**Development Section**

**Prototype 1 - Main Page**

**Design**

**Overview:**Overall, I intend to design the button-based navigation workflow necessary for users to navigate in between the different forms (pages). This section will therefore have a flow of what the user can do in the program I’m creating, the creation of the different form instances, and how transition to the next page will work. Where listing is needed, I will try to ensure proper navigation and data management.

**Decomposition to computable sections:**

|  |  |
| --- | --- |
| **Section** | **Justification (suitable for computation because…)** |
| Mapping Buttons to Open New Forms | This will allow my end user to navigate without errors by utilising different event handlers in the program to create and display new form instances. As a result I hope this will make sure user actions (button clicks) are mapped to their respective pages while also maintaining smooth transitions throughout the program as necessary per my analysis for my stakeholders listed above in requirements. |
| Implementing Navigation for Specific Pages | By assigning buttons to target pages (e.g., "Mock Test" opens InstructionsForm), the program can perform the way I want it to do for ease of user navigation in a predictable manner for testing and development throughout. This will also further help me eliminate errors, making sure the correct form opens in response to clicks on the buttons in the program by the end user. |
| Passing Data to Progress Page | Allows the Progress\_Page to show relevant user data - for example test scores. This will allows me to pass a dictionary from a global state in this program, making the content dynamic and real-time, as needed for my program relating to the analysis carried out which states that the user experience is tied directly to the ability to check their progress out. |
| Finalized Navigation Workflow | Allows for a logical workflow in relation to user interaction, navigation and data accessibility life cycle within my program. This will hopefully allow me to add smooth, button-based transitions while keeping the codebase fully functional and prevent any errors that hinder user experience. |

**Classes:**

**A screenshot of a computer screen

Description automatically generated**

Encapsulation

* The MainPage class is a combination of form-data (settings of a form, and event-handling routines).Some of the methods are Practice\_Click, Mock\_test\_Click and others that respond to the users’ actions without displaying the details.
* Methods like Practice\_Click, Mock\_test\_Click, and others handle user actions without exposing details.

Inheritance

* MainPage is also a class based on the Form class of the System.Windows.Forms namespace.
* Get things like the start position of a button, back color of a window form and the initialize component.s properties and methods like StartPosition, BackColor, and InitializeComponent.

Polymorphism

* Polymorphism in derived behavior or structure is well illustrated by functions such as Show () or Hide ().
* Based on active user actions, event handlers (e.g., Practice\_Click) exhibit runtime polymorphism.

Abstraction

* Predominantly, it conceals the logic of handling forms from the user interface.
* Most users engage with buttons without having a clue about the implementation that goes with the button.

Summary

* Encapsulation: Separately implemented behaviors concerning the encapsulated form in MainPage.
* Inheritance: Derived from Form.
* Polymorphism: Heavily relied on deep inheritance from Java and event handlers.
* Abstraction: Ease a form for the users.

***Step 1: Mapping Buttons to Open New Forms***

To navigate between different pages, I will create event handlers for every button. When the user clicks a button, the program will create an instance of the corresponding form, show it and optionally hide the current form in view.

**Pseudocode:**

WHEN button is clicked:

CREATE a new instance of the target form (e.g., PracticePage)

OUTPUT the target form

HIDE the current form (optional)

**Reason:**  
Mapping buttons to specific actions will allow my end users to navigate through the program seamlessly which is, as emphasised above and in my overall documentation, key to making this program useful – the ability for users to interact with the program and transition between the different forms with no errors or mishaps. By creating a new form instance on a button click, I can therefore make sure that the program responds immediately to the user’s input, improving interactivity and therefore the user experience. Hiding the current form (optional) will also prevent the clutter on the screen for my goal of user experience optimisation.

**Approach:**

|  |
| --- |
| **For all navigation buttons, I will define click event handlers.** |
| **Event handlers will be created for each form to instantiate and display it.** |
| **This overall should guarantee that the actions taken by users are mapped to the respective targets they were intended for within the program.** |

***Step 2: Implementing Navigation for Specific Pages***

In this next step, I need to add functionality to each button to allow users to open the appropriate page. These will consist of buttons such as "Mock Test," "Practice Page," and "Progress Page."

**Pseudocode for Mock Test Page:**

FUNCTION Mock\_test\_Click

CREATE new instance of InstructionsForm called nextForm

OUTPUT nextForm (Show it on the screen)

HIDE the current form

END FUNCTION

**Reason:**  
By clearly linking each button to its associated page (for example, the Mock Test button leads to the InstructionsForm), I can make sure that the program operates as the user expects it to. This is important for my intended user experience since the appropriate pages will load when users click on these buttons in the UI.

**Approach:**

|  |
| --- |
| **When Mock Test Button is pressed, I will try to make sure InstructionsForm page shows instructions before the user takes the test.** |
| **Completely check code for button references and their respective matches to prevent any bugs.** |

***Step 3: Passing Data to Progress Page***

To create the Progress\_Page, I will need to provide a dictionary that monitors the user’s test scores. This dictionary will then keep track of the tests that have been taken along with their respective scores for example Matthew scored 50% on Test 1. When I create the Progress\_Page, I will then be able to obtain the test scores from either a global state data class.  
  
**Pseudocode:**

FUNCTION Progress\_Click

RETURN testScores dictionary from GlobalData

CREATE new instance of Progress\_Page called nextForm, passing testScores as an argument

OUTPUT nextForm

CLOSE the current form

END FUNCTION

**Reason:**  
The *Progress\_Page* needs access to user progress data to display attempted tests and scores. By retrieving the testScores dictionary from a global state and passing it to the *Progress\_Page*, I ensure that the page reflects up-to-date and relevant information. This design also promotes separation of concerns, as data management is handled separately from UI navigation.  
  
The Progress\_Page should have access to the data of user progress such as attempted tests and scores. I call the global state for testScores dictionary and pass it to the Progress\_Page, therefore the page returns useful and current info. It also encourages separation of concerns, with data management being independent from UI navigation. I will set up a logic to get test scores from a global data store.

**Approach:**

|  |
| --- |
| **I will add logic which will aim to retrieve the test scores from a global data store.** |
| **When I navigate to that form, I will pass this dictionary to Progress\_Page constructor.** |
| **This should allow *Progress\_Page* to display scores dynamically and provide real-time feedback to the end user improving the user experience.** |

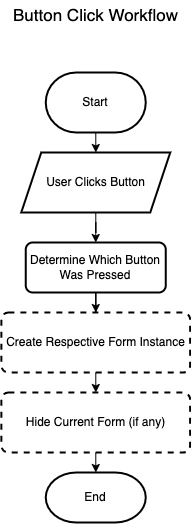
**Final Workflow Summary**

The finalized button navigation workflow will be implemented in three clear steps:

1. Since the state changes as described above, the program listens for individual button clicks.
2. Form Navigation Based on the button click, I will instantiate the relevant form and then show it
3. Data Management: For specific pages (e.g., *Progress\_Page*), I will pass necessary data to support dynamic content.

**Flowchart Representation**

The following flowchart summarizes the Button Click Workflow:

****

By carefully planning these steps, I can ensure smooth navigation between pages and efficient handling of user actions. Each form will serve its intended purpose, and any necessary data will be passed seamlessly. This design should ensures that the workflow is clean, functional, and easy to maintain.

**Development**

***Overview:***The notable feature in this C# created main page of traffic test application is that the main page of the traffic test application prototype is dedicated on organizing the multiple forms, including the Practice Page, Mock Test Page, Instructions Page, as well as the Progress Page. First, buttons were used for form manipulation where early releases focused on navigation problems, such as incorrect form addressing. To conform with this kind of format, the previously proposed logic had to have an Instructions Page before accessing the Mock Test. The subsequent advances involved a new Progress Page for monitoring the utilization performance. This was made possible through passing a dictionary to store test scores and a parameter to the instance of the Progress Page. Every cycle worked on changing the flow of home page and thus aimed at improving the navigation for the users.

***Development and Debugging (broken down into chronological iterations/updates):***

**Iteration 1:**  
A screenshot of a computer

Description automatically generated

**Problem:** How to Map buttons so that they open a new form?  
**Solution:** Applying the function with the Click Keyword with the name of the button.

private void Practice\_Click(object sender, EventArgs e)

{

PracticePage nextForm = new PracticePage();

nextForm.Show();

this.Hide();

}

Open new form:

PracticePage nextForm = new PracticePage();

nextForm.Show();

**Iteration 2:**

**Obstacle**: The Mock Test Page opening the Practice Page  
**Fix:** Reference Wasn’t Named Right Hence Causing the Practice Page to open Rather Than Mock test Page

private void Mock\_test\_Click(object sender, EventArgs e)

{

MockTest\_Page nextForm = new MockTest\_Page();

nextForm.Show();

this.Hide();

}

**Iteration 3:**

The Mock Test Page button now opens the Instruction form

private void Mock\_test\_Click(object sender, EventArgs e)

{

InstructionsForm nextForm = new InstructionsForm();

nextForm.Show();

this.Hide();

}

**Iteration 4:**

The New Progress\_Page constructor in my new logic which will require a Dictionary<int, int> argument that contains the scores for each test. This dictionary keeps track of which tests have been attempted and their respective scores. I need to pass this dictionary when creating an instance of Progress\_Page. In your MockTest\_Page, I already have a testScores dictionary that tracks the scores for each test. So, I can pass that dictionary when calling the constructor.

private void Progress\_Click(object sender, EventArgs e)

{

Progress\_Page nextForm = new Progress\_Page(GlobalData.TestScores);

nextForm.Show();

this.Hide();

}

**Iteration 5:**

Following the introduction of the Settings Page, the Hazard Perception Video Page, and the Update of Permanent Progress Storage, passing a dictionary parameter containing the test results is no longer required. In addition, I need to design new forms to map the Settings and Hazard Perception Video Page buttons.

private void Progress\_Click(object sender, EventArgs e)

{

Progress\_Page nextForm = new Progress\_Page();

nextForm.Show();

this.Hide();

}

private void button3\_Click(object sender, EventArgs e)

{

Settings\_Page nextForm = new Settings\_Page();

nextForm.Show();

this.Hide();

}

private void hazardvidtester\_Click(object sender, EventArgs e)

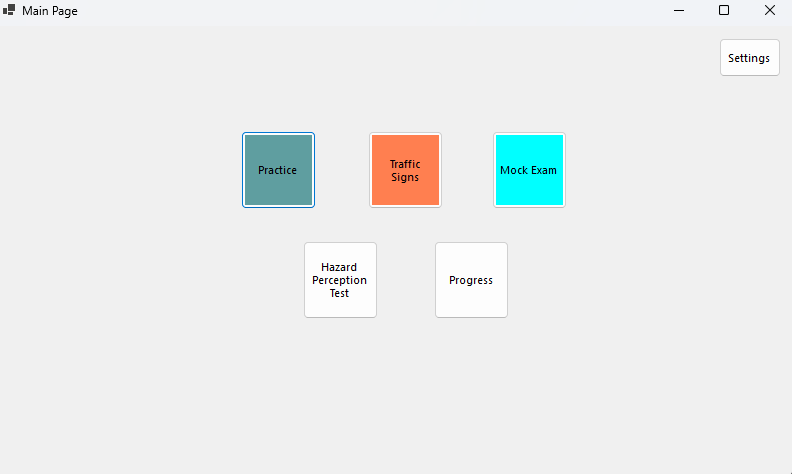
{

Hazard\_info nextForm = new Hazard\_info ();

nextForm.Show();

this.Hide();

}

******

***Review:***

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| Does the system allow users to mark topics as completed? |  |
| Does the system persist the checkbox state even when the user navigates away from the page? |  |
| When can a users click on a particular topic, what do they see as regards all the signs connected to it? |  |
| Does every traffic sign have a clear image and a description? |  |
| Does the progress of each topic update, at the instance when the user or the system, completes or sets the topic to the incomplete status? |  |
| Is it possible for the user to flow from one page to another (topics, progress, settings, signs)? |  |
| On the page showing the progress, must all the topics that were finished be indicated with perfect icons? |  |
| Is it easy to use the software to select topics, to look for signs whereas checkboxes must be manipulative? |  |
| Are users able to remove their findings on specific or on all specific topics in case they want a fresh start? |  |
| Are there any options that allow a user to see their performance over the course of the whole program on a single page? |  |
| Are checkboxes and progress indicators (checkmarks, completion percentages) clearly visible and easy to interact with? |  |
| Can users dynamically change the background color using a ColorDialog in the settings page? |  |
| Can users adjust font size using a slider, and toggle font styles (bold/italic) using checkboxes? |  |
| Are all user settings (background color, font size, font styles) applied globally and persisted between sessions? |  |
| Does the settings page provide real-time updates with previews for font and style changes? |  |
| Are images loaded efficiently without causing performance issues, even with a large number of topics or signs? |  |
| Can users interact with images, such as zooming in for better visibility of signs? |  |
| Does the system handle missing or corrupted image files gracefully by showing placeholders? |  |
| Does the layout remain clean, with proper spacing, alignment, and legibility for readability? |  |
| Is there clear error handling for issues like missing data, image errors, or faulty inputs? |  |
| Does the system load quickly without significant delays or lag when switching between pages or topics? |  |
| Does the system visually distinguish between completed and incomplete topics? |  |
| Is the reset button functional, deleting all relevant data files (scores, flagged questions) with success confirmation? |  |
| Does the system prevent crashes or unexpected behavior from faulty inputs or user actions? |  |
| Can flagged questions be viewed separately for easy access? |  |
| Are all forms and pages centered on the screen when opened? |  |
| Does the system handle high data volumes (e.g., many topics or signs) without performance issues? |  |
| Can additional content, such as multimedia (videos, animations), be added without major redesigns? |  |
| Is the system flexible enough to integrate new topics, signs, or complex features (e.g., quizzes, assessments)? |  |
| Does the overall user interface and experience align with personalization, usability, and consistency goals? |  |

**Processing Questions and User Answers**

To process questions and user answers, I will implement a loop that displays each question along with its answers, takes user input, and saves the selected answer into a dictionary.

**Pseudocode:**

DECLARE userAnswers AS DICTIONARY

FUNCTION ProcessQuestions(questionsList)

currentQuestionIndex = 0

WHILE currentQuestionIndex < questionsList.Count

OUTPUT questionsList[currentQuestionIndex.QuestionText]

OUTPUT questionsList[currentQuestionIndex.Answers]

OUTPUT “PLEASE GIVE AN INPUT”

INPUT userInput

IF userInput IS NULL OR EMPTY

CONTINUE // Go back to the same question

ELSE

userAnswers[currentQuestionIndex] = userInput

currentQuestionIndex += 1 // Move to next question

END IF

END WHILE

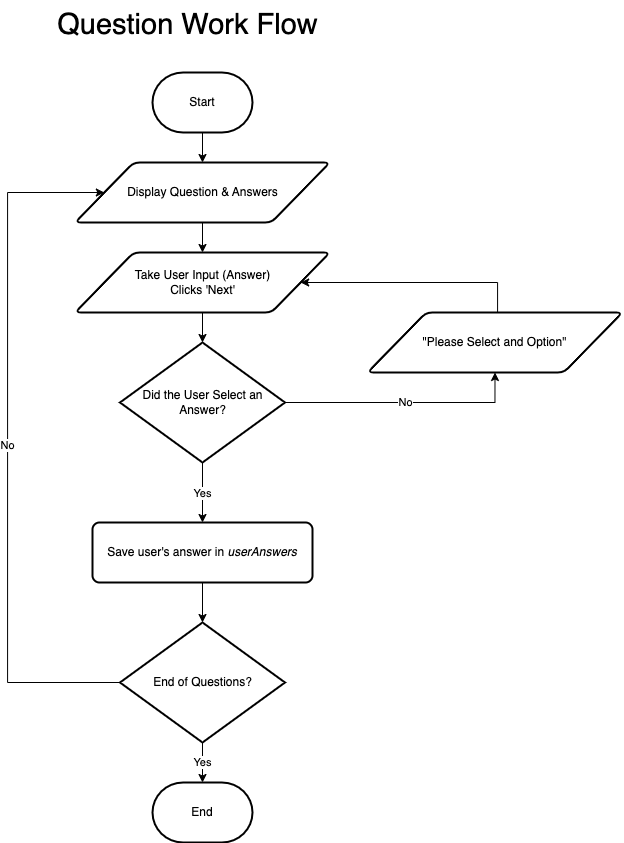
OUTPUT "All Questions Completed. Thank you!"

END FUNCTION

**Explanation:**

1. Dictionary Storage: Each answer is stored in a dictionary with question index as the key and user selected answer as the value.
2. Display and Input: Questions and answers are displayed, and user input is taken dynamically.
3. Validation : If the user does not select an answer, the program stays at the current question.
4. End Condition: The loop terminates when all questions are processed, and a completion message is shown. This is met when the user has gone through all the questions, and it displays a completion message.

**Flowchart Alignment:**

****

**Prototype 2 – Practice Page**

**Design**

**Overview:**In this section, I will design the workflow for the Practice Page, which encompasses navigation management, question management, and answer tracking from user inputs. The goal is to implement simple transitions between questions, calculate the user’s score, and display feedback based on user’s selections.

