

Name: Ibrahim Khalilullah

ID: F2020266029

1. **Implement Software Testing Life Cycle (STLC) Phases:**

STLC Phases for Skribbl.io Test Case Development are as follows

Requirement Analysis

* In this phase we will gather as much information regarding the application we are about to test and understand the functional & non-functional requirement of our application i.e. core gameplay, gameplay loop, advanced features and etc.

Test Planning

* In this phase we will define the scope of the testing regarding the application and the strategy we will utilize to achieve results from our tests.

Test Case Development

* In this phase we will create detailed test cases for our application that covers all game functionality, advanced features, edge cases and scenarios

Test Environment Setup

* In this phase we will setup the test environment to execute our tests.

Test Execution

* In this phase we will execute the test cases and report defects if any i.e. functional, non-functional, cross platform

Test Closure

* In this phase we will review test results, ensure all bugs are resolved, and document lessons learned

1. **Requirements Analysis:**

**Software Requirement Specification:**

* The software should allow users to join servers may they be public or private
* The user can customize their avatar and create a name
* The users can choose the language of their choosing
* The users can draw and others users should be able to guess it to earn points, if the users are unable to guess the drawer earns points.
* The software should reveal some words with some time interval to help the guessers
* The users could customize to draw according to their desire.
* Users should be able to leave game session whenever they want.
* The software should allow users to invite anyone to their session.
* The host users should have the power to kick anyone from the session.
* In the end the user with the most points wins.

**Functional Requirements:**

* Draw
* Erase
* Join public/private lobby
* Type to guess
* Earn points
* Invite Users
* Create lobby
* Customize avatar
* Play
* Private Lobby
* Like/Dislike
* Rounds
* User capacity
* Time
* Number of hints
* Choose colors to draw

**Non-Functional Requirements:**

* Performance
* Usability
* Reliability
* Security
* Compatibility
* Load Handling
* Localization
* Maintainability

1. **Implement Testing Strategies.**

* **White Box Testing:**

White box testing focuses on the internal working of the software and ensures that underlying code is correctly functioning. Strategies we can implement for successful white box testing are Unit testing, Control flow testing and data flow testing.

* **Black Box Testing:**

Black box testing focuses on testing the functionality of the application without considering the internal code structure. Strategies we can implement for successful black box testing are functional testing, Usability Testing, compatibility testing and localization testing.

* **Grey Box Testing:**

Grey box testing combines both white box and black box techniques, testing the internal logic with some knowledge of the codebase while also focusing on functional behavior. Strategies we can implement for successful grey box testing are Database & API testing, Session Management testing, Security testing and Integration testing.

**Unit Testing:**

* ***Unit test for Username Field:***

**-**Description**:** Verify that user can enter their name in the field

-Pre-Condition: Text field should be working

-Action: Call ChooseName ()

-Outcome: Users can pick their name before joining the game.



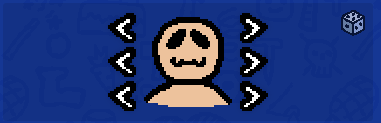
* ***Unit Test for Player Customization:***

-Description: Verify that users can customize their avatar to their liking.

-Pre-Condition: Customization functionality should be working

-Action: Call CharacterCustomization ()

-Outcome: Users can customize their character before joining the game



* ***Unit Test for Play button:***

-Description: Verify that user can press the Play button to join the game.

-Pre-Condition: Play button should be functioning correctly.

-Action: Call PlayButton ()

-Outcome: Users can press the play button and join a game session



* ***Unit Test for Private Room button:***

-Description: Verify that users can press the Private Room button to create or join a private game session.

-Pre-Condition: Private Room button is functioning correctly.

-Action: Call PrivateRoomButton ()

-Outcome: Users can press the Private Room button to create or join private game sessions.



* ***Unit Test for Type to Guess:***

-Description: Verify that users can type in the chat box to communicate and guess the word

-Pre-Condition: There should be a word to guess or players to communicate with

-Action: Call TypeToGuess()

-Outcome: Users can type in the chat box to communicate or guess the word



* ***Unit test for Draw:***

-Description: Verify that user can draw and other player have to guess the word

-Pre-Condition: It should be user turn to draw

-Action: Call Draw()

-Outcome: User can draw and other users have to guess in a time period and If they fail to guess the user drawing is awarded points.

