1. WinRunner

Introduction

WinRunner is an automated software testing tool developed by Mercury Interactive (now owned by HP) used to test the functionality of applications. It records user actions, creates test scripts, and replays them to check if an application works as expected.

Theory Explanation

WinRunner automates functional testing by capturing user interactions with an application's graphical user interface (GUI) and generating test scripts in a C-like language called Test Script Language (TSL). It records actions like clicking buttons or entering text and replays them to verify results against expected outcomes. It supports testing for web and desktop applications built in languages like C, Java, and Visual Basic. WinRunner includes features like checkpoints to validate application behavior and a recovery manager to handle unexpected errors during testing.

Key Points

- **Automation**: Records and replays user actions to test applications.
- **Test Script Language (TSL)**: Uses a C-like language for scripting.
- **Checkpoints**: Verifies UI elements, data, or functionality.
- **Recovery Manager**: Handles errors like crashes or pop-ups automatically.
- **Supported Languages**: Tests apps in C, Java, VB, HTML, etc.

Real-life Example

Imagine testing a banking app to ensure the login button works. WinRunner records you entering a username and password, clicking login, and checks if the dashboard loads correctly, saving time over manual testing.

Advantages & Disadvantages

Advantages:

- Automates repetitive functional testing, reducing time.
- Supports multiple programming languages and web/desktop apps.
- Easy to record and replay test scripts.

Disadvantages:

- Requires a paid license (not open-source).
- Limited to Windows; no support for macOS/Linux.

- Less popular today compared to modern tools like Selenium.

Applications

- **Functional Testing**: Test login, forms, or navigation in apps.
- **Regression Testing**: Verify new updates don't break existing features.
- **GUI Testing**: Check UI elements like buttons or text fields.

Exam-Focused Summary

- **WinRunner**: Paid tool for functional testing of web/desktop apps.
- **Key Features**: Records/replays actions using TSL, supports checkpoints, recovery manager.
- **Use Case**: Automate testing of banking app login.
- **Pros**: Easy to use, supports many languages.
- **Cons**: Paid, Windows-only, less used today.

1)Write a Selenium program using Node.js to automate a Google search.

```
const { Builder, By, Key } = require('selenium-webdriver');

async function googleSearch() {
    // Set up the Chrome browser
    let driver = await new Builder().forBrowser('chrome').build();

    try {
        // Open Google
        await driver.get('https://www.google.com');

        // Find the search bar by its name attribute
        await driver.findElement(By.name('q')).sendKeys('Artificial Intelligence', Key.RETURN);

        // Wait for 3 seconds to see the results
        await driver.sleep(3000);
    } catch (error) {
        console.log('An error occurred:', error);
    } finally {
        // Close the browser
        await driver.quit();
    }
}
googleSearch();
```

2)Create a Selenium script in Node.js to open a web page, extract the text

```
const { Builder, By } = require('selenium-webdriver');


async function extractText() {
    // Set up Chrome browser
    let driver = await new Builder().forBrowser('chrome').build();
    try {
        // Open Wikipedia page
        console.log('Opening Wikipedia...');
        await driver.get('https://en.wikipedia.org/wiki/Artificial_intelligence');
        await driver.sleep(5000); // Wait 5 seconds to observe
        // Find the first paragraph
        console.log('Finding paragraph...');
        let paragraph = await driver.findElement(By.css('p:not(:empty)')); // Use :not(:empty) to avoid empty <p> tags
        let text = await paragraph.getText();
        // Print the extracted text
        console.log('Extracted Text:', text);
        await driver.sleep(5000); // Wait 5 seconds before closing
    } catch (error) {
        console.log('An error occurred:', error.message);
    } finally {
        console.log('Closing the browser...');
        await driver.quit();
    }
}
extractText();
```