**Flowchart Representation:**

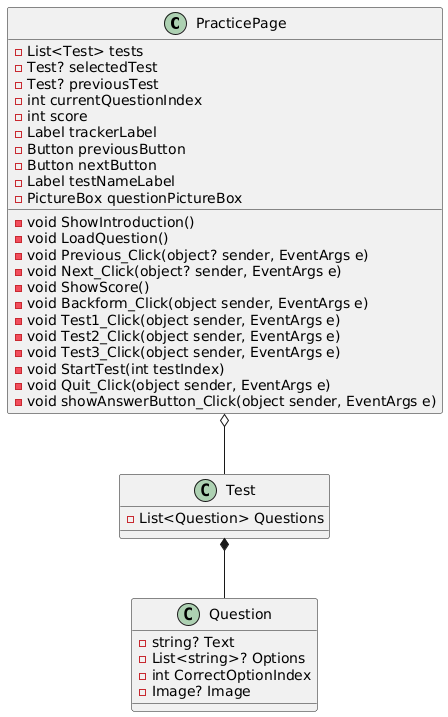
**A diagram of a flowchart

Description automatically generated**

**Decomposition into Computable Sections:**

|  |  |
| --- | --- |
| **Section** | **Justification (suitable for computation because…)** |
| Mapping Buttons to Navigate Between Questions | Enables users to navigate between questions with Next and Previous buttons. This will make sure the end user can navigate through the questions whilst keeping track of your progress. |
| Tracking and Displaying Current Question | Keeps track of the current question index to ensure users always know which question they are on. This ensures a clean user interface and proper question management for an effective UI in my program. |
| Updating the Score Dynamically | I need to try to dynamically calculate and updates the score based on the user's responses. This will provide real-time feedback and improves the user experience. |
| Display Correct Answers After Completion | The program needs to makes sure that after the user completes the test, they can see the right answers. This provides useful feedback for the user and aids retention as highlighted in my analysis. |

**Classes:**

****

Encapsulation

* The Test and Question classes contain test related information such as Questions and Options.
* Allow accessing only with properties using accesses, for instance, public List<Question> Questions.

Abstraction

* These methods such as LoadQuestion, ShowScore, ShowIntroduction masks the UI and the back-end functionality.

Inheritance

* PracticePage class extends Form and includes custom behavior and construction of methods linked with events.

Polymorphism

* By their behavior, event handlers (Next\_Click, Previous\_Click) illustrate the polymorphism example.

Summary

* Stresses the methodologies of encapsulation and abstraction.
* Supertype of the rest URLs; polymorphism for interaction with UI.

***Step 1: Mapping Buttons to Navigate Between Questions***

I am going to implement and allow users to use Next and Previous buttons to navigate through the questions. These buttons also update the currentQuestionIndex when clicked and route to the appropriate question.

**Pseudocode:**

WHEN NextButton\_Click:

IF currentQuestionIndex < selectedTest.Questions.Count - 1:

currentQuestionIndex =+1

LoadCurrentQuestion()

WHEN PreviousButton\_Click:

IF currentQuestionIndex > 0:

currentQuestionIndex =-1

LoadCurrentQuestion()

**Reason**: By mapping buttons to navigate actions, users can progress through the test in order and maintain flow. Also, the buttons will make sure that you iterate through the questions instead of going in an unwanted manner (for example, going from first to third question while skipping a second one which would be undesirable for the end user).

**Approach:**

* Attach click event handlers for Next and Previous buttons.
* Each handler will update the question index and refresh the displayed question accordingly.

***Step 2: Tracking and Displaying Current Question***

The program needs to know the index of the current question for it to show the correct question at any time. I am going to show the number of current question (e.g. Question 2 of 5).

**Pseudocode:**

FUNCTION UpdateQuestionDisplay

trackerLabel = Question [currentQuestionIndex + 1]   
END FUNCTION

**Reason:** Showing the current number of question allows users to see where they are in the test, which helps in navigation and in turn users know how far along we are.

**Approach:**

|  |
| --- |
| **Change the question index every time the user clicks on next or previous in test.** |
| **Ensure the trackerLabel always reflects the current position in the test.** |

***Step 3: Updating the Score Dynamically***

Here’s how the user will be scored as they answer each question. The user is given instant feedback on whether their answer was right or wrong.

**Pseudocode:**

FUNCTION UpdateScore(selectedOptionIndex)

IF selectedOptionIndex == CorrectOptionIndex(currentQuestion)

Score = +1  
END FUNCTION

**Reason:** As a result of this, the user will receive live feedback on their score; this information allows them to see how many questions they answered correctly thus providing a sense of real-time performance metrics.

**Approach:**

|  |
| --- |
| **Check the selected answer against the correct answer and update the score.** |
| **Display the score update in a dynamic way, either after each question or at the end of the test.** |

***Step 4: Displaying Correct Answers After Completion***

Once the user completes the test, the program should display the correct answers for each one of the questions. This feedback will enable users to understand and learn from their mistakes.

**Pseudocode:**

FUNCTION ShowCorrectAnswers

FOR LOOP question IN Questions(selectedTest)

IF selectedOptionIndex != CorrectOptionIndex(question)

Highlight incorrect answers

ELSE

Highlight correct answer  
END FUNCTION

**Reason**: Displaying the correct answers after the test is completed will allow users to learn from their mistakes and reinforce the correct answers for retention and understanding of the material as required in reference to my aims for the app.

**Approach:**

|  |
| --- |
| Once the test is over, the aim is to loop through each question and check the answer against the correct one in the database. |
| Highlight answers appropriately (e.g., correct answers in green, incorrect ones in red). |

Final Workflow Summary  
The finalized workflow for the Practice Page will consist of the following:

1. Navigation: The user navigates through the questions using the Next and Previous buttons, with the current question being displayed dynamically.
2. Score Calculation: The program updates the score as the user answers each question, providing real-time feedback.
3. Answer Display: Once the test is completed, the correct answers will be displayed to provide feedback to the user.

