SE 3XA3: Test Plan
DNA Says

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

December 4, 2016

Contents

List of Tables

Table 1: **Revision History**

Date	**Version**	**Notes**
2016/10/25	1.0	Addition of General Information
2016/10/29	1.1	Addition of Comparison to Existing Implementation
2016/10/30	1.2	Addition of Tests for Proof of Concept
2016/10/30	1.3	Addition of Tests for Functional Requirements
2016/10/30	1.4	Addition of Tests for Non-Functional Requirements
2016/10/31	1.5	Addition of Unit Tests
2016/10/31	1.6	Addition of Appendix
2016/10/31	2.0	Revision 0
2016/12/04	3.0	Revision 1

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 split 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 outlined in the Software Requirements Specification (SRS) document.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
SRS	Software Requirements 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 the results excluding how they are achieved
Structural	Focused on testing the actual processes - how results are achieved
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. This test plan document begins by describing the software system and its functionality. It then proceeds with introducing the reader with the test team and the plan for testing- 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 descriptions 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 a given visual/audio pattern and reproduce the same pattern, then the pattern will be extended and the game continues in that manner.

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

For unit testing we will be using the Python testing framework unittest. Sometimes referred to as PyUnit, this is a Python language version of JUnit, by Kent Beck and Erich Gamma.

For system testing we will be running the game by executing it on various devices and observing the functionalities.
For code analysis we will use humans with coding experience to analyze the code.

Table 4: **Testing Tools**

Tool	**Description**	**Use**
unittest	Python Unit Testing Framework	Unit Testing

2.5 Testing Schedule

See testing schedule here: Gantt Chart

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's 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 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.

3.1.2 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.

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 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.

- **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.

- **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.

3.1.4 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.
- **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 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.

- **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.

- **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.

- **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.

- **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.

- **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.

- **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 clickthe exit button while pattern is still being demonstrated. The user will verify that they are redirected back to main menu.

3.1.5 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.

- **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 ofthe game, whether game ends or not.

- **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.

3.2 Tests for Nonfunctional Requirements

3.2.1 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.
- **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.
- **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.

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, 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.
- **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

- **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.

- **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.

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 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.
- **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.
- **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 50, completes level.
  - Output: The game will be able to carry on normally.
  - Execution: User reaches level 50 of any mode which consists of a pattern of length 50 buttons.

3.2.4 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.
- **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.

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: Pattern 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 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.

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 part of 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 box.
- **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 main 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 four different coloured buttons that will display the pattern to the user. The final mode, JP Says, consists of nine different coloured buttons that will display the pattern to the user.

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 box which displays the current score and there is also an exit button which will allow the user to return to the main menu.

Under tests for functional requirements a section has been added for parallel tests in which some function is performed in both the existing implementation and the recreated implementation to highlight the differences and similarities between the two programs. For example when user does not enter the pattern within a certain time frame in the recreated version, they lose the game, whereas in the existing implementation the user has infinite amount of time to enter the pattern after it is displayed.

6 Unit Testing Plan

A set of unit tests will be performed on functions used in the implementation of the system. This testing is to ensure that changes to the game do not break the basic functioning of the game. The different modules will be unit tested independently to make sure that changes to the game do not break the basic functionality of the game. Test cases that will be used are outlined below.

6.1 Unit testing of internal functions

- **Test #1**
  - Test: main
  - Type: Functional, dynamic, 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
- **Test #2**
  - Test: setup
  - Type: Functional, dynamic, 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
- **Test #3**
  - Test: update

  - Type: Functional, dynamic, automated.
  - Description: checks if the screen is updated accordingly inside any moden
  - 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

- **Test #4**
  - Test: checkForQuit
  - Type: Functional, dynamic, manual
  - Description: Tests if game quits when quit button is pressed at any point of the game.
  - Initial State: Game active in any of the modes, a pattern is being played by the computer for the user to repeat it.
  - Input: Quit button pressed.
  - Output: Game quits.

- **Test #5**
  - Test: flashKeyAnimation
  - Type: Unit Test (dynamic, automated)
  - Description:Tests if key animation is error proof with wrong inputs
  - Initial State: Game launched in any mode
  - Input: Key, animationSpeed
  - Output: Button visual and associated sound, no output for inputs out of range or non existing

- **Test #6**
  - Test: changeBackgroundAnimation
  - Type: Unit Test (dynamic, automated)
  - Description: Tests changing background colour with wrong input.
  - Initial State: Game launched in any mode

    - Input: animationSpeed.
    - Output: Patterns displayed, background colour changed.

- **Test #7**
    - Test: gameOverAnimation
    - Type: Unit Test (dynamic, automated)
    - Description: Tests game response to different game over responses in terms of flashing screen and chord played.
    - Initial State: User enters wrong pattern
    - Input: color, animationSpeed
    - Output: Screen flashes white three times and chord played.

- **Test #8**
    - Test: getKeyClicked
    - Type: Unit Test (dynamic, automated)
    - Description: Tests key clicks used for input by user.
    - Initial State: Any game state
    - Input: x and y coordinates of the click
    - Output: If click is within range of button, that button is returned, otherwise click does nothing.

6.2 Unit testing of output files

The application does not generate any output files. The output for this application is the patterns played by the system as well as the error flashing in response to wrong pattern input. Test coverage for these outputs is covered in the above unit testing area.

7 Appendix

7.1 Symbolic Parameters

Table 5: **Symbolic Parameters**

Parameter	**Description**
key	Buttons are represented by keys in the code, each button has its own key.
checkForQuit	Function to check if Quit button has been pressed.
flashKeyAnimation	Function to produce the flashing animation for a select key passed to it as a parameter.
changeBackgroundAnimation	Function to change background colour dynamically.
gameOverAnimation	Function that causes the background to flash three times along with a chord played. This happens when user enters wrong pattern.
getKeyClicked	Function to check coordinates of mouse click and map it to corresponding button.

7.2 Usability Survey Questions

Sample Survey to be handed out to test users:

User Experience Survey

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

Time played:

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:

Difficulty:

Appearance:

Sounds:

Song: