Testing Documentation

1. Main Page

Version	Published	Changed By	Comment
CURRENT (v. 21)	**Apr 02, 2024 03:03**	Mohammad Shayaan Shahid	
v. 20	Apr 02, 2024 03:02	Mohammad Shayaan Shahid	
v. 19	Apr 02, 2024 03:01	Muzzammil Mudathir	
v. 18	Apr 02, 2024 02:58	Muzzammil Mudathir	
v. 17	Apr 01, 2024 23:22	Aryaman Arora	
v. 16	Apr 01, 2024 23:16	Aryaman Arora	
v. 15	Apr 01, 2024 22:53	Aryaman Arora	
v. 14	Apr 01, 2024 22:35	Mohammad Shayaan Shahid	
v. 13	Apr 01, 2024 22:35	Mohammad Shayaan Shahid Muzzammil Mudathir	
v. 12	Apr 01, 2024 21:01	Michael Peter Timothy Turkstra	
v. 11	Apr 01, 2024 20:26	Michael Peter Timothy Turkstra	

v. 10	Apr 01, 2024 20:26	Muzzammil Mudathir Michael Peter Timothy Turkstra
v. 9	Apr 01, 2024 20:24	Mohammad Shayaan Shahid Muzzammil Mudathir
v. 8	Apr 01, 2024 20:21	Muzzammil Mudathir
v. 7	Apr 01, 2024 20:18	Mohammad Shayaan Shahid Muzzammil Mudathir
v. 6	Apr 01, 2024 20:16	Mohammad Shayaan Shahid
v. 5	Apr 01, 2024 20:16	Mohammad Shayaan Shahid
v. 4	Apr 01, 2024 20:15	Mohammad Shayaan Shahid
v. 3	Apr 01, 2024 20:13	Mohammad Shayaan Shahid
v. 2	Apr 01, 2024 20:12	Muzzammil Mudathir
v. 1	Mar 31, 2024 23:22	Aryaman Arora

2. Introduction

Overview

This software is designed as an educational tool to enhance logical reasoning skills through interactive gam

eplay. It introduces a unique approach to learning logic, offering various modes and levels to cater to different learning paces and styles. Users will engage with sentences and logical symbols, applying logical connectors to form well-formed formulas (WFFs). The game includes a level system and a distinctive rapid-fire lightning mode to challenge users further, ensuring a comprehensive learning experience.

Objectives

The primary objective of this project is to facilitate collaborative learning and software development within a team, aiming to deliver a product that is both functional and aesthetically pleasing. The software's design goals include a clean, responsive interface that appeals to users and ensures ease of navigation. From an educational standpoint, the game aims to teach logical operators and the construction of well-formed formulas, making the learning process enjoyable and rewarding.

References

- https://www.geeksforgeeks.org/how-to-create-buttons-in-a-game-using-pygame/
- Project specification document
- requirements and documentation document
- design documentation
- https://www.youtube.com/watch?v=YbouZ2X8fGk
- https://www.youtube.com/watch?v=Ro82dac_J1Y
- https://www.youtube.com/watch?v=y9VG3Pztok8
- https://www.youtube.com/watch?v=G8MYGDf_9ho

3. Test Plan

Sign In

Test Case Name:	Test_Text_Input
Test Case Description:	Checks if user inputted data is recorded
Test Steps:	1. Run sign_in.py 2. Input test data 3. If test data entered, print passed
Pre-Requisites:	None
Expected Results:	Passed printed
Test Category:	Validation Test
Requirements:	Response to Player Input
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	Passed printed
Remarks:	Sign In correctly records user inputs

Test Case Name:	Test_Buttons
Test Case Description:	Checks to see if buttons are initialized correctly
Test Steps:	1. Set up button that just prints "pressed" to console 2. Run Sign_In.py 3. Press button and check if "pressed" is in console
Pre-Requisites:	None

Expected Results:	"pressed" printed to console
Test Category:	Validation Test
Requirements:	Mouse Based Interaction
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	"pressed" printed to console
Remarks:	Buttons work