By carefully planning these steps, I will ensure that the Practice Page functions efficiently, providing a smooth user experience. The navigation, score tracking, and answer display mechanisms will help improve the test-taking experience and make the page dynamic and interactive as needed for this program to be successful in it’s approach of teaching users.

**Development**

***Overview:***

The Practice Page is an interactive quiz-based learning platform designed to help users practice and test their knowledge on various topics. Users can select from multiple predefined tests, each containing a unique set of questions. The page presents one question at a time, offering multiple answer options for users to choose from. After submitting an answer, users receive immediate feedback on its correctness or view their results at the end of the test. The system tracks user progress, showing the number of questions answered out of the total, and provides a final score summary upon completion. The interface is simple, intuitive, and user-friendly, allowing for easy navigation between questions. By offering a self-paced learning experience, the Practice Page encourages users to improve their understanding of key concepts while tracking their performance. This approach makes it an effective tool for education, exam preparation, and skill development.

***Development and Debugging (broken down into chronological iterations/updates):***

**Iteration 1:**

  
**Problem**: How to Map the start button to start the quiz?  
**Solution**: Use A Function class to call another class which clears the form from everything and loads in questions

Controls.Clear();

**Problem**: Tracker to track the question number you are on and iterating to the next question?  
**Solution**: Introduce a Tracker Label ***private Label trackerLabel;***   
The total number of questions in the test is determined by the `selectedTest.Questions.Count` property, which retrieves the size of the `Questions` collection in the `selectedTest` object. This count is used to ensure the test ends when the current question index (`currentQuestionIndex`) reaches the total number of questions

if (selectedTest == null || currentQuestionIndex >= selectedTest.Questions.Count)

**To update the tracker label:** that displays the current question number Each `Question` object in the `Questions` collection contains the question text and its possible options.

trackerLabel.Text = $"Question {currentQuestionIndex + 1} of {selectedTest.Questions.Count}";

**Iterate through the options**: A for loop is used for the collection selectedTest.Questions contains Question objects, where each question has its text (currentQuestion.Text) and a list of options (currentQuestion.Options).

for (int i = 0; i < currentQuestion.Options.Count; i++)

{

RadioButton optionButton = new RadioButton

{

Text = currentQuestion.Options[i],

AutoSize = true,

Tag = i // Store the option index

};

optionButton.Location = new Point((ClientSize.Width - optionButton.Width) / 2, yPosition);

Controls.Add(optionButton);

yPosition += 30;

}

**Iteration 2:**

**Problem**: I need buttons to navigate between questions but in a smoother way, ensuring I don't navigate to a non-existent question.  
**Solution**: I use the currentQuestionIndex variable to keep track of the current question I'm viewing. This variable updates when I navigate between questions, ensuring the correct question from selectedTest.Questions is loaded.

nextButton for Moving Forward by incrementing currentQuestionIndex which calls LoadQuestion() to load the next question

private void NextButton\_Click(object sender, EventArgs e)

{

currentQuestionIndex++;

LoadQuestion();

}

ensures the subsequent question is displayed as long as currentQuestionIndex is less than selectedTest.Questions.Count.

previousButton for Moving Backward by decrementing currentQuestionIndex which calls LoadQuestion() to load the previous question

private void PreviousButton\_Click(object sender, EventArgs e)

{

if (currentQuestionIndex > 0)

{

currentQuestionIndex--;

LoadQuestion();

}

}

Navigation backward is only allowed when currentQuestionIndex is greater than 0, ensuring the user doesn't navigate to a non-existent question.

**Problem**: I need to see how many questions I've gotten right and which option is correct. How will the scoring mechanism work?  
**Solution**: I will compare the selected option with the correct answer for each question. Each Question object has a property called CorrectOptionIndex, which stores the index of the correct option in the Options list.

private void PreviousButton\_Click(object sender, EventArgs e)

{

if (currentQuestionIndex > 0)

{

currentQuestionIndex--;

LoadQuestion();

}

}

When the user clicks the Next button, iterations through the Controls collection are done to check which RadioButton is selected. The Tag property of the selected RadioButton holds the index of the selected option by using foreach:

foreach (Control control in Controls)

{

if (control is RadioButton radioButton && radioButton.Checked)

{

int selectedOption = (int)radioButton.Tag;

...

}

}

The Scoring Mechanism works by comparing the user's selected option (selectedOption) with the correct option (CorrectOptionIndex) for the current question. If they match, the score is incremented:

foreach (Control control in Controls)

{

if (control is RadioButton radioButton && radioButton.Checked)

{

int selectedOption = (int)radioButton.Tag;

...

}

}

At the end of the quiz, the ShowScore() method is called. This method clears the form and displays the user's total score out of the total number of questions:

Label scoreLabel = new Label

{

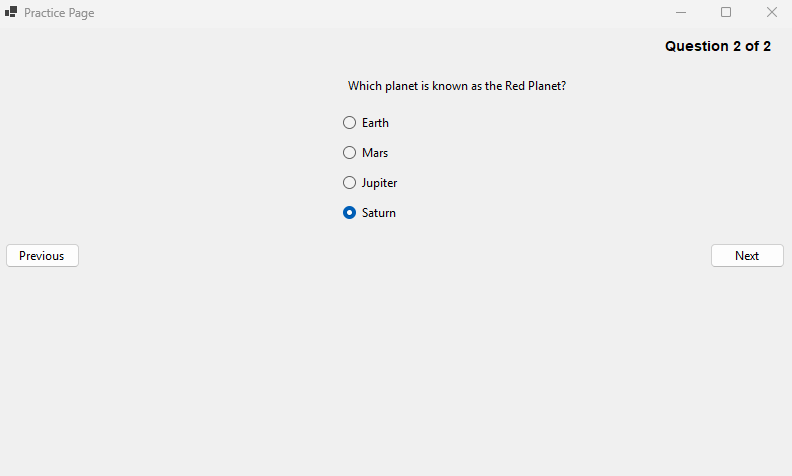
Text = $"Quiz Completed!\nYour score: {score} out of {questions.Count}",