* ***Unit test for Score:***

-Description: Verify that users get awarded with points when guessing correctly.

-Pre-Condition: A word should be guessed correctly

-Action: Call AwardScore()

-Outcome: Player get awarded with points when guessing correctly and get a total score in the end to decide the winner.



* ***Unit Test for Time Remaining:***

-Description: Verify that player guess the word in a specific time period or they don’t get points.

-Pre-Condition: Guessing period should have begun

-Action: Call TimeRemaining()

-Outcome: Users have to guess the word before the time runs out.



* ***Unit test for setting button:***

-Description: Verify that settings are opened when user clicks on the setting button

-Pre-Condition: Setting button should be working

-Action: Call SettingButton()

-Outcome: When user clicks on the setting button settings are opened.



* ***Unit test for Player drop down field:***

-Description: Verify that user can specify number of players in a game session

-Pre-Condition: It is set to 8 by default and minimum could be 2

-Action: Call NumOfPlayer()

-Outcome: User can specify number of players in a game session with minimum being 2 and maximum being 20.



* ***Unit test for Language drop down field:***

-Description: Verify that users can set their desired language

-Pre-Condition: Language should be selected and default would be English

-Action: Call SelectLanguage()

-Outcome: Users can select their desired language.



* ***Unit test for Draw Time drop down field:***

-Description: Verify that user can select the draw time period.

-Pre-Condition: Draw time period is set to 80 by default.

-Action: Call DrawTime()

-Outcome: User can set the limit for Draw Time with minimum being 15 sec and maximum being 240 sec.



* ***Unit test for Rounds:***

-Description: Verify that user can set the number of rounds in the match.

-Pre-Condition: Number of rounds should be set to 3 by default

-Action: call SetNumRounds()

-Outcome: User can set number of rounds after which the winner is decided



* ***Unit test for Word Mode:***

-Description: Verify that user can select multiple word modes

-Pre-Condition: Word mode should be set to Normal by default

-Action: Call Wordmode()

-Outcome: User can set the word mode to their liking.



* ***Unit test for Word Count:***

-Description: Verify that users can set the word count to their liking.

-Pre-Condition: Word count should be set to 3 by default

-Action: Call SetWordCount()

-Outcome: User can set word count to minimum 1 and maximum 5



* ***Unit test for Hints:***

-Description: Verify that user can set number of hints during a round.

-Pre-Condition: Number of hints should be set to 2 by default.

-Action: Call SetHints()

-Outcome: User can set the number of hints in the duration of round.



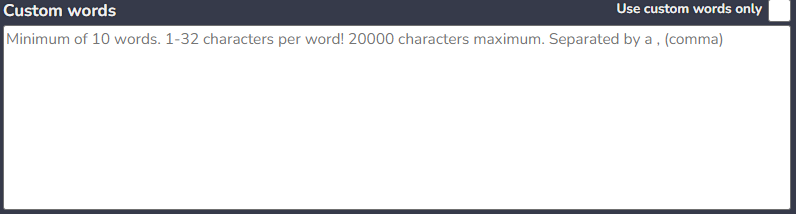
* ***Unit test for Custom Field:***

-Description: Verify user can custom words which are minimum of 10 words,1-32 characters per word

-Pre-Condition: Custom word field is empty and is not compulsory

-Action: call CustomWord()

-Outcome: User can set Custom words for the entirety of the game session.



* ***Unit test for Start Button:***

-Description: Verify that when user click Start button game begins

-Pre-Condition: Start button should be functioning correctly.

-Action: Call Startbutton()

-Outcome: User can start the game session by pressing the start button.



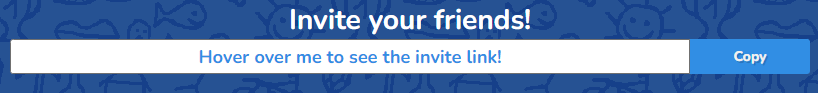
* ***Unit test for Invite Friends:***

-Description: Verify that user can invite their friends by generating join link for the game session

-Pre-Condition: User have to start a private session to invite their friends.