Test Case Name:	Test_Add_Username
Test Case Description:	Checks if an account is linked to the inputed username, if not adds account
Test Steps:	1. Run Sign_In.py 2. Input unknown username
Pre-Requisites:	None
Expected Results:	Usernames.txt file updated with new username
Test Category:	Validation Testing
Requirements:	Multiple Players
Automation:	Manually Run by Humans
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	Usernames.txt file updated with new username
Remarks:	New accounts can be created

Test Case Name:	Test_Key_Entry
Test Case Description:	Checks if a valid developer/instructor key is inputted, and signs into developer/instructor main menu
Test Steps:	1. Run Sign_In.py 2. Input valid username 3. Input key
Pre-Requisites:	User has pre-existing account
Expected Results:	User pathed into developer/instructor main menu
Test Category:	Validation and Integration Test
Requirements:	Instructor Dashboard
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024

Pass/Fail:	Pass
Test Results:	User pathed into developer/instructor main menu
Remarks:	Sign_In correctly directs user to developer/instructor main menu

Test Case Name:	Test_Sign_In
Test Case Description:	Checks if a valid username is inputted, and signs user into player main menu
Test Steps:	1. Run Sign_In.py 2. Input valid username
Pre-Requisites:	User has pre-existing account
Expected Results:	User pathed into player main menu
Test Category:	Integration Testing
Requirements:	Multiple Players
Automation:	Manually Run by humans
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	User pathed into player main menu
Remarks:	Sign_In correctly directs user to player main menu

Training Mode

Test Case Name:	Test_Files
Test Case Description:	Check to see if the txt files are read and written to correctly
Test Steps:	1. Run trainingmode.py 2. Print out the variables: username, questions, options, answers and difficulty to console 3. Visually check the contents of questions.txt, answers.txt, and to see if the variables were assigned correctly
Pre-Requisites:	• User is signed in as a player • User presses new game • User presses training mode
Expected Results:	• username = user inputted username • questions = content of question.txt file, without question number • answers = content of answers.txt file, without question number
Test Category:	Validation Test
Requirements:	The application will store all data locally
Automation:	Manually Run by Human
Date Run:	Mar. 30, 2024
Pass/Fail:	Pass

Test Results:	• username = user inputted username • questions = content of question.txt file, without question number • answers = content of answers.txt file, without question number • difficulty = content of difficulty.txt file, without question number
Remarks:	Test was passed and correctly accesses all question and user data

Test Case Name:	Test_CorrectWrong
Test Case Description:	Checks to see if there is feedback displayed on the screen
Test Steps:	1. Run trainingmode.py 2. Drag the correct answer into the box
Pre-Requisites:	None
Expected Results:	Correct will be displayed on the screen until user clicks on next question
Test Category:	Validation Test
Requirements:	Mouse Based Interaction
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	Correct was printed onto the screen
Remarks:	Feedback works

Lightning Mode

Test Case Name:	Test_Files
Test Case Description:	Check to see if the txt files are read and written to correctly
Test Steps:	1. Run lightningMode.py 2. Print out the variables: username, questions, options, answers and difficulty to console 3. Visually check the contents of cur_username.txt, questions.txt, answers.txt, and difficulty.txt to see if the variables were assigned correctly
Pre-Requisites:	• User is signed in as a player • User presses new game • User presses lightning mode
Expected Results:	• username = user inputted username • questions = content of question.txt file, without question number • answers = content of answers.txt file, without question number • difficulty = content of difficulty.txt file, without question number

Test Category:	Validation Test
Requirements:	The application will store all data locally
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	• username = user inputted username • questions = content of question.txt file, without question number • answers = content of answers.txt file, without question number • difficulty = content of difficulty.txt file, without question number
Remarks:	Test was passed and correctly accesses all question and user data

Test Case Name:	Test_Draggable_Objects
Test Case Description:	Checks to see if draggable objects are initialized correctly
Test Steps:	1. Run lightningMode.py 2. Visually examine if the draggable objects on screen are from options.txt 3. Drag object around the screen to confirm that it is movable 4. Drag object over solution box, "collision" will be printed to console 5. Drag correct object over solution box, "correct" will be printed to console
Pre-Requisites:	• User is signed in as a player • User presses new game • User presses lightning mode
Expected Results:	• Object is movable • "collision" printed to console • "correct" printed to console
Test Category:	Validation Test
Requirements:	Mouse Based Interaction
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	• Object is movable • "collision" printed to console • "correct" printed to console
Remarks:	Test passed, draggable object are implemented correctly