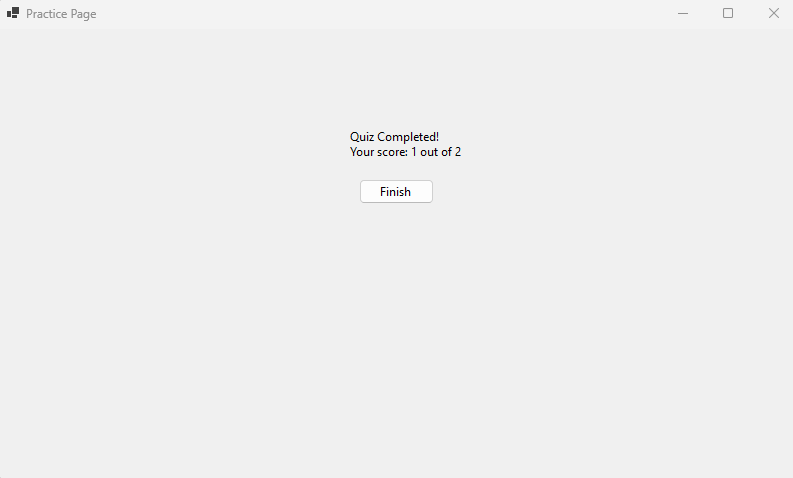
AutoSize = true

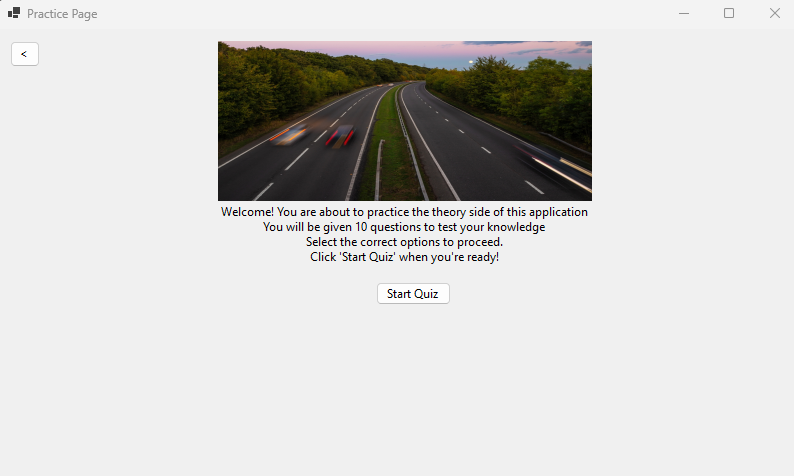
};

***A screenshot of a computer

Description automatically generated***

******

****

**Iteration 3:**   
  
**Problem:** How to Have Multiple Tests with different Questions? **Solution:** I am implementing multiple tests with different questions by organizing the questions into a List<Test>, where each Test object contains its own set of questions.

public *class* Test

{

public List<Question> Questions { *get*; *set*; } = new List<Question>();

}

public *class* Question

{

public string? Text { *get*; *set*; }

public List<string>? Options { *get*; *set*; }

public int CorrectOptionIndex { *get*; *set*; }

}

This structure allows each Test to encapsulate its own unique set of questions.

**Problem:** How to store these tests?  
**Solution:** I am creating multiple tests by using a List<Test> to store multiple tests, with each Test initialized with a unique set of questions.

tests = new List<Test>

{

new Test

{

Questions = new List<Question>

{

new Question

{

Text = "What is the capital of France?",

Options = new List<string> { "Berlin", "Madrid", "Paris", "Rome" },

CorrectOptionIndex = 2

},

new Question

{

Text = "Which planet is known as the Red Planet?",

Options = new List<string> { "Earth", "Mars", "Jupiter", "Saturn" },

CorrectOptionIndex = 1

},

}

},

new Test

{

Questions = new List<Question>

{

new Question

{

Text = "What is the largest ocean on Earth?",

Options = new List<string> { "Atlantic", "Indian", "Arctic", "Pacific" },

CorrectOptionIndex = 3

},

new Question

{

Text = "What is the square root of 64?",

Options = new List<string> { "6", "7", "8", "9" },

CorrectOptionIndex = 2

},

}

},

// More tests can be added here

};

**Problem:** How to make the tests Random and Dynamic at the same time? Why To Choose Tests at Random?   
**Solution:** I am making the test selection random by using the Random class in the StartQuizButton\_Click method.

private void StartQuizButton\_Click(object? sender, EventArgs e)

{

Random random = new Random(); // Create a new instance of the Random class

selectedTest = tests[random.Next(tests.Count)]; // Randomly select a test

currentQuestionIndex = 0;

score = 0;

LoadQuestion(); // Start loading the first question

}

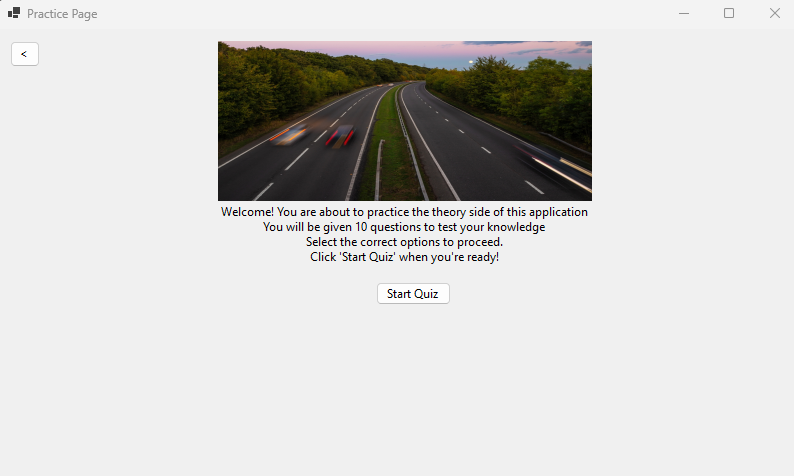
I use the Random class to generate pseudo-random numbers. The selection is done by random.Next(tests.Count), which generates a random integer between 0 (inclusive) and tests.Count (exclusive). This ensures that the index is valid within the bounds of the tests list, so a test is picked from the list.