-Action: Call InviteFriendLink()

-Outcome: User can invite their friends to their private match via an invite link.



**Integration Testing:**

* ***Integration test from Username to Avatar:***

-Description: Verify that user can set their name and Avatar as they desire.

-Pre-Condition: Username cannot be empty

-Action: Call UsernameSet() & AvatarCustomization()

-Outcome: User can seamlessly transition from Username Creation to Avatar Customization.

* ***Integration test from Play to Game Session:***

-Description: Verify that when user clicks on Play button they can join an existing Game session

-Pre-Condition: Play button should be working and game server should be active

-Action: Call PlayButton() & JoinSession()

-Outcome: Users can seamlessly transition from Main menu to game by pressing the Play button.

* ***Integration test for guessing to score:***

-Description: Verify that users get score for correctly guessing a word

-Pre-Condition: A word should be available to guess

-Action: Call CorrectGuess() & Score()

-Outcome: User get score points for correctly guessing a word.

* ***Integration test from Create Private Room to Private Room setting:***

-Description: Verify that pressing the Create private room button transition to private room setting before creating a private session

-Pre-Condition: Create private room button works and private room settings are set to default.

-Action: Call CreatePrivateRoomButton() & PrivateRoomSettings()

-Outcome: User transition from main menu to private room setting after clicking create private room.

**Incremental Integration Testing:**

* ***Incremental Integration from Main menu to Game Session:***

-Description: Verify that user can interact with the main menu and can customize their name and character and when they press Play they can join an existing game session

-Pre-Condition: Functionality in the main menu should be functioning properly.

-Action: Call SetUsername(), AvatarCustomization() & PlayButton()

-Outcome: User can transition from main menu to game session successfully after using functionality in the main meu.

* ***Incremental Integration from Game Session to End Game Session:***

-Description: Verify that after the rounds are complete game session ends after deciding the winner of the session.

-Pre-Condition: User can enter the game session.

Action: Call Draw(), Score() & Winner()

-Outcome: Users can use the functionality in the game session till the game session ends and the winner is decided.

* ***Incremental Integration from Main menu to Create Private Room:***

-Description: Verify that users can transition from main menu to private room after they press the create private room session.

-Pre-Condition: Main menu functionality and Create Private Room functionality should be functioning.

-Action: Call CreatePrivateRoom()

-Outcome: User can transition from Main menu to private room when they press the create private room button.

**System Testing:**

* ***Complete Game flow testing:***

-Description: Verify that user can get from the start of the game to end without any disruption to the game flow loop.

-Pre-Condition: Game functionality should be functioning.

-Action: Call GameManager()

-Outcome: User can get from main menu to the game session and from there to the end of game session without breaking the loop.

* ***Multiplayer Integration testing:***

-Description: Verify that multiple users can join a game session without performance degradation.

-Pre-Condition: Player should be able to join the game session

-Action: Call JoinGame()

-Outcome: Users can join the game session without any degradation to the performance.

* ***Game Setting Testing:***

-Description: Verify that users can create custom games i.e private lobbies and customize to their liking and the game should behave accordingly to the setting during gameplay

-Pre-Condition: User can create custom games

-Action: call createPrivateRoom()

-Outcome: Users can create custom games and customize it to their liking and the game behaves accordingly to the setting applied during the gameplay.

**Load Testing:**

* ***Concurrent Users:***

-Description: Verify that a set number of users are trying to connect and create rooms i.e 50,000 users and the system can handle the load without crashing

-Pre-Condition: Users should be able to connect to the website to connect and create rooms

-Action: Call joinRoom() & CreateRoom()

-Outcome: A set number of users can connect and create rooms without system crashing.

**Regression Testing:**

* ***Player Score System testing:***

-Description: Verify that after modifying the word selection algorithm, verify that the scoring system still works correctly.

-Pre-Condition: Users modify the word selection algorithm by choosing custom words.

-Action: Call ChooseWord() & CustomWord()

-Outcome: After modifying the word selection algorithm, the scoring system works correctly.