Test Case Name:	Test_Write_To_Leaderboard
Test Case Description:	Checks to see if users high scores are added to the leaderboard

Test Steps:	1. Run lightningMode.py 2. Set high score 3. Navigate to Leaderboards and visually verify the record is saved
Pre-Requisites:	• User is signed in as a player • User presses new game • User presses lightning mode
Expected Results:	New score is added to the leaderboard
Test Category:	Integration Testing
Requirements:	High Score Table
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	New score is added to the leaderboard
Remarks:	lightningMode correctly updates leaderboards with new high scores

Test Case Name:	Test_Timer
Test Case Description:	Checks if the game session ends when timer is done
Test Steps:	1. Run lightningMode.py 2. Wait for timer to time out 3. Check if lightning mode closes and main menu is opened
Pre-Requisites:	• User is signed in as a player • User presses new game • User presses lightning mode
Expected Results:	Results shown, then lightning mode closes and main menu opens
Test Category:	Validation and Integration Test
Requirements:	Time-Based Elements
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	Results shown, then lightning mode closes and main menu opens
Remarks:	Lightning Mode timer works correctly

Developer Mode

Test Case Name:	Test_Files

Test Case Description:	Check to see if the txt files are read and written to correctly
Test Steps:	1. Run developerMode.py 2. Print out the variables: username, questions, options, answers and difficulty to console 3. Visually check the contents of cur_username.txt, questions.txt, answers.txt, and difficulty.txt to see if the variables were assigned correctly
Pre-Requisites:	• User is signed in as a instructor/developer • User presses new game • User presses developer mode
Expected Results:	• username = user inputted username • questions = content of question.txt file, without question number • answers = content of answers.txt file, without question number • difficulty = content of difficulty.txt file, without question number
Test Category:	Validation Test
Requirements:	The application will store all data locally
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	• username = user inputted username • questions = content of question.txt file, without question number • answers = content of answers.txt file, without question number • difficulty = content of difficulty.txt file, without question number
Remarks:	Test was passed and correctly accesses all question and user data

Test Case Name:	Test_Error_Ouput
Test Case Description:	Checks that anything in stdout is printed on screen in game
Test Steps:	1. Run developerMode.py 2. Print("Test Error") to stdout 3. Check if "Test Error" is printed on the in game screen
Pre-Requisites:	• User is signed in as a instructor/developer • User presses new game • User presses developer mode
Expected Results:	"Test Error" printed on the in game screen
Test Category:	Validation Testing
Requirements:	Debug Mode
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass

Test Results:	"Test Error" printed on the in game screen
Remarks:	Test passed, errors are correctly shown on screen

Test Case Name:	Test_Buttons
Test Case Description:	Checks to see if buttons are initialized correctly
Test Steps:	1. Set up button that just prints "pressed" to console 2. Run developerMode.py 3. Press button and check if "pressed" is in console
Pre-Requisites:	• User signed in as instructor/developer • User presses new game • User presses lightning mode
Expected Results:	"pressed" printed to console
Test Category:	Validation Test
Requirements:	Mouse Based Interaction
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	"pressed" printed to console
Remarks:	Buttons work

Test Case Name:	Test_Draggable_Objects
Test Case Description:	Checks to see if draggable objects are initialized correctly
Test Steps:	1. Run developerMode.py 2. Visually examine if the draggable objects on screen are from options.txt 3. Drag object around the screen to confirm that it is movable 4. Drag object over solution box, "collision" will be printed to console 5. Drag correct object over solution box, "correct" will be printed to console
Pre-Requisites:	• User is signed in as a player • User presses new game • User presses lightning mode
Expected Results:	• Object is movable • "collision" printed to console • "correct" printed to console
Test Category:	Validation Test
Requirements:	Mouse Based Interaction
Automation:	Manually Run by Human

Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	• Object is movable • "collision" printed to console • "correct" printed to console
Remarks:	Test passed, draggable object are implemented correctly

Test Case Name:	Test_Timer
Test Case Description:	Checks if the game session ends when timer is done
Test Steps:	1. Run developerMode.py 2. Wait for timer to time out 3. Check if lightning mode closes and main menu is opened
Pre-Requisites:	• User is signed in as a instructor/developer • User presses new game • User presses lightning mode
Expected Results:	Results shown, then lightning mode closes and main menu opens
Test Category:	Validation and Integration Test
Requirements:	Time-Based Elements
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	Results shown, then lightning mode closes and main menu opens
Remarks:	Lightning Mode timer works correctly

Landing Page

Test Case Name:	Test_Buttons
Test Case Description:	Checks to see if buttons are initialized correctly
Test Steps:	1. Set up button that just prints "pressed" to console 2. Run modemenu.py when user clicks "START GAME" 3. Run last level played in training mode when user clicks "LOAD GAME" 4. Run Settings.py when user clicks "SETTINGS" 5. Run Leaderboard.py when user clicks "LEADERBOARD" 6. Quit program when user clicks "QUIT" 7. Press button and check if "pressed" is in console
Pre-Requisites:	• User signed in as instructor/developer • User signed in as user • User presses "Proceed"
Expected Results:	"pressed" printed to console

Test Category:	Validation Test
Requirements:	Mouse Based Interaction
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	"pressed" printed to console
Remarks:	Buttons work

Mode Menu

Test Case Name:	Test_Buttons
Test Case Description:	Checks to see if buttons are initialized correctly
Test Steps:	1. Set up button that just prints "pressed" to console 2. Run trainingMode.py when user clicks "TRAINING MODE" 3. Run lightningMode.py when user clicks "LIGHTNING MODE" 4. Run instructorMode.py when user clicks "Instructor Mode" as instructor/developer 5. Run developerMode.py when user clicks "Developer Mode" as instructor/developer 6. Run tutorial.py when user clicks "TUTORIAL" 7. Goes back to modemenu.py when user clicks "BACK TO MAIN MENU" 8. Press button and check if "pressed" is in console
Pre-Requisites:	• User is signed in as a instructor/developer • User presses new game • User presses "START GAME" to run modemenu.py
Expected Results:	"pressed" printed to console
Test Category:	Validation Test
Requirements:	Mouse Based Interaction
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	"pressed" printed to console
Remarks:	Buttons work

Tutorial

Test Case Name:	Test_Buttons
Test Case Description:	Checks to see if buttons are initialized correctly

Test Steps:	1. Set up button that just prints "pressed" to console 2. Go to next page when user clicks "NEXT" 3. Go to previous page when user click "PREVIOUS" 4. Goes back to modemenu.py when user clicks "BACK TO MAIN MENU" 5. Press button and check if "pressed" is in console
Pre-Requisites:	• User is signed in as a instructor/developer • User presses new game • User presses "START GAME" to run modemenu.py • User presses "TUTORIAL" in modemenu.py to run tutorial.py
Expected Results:	"pressed" printed to console
Test Category:	Validation Test
Requirements:	Mouse Based Interaction
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	"pressed" printed to console
Remarks:	Buttons work

Instructor Mode + add Question Page

Test Case Name:	Test_Buttons on Instructor Screen
Test Case Description:	Checks to see if buttons functionality works as expected
Test Steps:	1. Click Next Question button 2. Click Prev Question button 3. click See Answer button 4. click on Add Question Button
Pre-Requisites:	• User is signed in as an instructor user • User presses new game • User presses "START GAME" to run modemenu.py • user is on the instructor mode screen
Expected Results:	1. When Next Question button is clicked the screen moves to the next question 2. When Prev Question button is clicked the screen moves to the previous question 3. When See Answer button is clicked the correct solution is prompted on the screen 4. click Add Question Button is clicked a new window for the add question screen opens overtop of the instructor mode screen
Test Category:	Integration Tests
Requirements:	Mouse based interaction
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	All button functionality works as expected