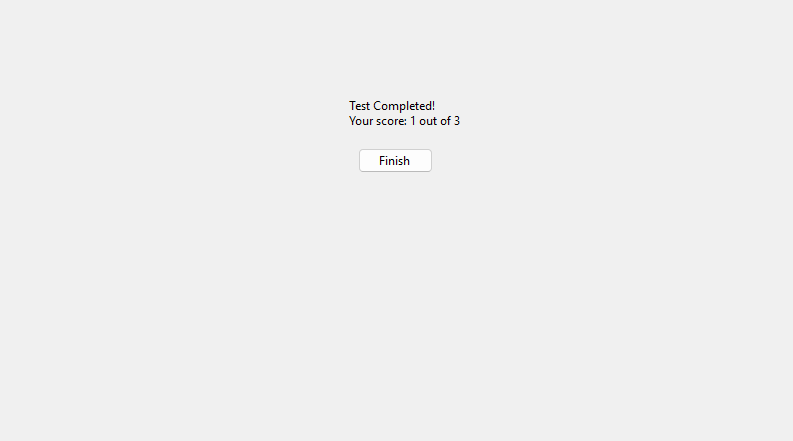
selectedTest = tests[random.Next(tests.Count)];

Dynamic Selection is ensured whenever the StartQuizButton\_Click method is executed (when the user starts the quiz), a new random test is selected.  
This ensures variety in the quiz experience every time it starts.

The Use of Choosing Random test is to ensure that the quiz feels fresh for the user by not always starting with the same test and to evenly distribute the selection across all available tests

**Obstacle**: When the test is End and the Quiz is started by the User it shows me the final result of the test rather than starting a new one



******

**Fix**: The issue arises because the currentQuestionIndex and score variables are not reset after completing the first test. When I click the "Start Test" button again, the previous values persist, causing the program to skip to the score display.

private void StartQuizButton\_Click(object? sender, EventArgs e)

{

// Reset test-specific variables

currentQuestionIndex = 0;

score = 0;

// Randomly select one test

Random random = new Random();

selectedTest = tests[random.Next(tests.Count)];

InitializeTimer();

LoadQuestion();

}

**Obstacle:** Facing the same test which the user has just attempted  
**Fix:** I will change the random selection logic to ensure I don't repeat the same test as the previous one. The previousTest variable is used to store the last selected test.

private void StartQuizButton\_Click(object? sender, EventArgs e)

{

// Randomly select a test that is not the same as the previous one

Random random = new Random();

Test? newTest;

do

{

newTest = tests[random.Next(tests.Count)];

} while (newTest == previousTest); // Repeat until a different test is selected

previousTest = newTest; // Update the previous test

//...(Below Logic Remains the same)

}

**Problem:** showing the user What test They are attempting at the moment.  
**Solution:** To display the name or index of the current test, adding a label that updates whenever a new test is selected

Modifying the StartQuizButton\_Click method to update the label when a test is selected

// Update the test name label

int testIndex = tests.IndexOf(selectedTest) + 1; // Test index starts from 1 testNameLabel.Text = $"Test {testIndex}";

Ensuring the Label is Visible During the Quiz by adding the testNameLabel to the form in the LoadQuestion method to ensure it remains visible when navigating through questions:

// Display the current question

Question currentQuestion = selectedTest.Questions[currentQuestionIndex]; Label questionLabel = new Label { Text = currentQuestion.Text, AutoSize = true }; questionLabel.Location = new Point((ClientSize.Width - questionLabel.Width) / 2, 50); Controls.Add(questionLabel);

**Iteration 4:**

I am changing the logic of random tests to user choice, as this is a practice mode and that logic is more suitable for the mock test page. I am also replacing the StartQuiz class with StartTest.

private void StartTest(int testIndex)

{

selectedTest = tests[testIndex];

currentQuestionIndex = 0;

score = 0;

// Update the test name label

testNameLabel.Text = $"Test {testIndex + 1}";

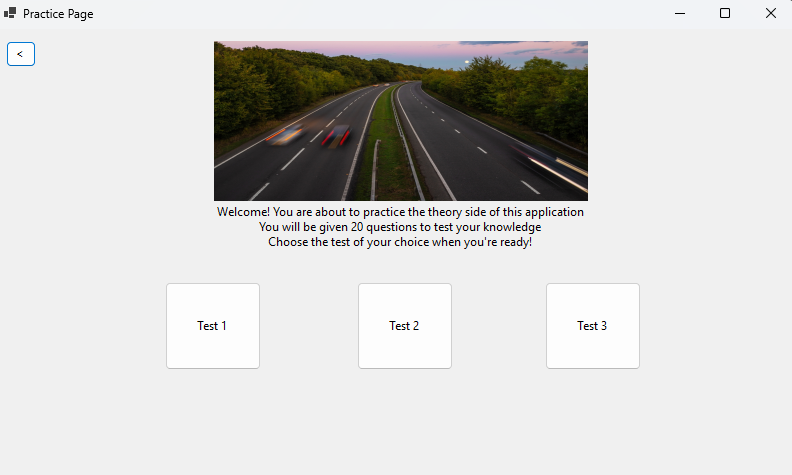
LoadQuestion();

}

// Update the test name label

testNameLabel.Text = $"Test {testIndex + 1}";