* ***Room Creation and Joining testing:***

-Description: Verify that updates to chat functionality do not affect the ability to create or join game rooms.

-Pre-Condition: User should use chat functionality before testing the ability to join game

-Action: Call Chat()

-Outcome: Updates to chat functionality do not affect the ability to create or join game rooms.

**Deployment Testing:**

* ***Staging Environment Deployment testing:***

-Description: Verify that the game works on a staging server, with no issues related to server configuration or deployment scripts.

-Pre-Condition: Servers should be available.

-Action: Call SystemServer(), ServerConfig()

-Outcome: The game works on a staging server, with no issues related to server configuration or deployment scripts

* ***Production Deployment testing:***

-Description: Verify that after deployment to production, all services (game servers, WebSocket connections, etc.) are running smoothly and the application is accessible to all players.

-Pre-Condition: Game should deployed on servers

-Action: Call Deploy(), CreateBuild()

-Outcome: After deployment to production, all services (game servers, WebSocket connections, etc.) are running smoothly and the application is accessible to all players.

**Validation/Invalidation Testing:**

* ***Validation Testing for Code creation:***

-Description: Verify that when valid room codes are entered, players can join the correct game room.

-Pre-Condition: Valid Codes are generated to enter the room

-Action: Call CreateRoomToken()

-Outcome: When valid room codes are entered, players can join the correct game room

* ***Validation Testing for Score Calculation:***

-Description: Verify that score calculation happens correctly when the guessing time is different for -various players.

-Pre-Condition: Players have to guess in the given time to get score

-Action: Call Score(), CalculateScore()

-Outcome: Score calculation happens correctly when the guessing time is different for various players.

* **Invalidation Test for Code Creation:**

-Description: Verify that attempt to join a room with an invalid or expired code and ensure an appropriate error message is shown.

-Pre-Condition: Invalid code is entered to join the room.

-Action: Call displayError()

-Outcome: Attempt to join a room with an invalid or expired code and ensure an appropriate error message is shown.

* **Invalidation Test for invalid/Offensive Words:**

-Description: Verify that Inputting an invalid or offensive word in a custom word list and check if it gets rejected.

-Pre-Condition: Invalid Word should be typed to generate this message.

-Action: Call DisplayError()

-Outcome: Inputting an invalid or offensive word in a custom word list and check if it gets rejected.

1. **Apply Quality Principles:**

**Garvin's Quality Factors:**

* ***Performance:***

- Testing Type: Functionality Testing

- Objective: Ensure that the content software system deliver all content, functions, and options that are such as a part of the system

- Example Test: The game play loop and complete game flow is according to the requirements.

* ***Features:***

-Testing Type: System Testing

-Objective: Ensure that all core features (word selection, drawing, chat, scoring) function correctly and meet user requirements.

-Example Test: Validate that all drawing tools work correctly, and scoring is calculated as intended.

**McCall’s Quality Factor:**

* ***Product Operation Testing:***

-Conduct load and performance testing to ensure that Skribbl.io operates efficiently with large player counts, real-time updates, and smooth user interaction.

-Perform functional and usability tests to ensure correct behaviour of gameplay features, as well as security tests to maintain data integrity.

* ***Product Revision Testing:***

Emphasize code maintainability by ensuring modular code architecture that can easily accommodate new features (e.g., additional game modes).

Implement flexibility tests to allow customization and configuration of gameplay by users, such as altering game settings.

* ***Product Transition Testing:***

Ensure cross-platform compatibility by running the game on various devices and browsers.

Implement API integration tests to validate the game's ability to interact with external systems and ensure smooth interoperability between components.

**ISO Quality Factors:**

ISO’s software quality factors focus on characteristics like functionality, reliability, usability, efficiency, maintainability, and portability.

**Testing Incorporation:**

* ***Functionality:***

-Testing Type: Functional Testing

-Objective: Ensure that all features, such as drawing, guessing, and scoring, work as specified.

-Example Test: Validate that guesses are recorded correctly and that points are awarded as per the defined rules.

