Integration Testing

Table of Contents

Overview

We will use the Bottom-Up integration testing strategy, as the lower-level components had been written first. In addition, we will use thread-based testing based on various scenarios. As our code uses the Model-View-Controller architectural pattern, we will test the model classes separately from the view and controller classes. Drivers will be used where appropriate.

Tests

User Data Cluster

Includes the ActionBlockData (and ActionTypeEnum), ActionChainData, ProgressionData, and UserData (and UserTypeEnum) classes

Test Case Name	Create user data
Test Case Description	Data for a new user is created
Test Steps	1. Create a new user data object 2. Call the UserData.exportData() method to create a new user data file 3. Check the userdata folder to see if the file was properly created
Pre-Requisites	None
Expected Results	New file is created with the user type, username and password
Test Category	Integration test
Requirement	Functional requirement 5 a. The game has a save system that creates/ accesses independent save files to keep track of each student's progression.
Automation	Manual
Date Run	31 Mar 2024
Pass/Fail	Pass
Test Results	File was created
Remarks	Model test; driver used

Test Case Name	Save progress
Test Case Description	Update user progress
Test Steps	1. Create string representing progression 2. Call the ProgressionData.importData() method 3. Call UserData.addProgressionData() using the return value of 2 as parameter 4. Call the UserData.exportData() method 5. Check the file to see if progress was updated
Pre-Requisites	User data file exists

Expected Results	The file is properly updated
Test Category	Integration test
Requirement	Functional requirement 5 a. The game has a save system that creates/ accesses independent save files to keep track of each student's progression.
Automation	Manual
Date Run	31 Mar 2024
Pass/Fail	Pass
Test Results	File was successfully updated with the test string
Remarks	Model test; driver used

Progression Cluster

Includes the User Data Cluster, plus the StudentProgressionData and TeacherProgressionnData classes

Test Case Name	Load Teacher progression data
Test Case Description	All user data is loaded into the TeacherProgressionData class
Test Steps	1. Call the TeacherProgressionData constructor 2. Use the getUserData() method to check if the files are correctly imported
Pre-Requisites	At least one user data file exists
Expected Results	File contents match the original file
Test Category	Integration test
Requirement	Functional requirement 11. c. The progression screen displays the following statistics: (...)
Automation	Manual
Date Run	31 Mar 2024
Pass/Fail	Pass
Test Results	Username was printed
Remarks	Model test; driver used

Test Case Name	Load Student progression view
Test Case Description	A student logged in has all their progression displayed on the UI
Test Steps	1. Log in via the login screen 2. Click into the progression screen
Pre-Requisites	Log in was successful
Expected Results	All of the user's progression information are displayed
Test Category	Integration test
Requirement	Functional requirement 11. c. The progression screen displays the following statistics: (...)
Automation	Manual
Date Run	*30 Mar 2024*
Pass/Fail	Pass

Test Results	
Remarks	View & controller test using GUI

Test Case Name	Load teacher view
Test Case Description	Load progression of all students
Test Steps	1. Log in as a teacher 2. Click progression button
Pre-Requisites	User is logged in as a teacher,
Expected Results	All students' progression data are displayed as a list
Test Category	Integration test
Requirement	Functional requirement 11. c. The progression screen displays the following statistics: (...)
Automation	Manual
Date Run	30 Mar 2024
Pass/Fail	Pass
Test Results	All user files are loaded and displayed
Remarks	View & controller test using GUI

Login Cluster

Includes the User Data Cluster, plus the LoginData class.