Remarks:	Buttons work

Test Case Name:	Test_Buttons Test_Text_Boxes on Add Question Screen
Test Case Description:	Check to see if button and text boxes functionality works as expected
Test Steps:	1. Click the Submit button when all text boxes are not filled in 2. Click the submit button when all the text boxes are filled in 3. click the return button 4. check to see all text boxes accept input
Pre-Requisites:	• User is signed in as an instructor user • User presses new game • User presses "Instructor Mode" to run instructorMode.py • user is on the instructor mode screen • user pressed Add Question button to launch addQuestion.py
Expected Results:	1. when the submit button is pressed when all the textboxes are not filled in the submit action does not work all text boxes must have input 2. when all the text boxes are filled in and the user hits submit this should clear all the text boxes 3. when the user clicks on return button the add question window should close 4. the user should be able to type input in all of the text boxes
Test Category:	Integration Tests
Requirements:	Mouse based interaction + keyboard input
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	All button functionality works as expected all text box functionality works as expected
Remarks:	Buttons work and text boxes work

Test Case Name:	Test_Write_Questions
Test Case Description:	checks that the user can properly add questions to the question files
Test Steps:	1. properly fill in the text boxes in the add Question screen based on the text box descriptions. Question description, question solution, question difficulty, option 1, option 2, option 3, option4 2. hit submit button
Pre-Requisites:	• User is signed in as an instructor user • User presses new game • User presses "Instructor Mode" to run instructorMode.py • user is on the instructor mode screen • user pressed Add Question button to launch addQuestion.py • user should have ability to open questions.txt, answers.txt, options.txt and difficulty.txt to view input.

Expected Results:	1. the input written to question description text box gets written to questions.txt in the form of "n. input typed" (here n represents the newest question number) on the next newline
	2. the input written to the Question solution text box gets written to answers.txt in the form of "n. input typed" (here n is representing the newest question number) on the next newline
	3. the input written to the question difficulty text box gets written to difficulty.txt in the form of "n. input typed" (here n is representing the newest question number) on the next newline
	4. the input written to the 4 options text boxes gets written to options.txt in the form of
	`"n.`
	`option 1 input text`
	`option 2 input text`
	`option 3 input text`
	`option 4 input text"`
	`(here n is representing the newest question number) on the next newline`

Test Category:	validation Tests
Requirements:	Mouse-based interaction + keyboard input
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	All question field inputs are written and formatted correctly to the appropriate question-related file
Remarks:	add question functionality works correctly

Leaderboard Page

Test Case Name:	Test_Buttons on Leaderboard screen
Test Case Description:	checks that buttons on leaderboard screen work properly
Test Steps:	1. Click the Back button
Pre-Requisites:	• The user is signed in from the sign-in page • from the main menu page, the user selects and clicks on the Leaderboard screen
Expected Results:	`1. when user clicks on the back button the user is returned to the main menu screen`
Test Category:	integration Tests
Requirements:	Mouse-based interaction
Automation:	Manually Run by Human
Date Run:	Mar. 29, 2024
Pass/Fail:	Pass
Test Results:	Button functionality works user is returned to the main menu when the click the back button
Remarks:	back button functionality works correctly

4. Summary

The testing documentation for the CS2212 group project offers a systematic approach to validating the functionality and educational effectiveness of our logic-based game. The testing framework is carefully designed to examine each aspect of gameplay, ensuring a robust and user-centric learning experience. The documentation provides a structured overview, beginning with an introduction, all the test plans for the unit testing, integration testing, validation testing, and system testing that make sure all the aspects of our game works.

The test plan include details of our 'Lightning Mode', 'Developer Mode', 'Training Mode', 'Instructor Mode', 'Sign-in', 'Leaderboard Page', 'Landing Page', 'Tutorial Page' pages. This documentation not only serves as a guide for the team to ensure that every feature is rigorously tested but also acts as a record of testing activities. By detailing the outcomes of each test case, the documentation facilitates the identification of any errors, usability issues, and areas for improvement, enabling the team to make informed decisions about modifications and enhancements. Including various modes within the testing plan reflects a comprehensive approach to assess the game's functionality across different user scenarios, ensuring the game remains responsive, stable, and engaging across all functionalities.