* ***Reliability:***

-Testing Type: Stress & Load Testing

-Objective: Ensure the game remains reliable under stress conditions, like high player counts or extended play.

-Example Test: Simulate a large number of players joining and leaving the game room to test server stability.

* ***Usability:***

-Testing Type: Usability Testing

-Objective: Ensure that the interface is easy to navigate and understand.

-Example Test: Evaluate the ease of joining a game, selecting a word, and starting a drawing round.

* ***Efficiency:***

-Testing Type: Performance Testing

-Objective: Measure how efficiently the game uses resources (e.g., CPU, memory).

-Example Test: Test how much CPU usage increases when multiple players join a room.

* ***Maintainability:***

-Testing Type: Code Maintainability Testing

-Objective: Ensure the system can be easily modified or extended.

-Example Test: Verify that new features, like adding extra drawing tools, can be easily integrated into the existing codebase.

* ***Portability:***

-Testing Type: Cross-Browser & Cross-Device Testing

-Objective: Ensure that the game functions correctly across different browsers and devices.

-Example Test: Ensure compatibility with popular browsers like Chrome, Firefox, Safari, and Edge.

1. **Develop and Execute Test Cases:**

**5.1. Username & Avatar Customization**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail | Pre-condition | Post-condition | Notes |
| TC\_1.1 | Check that user can setup a username before joining or creating a game session | 1.Click on username field.  2. Type desired username.  3. Username should be appropriate. | Username = unique\_user\_123 | User should be able to setup a username. | As expected. | Pass | User have selected the username text field. | User have typed valid username in the username field. | User should have appropriate name. |
| TC\_1.2 | Check that user can customize their character. | 1.After Selecting username, user can click on avatar to customize.  2. User can customize the color etc. | Avatar customization in progress. | User should be able to customize avatar to their liking. | As expected. | Pass | User have selected the avatar customization part. | User have customized their avatar. | User should have selected avatar customization. |

**5.2. Game Lobby**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail | Pre-condition | Post-condition | Notes |
| TC\_2.1 | Check if a user can create a new game lobby. | 1. Launch Skribbl.io.  2. Click "Create Private Room".  3.Customize settings and click "Create Room". | Game Settings: 5 rounds, Custom Word Pack | A new game lobby should be created with the chosen settings. | As expected. | Pass | User is on the game lobby page. | A private game lobby is created with the selected settings. | Test various combinations of game settings (rounds, word packs). |
| TC\_2.2 | Check if a user can join an existing game lobby using an invite link. | 1. Copy invite link. 2. Open the link in a browser.  3. Click "Join Game". | Invite Link: skribbl.io/abcd123 | User should successfully join the lobby. avatar to their liking. | As expected. | Pass | The game lobby is already created, and an invite link is available. | User is successfully in the game lobby. | Ensure the link is valid and accessible before testing. |

**5.3.Gameplay**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail | Pre-condition | Post-condition | Notes |
| TC\_3.1 | Check if a player can draw a given word. | 1.Click on username field.  2. Type desired username.  3. Username should be appropriate. | Username = unique\_user\_123 | User should be able to setup a username. | As expected. | Pass | User have selected the username text field. | User have typed valid username in the username field. | User should have appropriate name. |
| TC\_3.2 | Check if other players can guess the word being drawn. | 1. Another player draws a word.  2. Observe the drawing and type a guess.  3. Submit the guess. | Guess = "Cat" | If the guess is correct, the system should display a message and award points. | As expected. | Pass | A game has started, and another player is drawing. | User's guess is validated, and points are awarded if correct. | Verify system performance when multiple users guess at the same time. |

**5.4. Chat Functionality**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail | Pre-condition | Post-condition | Notes |
| TC\_4.1 | Check if users can send chat messages during the game. | 1.Click on username field.  2. Type desired username.  3. Username should be appropriate. | Username = unique\_user\_123 | User should be able to setup a username. | As expected. | Pass | User have selected the username text field. | User have typed valid username in the username field. | User should have appropriate name. |
| TC\_4.2 | Check if the chat filters inappropriate words. | 1. Type an inappropriate word into the chat.  2. Press "Send". | Message = inappropriate word | The system should filter the word and display asterisks instead. | As expected. | Pass | User is in an active game with chat functionality enabled. | The message is filtered, and inappropriate words are replaced with asterisks. | Test different types of inappropriate words to ensure the filter works correctly. |

