrect key has been inputted. The implementation of DNA Says makes the entire screen flash white 3 times and outputs a harmonic combination of notes that indicate an incorrect input.

Along with all the differences that truly make the game DNA Says truly unique, there are numerous similarities as the essence of the game is the same. Both implementations of the game have a main menu that allow the user to enter a mode and play the game. During a game, there is a score icon which displays the current score and there is also an exit button which will allow the user to return to the main menu.

6 Unit Testing Plan

6.1 Unit testing of internal functions

6.2 Unit testing of output files

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

7.2 Usability Survey Questions?

This is a section that would be appropriate for some teams.