3)Write a Selenium script to handle browser navigation (back, Forward, refresh)

```
const { Builder } = require('selenium-webdriver');

async function browserNavigation() {
    // Set up Chrome browser
    let driver = await new Builder().forBrowser('chrome').build();

    try {
        // Step 1: Open Google
        console.log('Opening Google...');
        await driver.get('https://www.google.com');
        await driver.sleep(5000); // Wait 5 seconds to observe

        // Step 2: Navigate to Wikipedia
        console.log('Navigating to Wikipedia...');
        await driver.get('https://en.wikipedia.org/wiki/Main_Page');
        await driver.sleep(5000); // Wait 5 seconds

        // Step 3: Go back to Google
        console.log('Going back to Google...');
        await driver.navigate().back();
        await driver.sleep(5000); // Wait 5 seconds

        // Step 4: Go forward to Wikipedia
        console.log('Going forward to Wikipedia...');
        await driver.navigate().forward();
```

```
        await driver.sleep(5000); // Wait 5 seconds

        // Step 5: Refresh the current page
        console.log('Refreshing the page...');
        await driver.navigate().refresh();
        await driver.sleep(5000); // Wait 5 seconds
    } catch (error) {
        console.log('An error occurred:', error);
    } finally {
        // Close the browser
        console.log('Closing the browser...');
        await driver.quit();
    }
}

// Run the function
browserNavigation();
```

4)Write a Node.js program using Selenium to count the total number of hyperlinks on the page

```
const { Builder, By } = require('selenium-webdriver');
async function countHyperlinks() {
    // Set up Chrome browser
    let driver = await new Builder().forBrowser('chrome').build();
    try {
        // Open Wikipedia main page
        console.log('Opening Wikipedia...');
        await driver.get('https://en.wikipedia.org/wiki/Main_Page');
        await driver.sleep(5000); // Wait 5 seconds to observe
        // Find all hyperlinks (<a> tags)
        let hyperlinks = await driver.findElements(By.tagName('a'));
        // Count and print the total number of hyperlinks
        let totalLinks = hyperlinks.length;
        console.log(`Total number of hyperlinks: ${totalLinks}`);
        // Wait 5 seconds before closing
        await driver.sleep(5000);
    } catch (error) {
        console.log('An error occurred:', error);
    } finally {
        // Close the browser
        console.log('Closing the browser...');
        await driver.quit();
    }
}
// Run the function
countHyperlinks();
```

5)Develop a Selenium script to automate login testing

```
const { Builder, By } = require('selenium-webdriver');

async function loginTest() {
    // Set up Chrome browser
    let driver = await new Builder().forBrowser('chrome').build();

    try {
        // Step 1: Open the login page
        console.log('Opening login page...');
        await driver.get('https://the-internet.herokuapp.com/login');
        await driver.sleep(5000); // Wait 5 seconds to observe

        // Step 2: Enter username
        console.log('Entering username...');
        await driver.findElement(By.id('username')).sendKeys('tomsmith');
        await driver.sleep(5000); // Wait 5 seconds

        // Step 3: Enter password
        console.log('Entering password...');
        await driver.findElement(By.id('password')).sendKeys('SuperSecretPassword!');
        await driver.sleep(5000); // Wait 5 seconds

        // Step 4: Click the login button
        console.log('Submitting login form...');
```

```
        await driver.findElement(By.css('button[type="submit"]')).click();
        await driver.sleep(5000); // Wait 5 seconds

        // Step 5: Verify login success
        console.log('Checking login result...');
        let successMessage = await driver.findElement(By.css('.flash.success')).getText();
        console.log('Login Result:', successMessage.includes('You logged into a secure area') ? 'Success' : 'Failed');
        await driver.sleep(5000); // Wait 5 seconds before closing
    } catch (error) {
        console.log('An error occurred:', error);
    } finally {
        // Close the browser
        console.log('Closing the browser...');
        await driver.quit();
    }
}

// Run the function
loginTest();
```

6)Write Selenium script that captures a screenshot

```
const { Builder } = require('selenium-webdriver');
const fs = require('fs');
async function captureScreenshot() {
    // Set up Chrome browser
    let driver = await new Builder().forBrowser('chrome').build();
    try {
        // Step 1: Open Google
        console.log('Opening Google...');
        await driver.get('https://www.google.com');
        await driver.sleep(5000); // Wait 5 seconds to observe
        // Step 2: Capture screenshot
        console.log('Capturing screenshot...');
        let screenshot = await driver.takeScreenshot();
        fs.writeFileSync('screenshot.png', screenshot, 'base64');
        console.log('Screenshot saved as screenshot.png');
        // Wait 5 seconds before closing
        await driver.sleep(5000);
    } catch (error) {
        console.log('An error occurred:', error);
    } finally {
        // Close the browser
        console.log('Closing the browser...');
        await driver.quit();
    }
}
// Run the function
```

```
captureScreenshot();
```