**5.5. Post Game**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail | Pre-condition | Post-condition | Notes |
| TC\_5.1 | Check if the leaderboard is displayed after the game ends. | 1. Complete all rounds of the game.  2. Wait for the leaderboard to appear. | Leaderboard displaying | The leaderboard should display scores of all players, and the winner should be highlighted. | As expected. | Pass | A game with multiple rounds has completed. | The leaderboard is displayed, and the final winner is shown. | Ensure that players with tied scores are handled appropriately. |
| TC\_5.2 | Check if players can start a new game after completing the previous one. | 1. After the game ends, click "Play Again". | New game after previous completion | The game should reset, and players should be moved to a new game lobby. | As expected. | Pass | A game has been completed. | A new game session is initiated. | Test if all players are moved to the new game lobby and previous game data is cleared. |

**5.6. Game Customization**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail | Pre-condition | Post-condition | Notes |
| TC\_6.1 | Check if a user can customize the number of rounds in a private room. | 1. Create a private room.  2. Set number of rounds to 10.  3. Start the game. | Number of Rounds = 10 | The game should play for 10 rounds. | As expected. | Pass | User is in the "Create Private Room" settings menu. | The game begins with 10 rounds as configured. | Test various round configurations (e.g., 3, 5, 15 rounds). |
| TC\_6.2 | Check if a user can add a custom word list to the game. | 1. Create a private room.  2. Upload a custom word list.  3. Start the game. | Word List: (Python, Java, HTML) | he words in the custom list should appear during gameplay. | As expected. | Pass | User has prepared a custom word list | Custom words are successfully included | Ensure that only valid words (no special characters) are included. |

**5.7. Player Management**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail | Pre-condition | Post-condition | Notes |
| TC\_7.1 | Check if a host can kick a player from the game lobby. | 1. Create a private room and invite players.  2. Click on the "Kick" button next to a player's name. | Player = "Player123" | The player should be removed from the game lobby and cannot rejoin. | As expected. | Pass | Host has created a private room and players have joined. | The player is removed from the game, and the remaining players continue. | Ensure the host role has sufficient permissions to kick players during the game. |
| TC\_7.2 | Check if a user can mute another player in the chat. | 1. Start the game.  2. Another player spams the chat.  3. Click "Mute" next to the player's chat message. | Player = "Spammer123" | The muted player's messages should no longer appear in the user's chat. | As expected. | Pass | User is in an active game where chat functionality is enabled. | The player is muted, and future messages are hidden from view. | Test muting and unmuting functionality across multiple rounds |

**5.8. Setting & Game Option**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail | Pre-condition | Post-condition | Notes |
| TC\_8.1 | Check if the game supports changing language settings. | 1. Go to the settings menu.  2. Change the language to "Spanish". 3. Start a new game. | Language = Spanish | The interface and words should be in the selected language. | As expected. | Pass | User is in the settings menu of Skribbl.io. | Game and UI are presented in the new language setting. | Ensure all supported languages function as expected. |
| TC\_8.2 | Check if players can enable or disable drawing assistance (guidelines). | 1. Go to settings.  2. Toggle drawing assistance on/off.  3. Start a new game and begin drawing. | Drawing Assistance = On/Off | Drawing guidelines should appear (or disappear) based on the setting. | As expected. | Pass | User is in the settings menu before the game starts. | The drawing assistance behaves as per the setting configuration. | Test this feature with various drawing complexities to check usability. |