Test Case Name	Register new user
Test Case Description	Create a new user data
Test Steps	1. Create a new LoginData object 2. Set username and password 3. Call the registerActiveUser() method 4. Use getActiveUserData() to check if the UserData object was created
Pre-Requisites	None
Expected Results	New UserData object created with username and password
Test Category	Integration test
Requirement	Functional requirement 7 g. The username and password are stored in each save file for identification.
Automation	Manual
Date Run	01 Apr 2024
Pass/Fail	Pass
Test Results	Correct username was printed
Remarks	Model test; driver used

Test Case Name	Get login data from view
Test Case Description	Get the values of Username and Password from UI elements

Test Steps	1. Go to the login screen, input username and password 2. Click login button
Pre-Requisites	Userdata class exists
Expected Results	Username and password loaded into a userdata instance and printed to console
Test Category	Integration test
Requirement	Functional requirement 7 d. The multi-user login screen prompts and stores the following login information: 1. Username 2. Password (custom student passwords, or unique secret passwords for teachers/ developers)
Automation	Manual
Date Run	30 Mar 2024
Pass/Fail	Pass
Test Results	Username and password outputted
Remarks	View & controller test using GUI

Gameplay Cluster

Includes the User Data Cluster, plus the MazeData class and the GameplayController class.

Test Case Name	Load gameplay view
Test Case Description	Load the correct maze and action chain from maze data and student data, display them on the view
Test Steps	1. login 2. select progression 3. select a maze
Pre-Requisites	If testing for an action chain, the user must have a saved action chain
Expected Results	Maze and saved action chain displayed on screen correctly
Test Category	Integration test
Requirement	Functional requirement 1. g. The gameplay has a visual display system to show the order of the action blocks.
Automation	Manual
Date Run	01 Apr 2024
Pass/Fail	Pass
Test Results	Saved action chain displayed with initial maze arrangement
Remarks	View & controller test using GUI

Test Case Name	Update Maze
Test Case Description	Update MazeView and MazeData based on the current state of the game
Test Steps	1. Load any level 2. Construct an action list 3. hit play
Pre-Requisites	Player can construct an action chain
Expected Results	The player character moves and acts according to the action chain. Movements are displayed in the maze

Test Category	Integration test
Requirement	Functional requirement 1. g. The gameplay has a visual display system to show the order of the action blocks.
Automation	Manual
Date Run	01 Apr 2024
Pass/Fail	Pass
Test Results	Player character acts in accordance with the player's action chain
Remarks	View & controller test using GUI

Test Case Name	Update action chain
Test Case Description	Load the correct action chain from file (if it exists) and display it in gameplayView,
Test Steps	1. Load any level 2. Add any number of various action blocks 3. Confirm that the added blocks are displayed correctly and ordered correctly
Pre-Requisites	User must be able to add action blocks manually
Expected Results	The added action blocks are appended to the bottom of the action list
Test Category	Integration test
Requirement	Functional requirement 1. g. The gameplay has a visual display system to show the order of the action blocks.
Automation	Manual
Date Run	01 Apr 2024
Pass/Fail	Pass
Test Results	Action blocks successfully displayed in the correct order
Remarks	View & controller test using GUI

High Score Cluster

Includes the HighScoreData and the HighScoreController classes

Test Case Name	Load high score view
Test Case Description	Display high scores
Test Steps	1. Select the high score button from main menu
Pre-Requisites	A high-score file exists
Expected Results	Names and scores are loaded into HighScoreData class and displayed in HighScoreView
Test Category	Integration testing
Requirement	Functional requirement 6. c. The high-score table is updated automatically each time the high-score table screen is opened.
Automation	Manual
Date Run	31 Mar 2024
Pass/Fail	pass
Test Results	

Remarks	At least 5 entries are loaded, if the high score file has < 5 entries, placeholder entries are displayed View & controller test using GUI

Test Case Name	Update high score view
Test Case Description	When a new high score has been reached by a player, save the player's name and the score
Test Steps	**1.** Enable debug mode **2.** Save a new high score **3.** Select high score button in main menu
Pre-Requisites	Game launched
Expected Results	The high score table should be updated with your name and score
Test Category	Integration testing
Requirement	Functional requirement 6. c. The high-score table is updated automatically each time the high-score table screen is opened.
Automation	Manual
Date Run	01 Apr 2024
Pass/Fail	Fail
Test Results	High score table unchanged, despite the student progression screen showing the highest score
Remarks	View & controller test using GUI