**5.9. In-Game Score & Leaderboards**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail | Pre-condition | Post-condition | Notes |
| TC\_9.1 | Check if points are correctly awarded based on guessing speed. | 1. Start a game with multiple players.  2. Player guesses the word quickly.  3. Check points awarded. | Guess Time = 3 seconds | The player should be removed from the game lobby and cannot rejoin. | As expected. | Pass | Host has created a private room and players have joined. | The player is removed from the game, and the remaining players continue. | Ensure the host role has sufficient permissions to kick players during the game. |
| TC\_9.2 | Check if the final leaderboard reflects accurate scores. | 1. Complete the game rounds.  2. Wait for the final leaderboard to appear. | Player1= 1000 points  Player2=900 points | The leaderboard should accurately show player rankings based on total points. | As expected. | Pass | Game has concluded, and the final round is completed. | Leaderboard displays the final ranking and scores of all players. | Ensure any ties are handled properly in the ranking display. |

**5.10. Mobile Compatibility**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail | Pre-condition | Post-condition | Notes |
| TC\_10.1 | Check if the game runs smoothly on mobile browsers. | 1. Open Skribbl.io on a mobile browser.  2. Join or create a game.  3. Play a few rounds. | Device: Mobile browser (Chrome/Safari) | The game should run smoothly without lag, and the interface should be responsive. | As expected. | Pass | User is using a mobile browser. | Game runs smoothly, and no visual issues occur on mobile. | Test the interface on different mobile devices and browsers for compatibility. |
| TC\_10.2 | Check if drawing functionality works smoothly on mobile touch screens. | 1. Join a game on mobile.  2. Draw the given word using touch controls. | Device: Touchscreen | Drawing tools should work properly, and players should be able to draw accurately using touch. | As expected. | Pass | User is using a mobile touch screen to play the game. | Drawing experience is smooth, and drawings are rendered correctly. | Test the drawing functionality on both small and large screen sizes. |

1. **Identify potential errors/bugs or errors if any:**

**Final Report:**

**Testing Process:**

* **Project Overview**

Skribbl.io is a real-time multiplayer drawing and guessing game where players draw a word and others try to guess it. Given its interactive and collaborative nature, the project required extensive testing across various domains, including functionality, performance and user experience.

The testing process for the application was thorough and it went from multiple phases like Unit testing, Integration testing, Incremental Integration testing, system testing, load testing, regression testing, Deployment testing and Validation/Invalidation testing.

* ***Unit Testing:***

In unit testing, We Tested individual components or units of the game to ensure that each one works as expected. Unit tests focus on isolated parts of the code, like functions or methods.

Example: Drawing Tools, Avatar Customization etc.

* ***Integration Testing:***

In Integration testing, the interaction between different modules is tested to ensure that they work together correctly. These tests focus on data flow between different components

Example: Drawing and Guessing Interaction, Room Management etc.

* ***Incremental Integration testing:***

In Incremental Integration testing, we tested individual modules as they are integrated one by one. Incremental testing is done gradually, ensuring that every module works well with previously integrated modules.

Example: Adding Player in a Room, Game Session State

* ***System Testing:***

Tested the complete end-to-end functionality of the game under typical user scenarios.

* ***Load Testing:***

Evaluated how the game performed under high traffic and concurrent user loads.

* ***Regression Testing:***

Ensured that new changes and bug fixes did not introduce new defects or affect existing functionalities.

* ***Deployment Testing:***

Validated the installation and deployment process across different environments.

* ***Validation/Invalidation Testing:***

Verified that the game adheres to its requirements and functional specifications.

**Testing Outcomes**

The following is a summary of the testing outcomes:

* ***Functional Bugs:***

Minor issues were identified in word selection, score calculation, and round timing.

Gameplay mechanics worked well overall, but some mobile-related bugs were fixed, improving user experience.

* ***Performance Issues:***

Under heavy loads, the game experienced slower response times and occasional server disconnections.

Performance optimizations are recommended for handling large player counts

**Recommendations for Future Improvements**

* ***Performance Optimization:***  
  Address server performance under high traffic by optimizing resource management and considering the use of distributed servers to handle increased loads.
* ***Mobile Usability:***  
  Improve touch responsiveness for mobile users, especially with the drawing tools. Adjust the UI layout to better fit smaller screens and enhance ease of navigation.

**Conclusion**

The testing process revealed that Skribbl.io is a well-designed game with a stable core, though it requires improvements in performance and mobile usability. With additional optimizations and continued monitoring, the game can be scaled effectively while maintaining a high-quality player experience.