LoadQuestion();

******

***Problem:*** What if I want to quit the practice? A Quit button will help me quit at any given time.

***Solution:*** I will add a Quit button, which, when clicked, takes me back to the initial screen.

private void Quit\_Click(object sender, EventArgs e)

{

// Reset test-related variables

selectedTest = null;

previousTest = null;

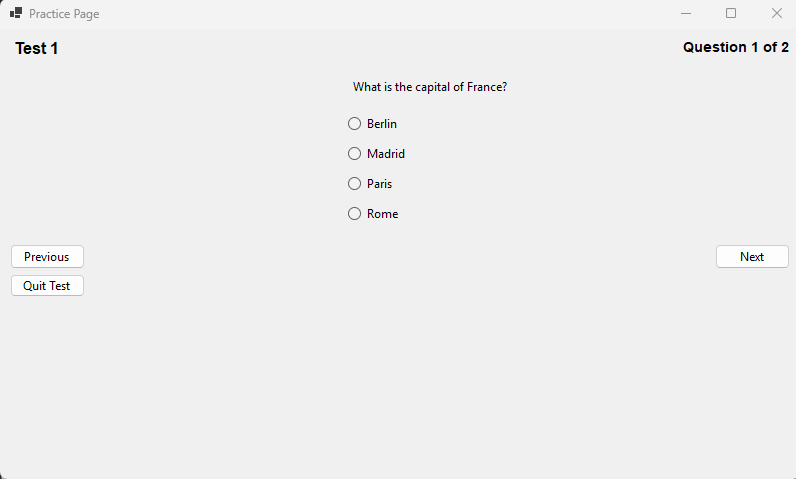
currentQuestionIndex = 0;

score = 0;

// Return to the introduction screen

ShowIntroduction();

}

****Problem:** What if the user wants to see the answer to the question right there?  
**Solution:** I will add a "Show Answer" button, which, when clicked, highlights the correct answer in green.

private void showAnswerButton\_Click(object sender, EventArgs e)

{

if (selectedTest == null || currentQuestionIndex >= selectedTest.Questions.Count)

{

MessageBox.Show("No question loaded to show the answer.");

return;

}

// Find the correct answer for the current question

Question currentQuestion = selectedTest.Questions[currentQuestionIndex];

int correctOptionIndex = currentQuestion.CorrectOptionIndex;

// Iterate through controls to find the RadioButton with the correct answer

foreach (Control control in Controls)

{

if (control is RadioButton radioButton && (int)radioButton.Tag == correctOptionIndex)

{

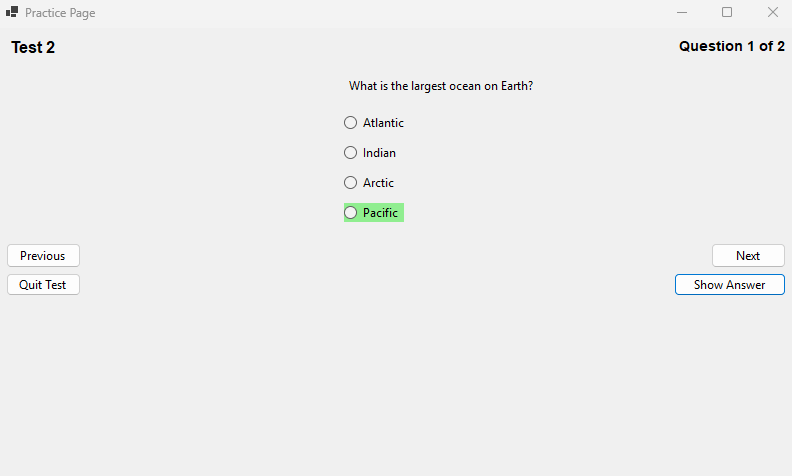
radioButton.BackColor = Color.LightGreen; // Highlight the correct answer

break;

}

}

}

******

**Iteration 5:**

Problem: How to add pictorial Questions?  
Solution: I will introduce a picture box to display the picture and set a proper location so that it doesn’t overlap with any button.

// Initialize PictureBox for question images

questionPictureBox = new PictureBox

{

SizeMode = PictureBoxSizeMode.Zoom,

Size = new Size(300, 200), // Default size

Visible = false, // Initially hidden; only shown if there's an image

Location = new Point((ClientSize.Width - 300) / 2, 200) // Centered

};

Edit load questions function to display and show the picture if it exists

// Display image if available

if (currentQuestion.Image != null)

{

questionPictureBox.Image = currentQuestion.Image;

questionPictureBox.Visible = true;

}

else

{

questionPictureBox.Visible = false;

}

Make the image property global and give proper path of the image as well

public Image? Image { get; set; } // New property for image

Image =Image.FromFile("E:\\CLIENT\\WindowsFormsApp1\\STOP\_SIGN\_PIC.jpg") // Add a valid image file path

******

***Review:***

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| Does the system allow users to mark topics as completed? |  |
| Does the system persist the checkbox state even when the user navigates away from the page? |  |
| When can a users click on a particular topic, what do they see as regards all the signs connected to it? |  |
| Does every traffic sign have a clear image and a description? |  |
| Does the progress of each topic update, at the instance when the user or the system, completes or sets the topic to the incomplete status? |  |
| Is it possible for the user to flow from one page to another (topics, progress, settings, signs)? |  |
| On the page showing the progress, must all the topics that were finished be indicated with perfect icons? |  |
| Is it easy to use the software to select topics, to look for signs whereas checkboxes must be manipulative? |  |
| Are users able to remove their findings on specific or on all specific topics in case they want a fresh start? |  |
| Are there any options that allow a user to see their performance over the course of the whole program on a single page? |  |
| Are checkboxes and progress indicators (checkmarks, completion percentages) clearly visible and easy to interact with? |  |
| Can users dynamically change the background color using a ColorDialog in the settings page? |  |
| Can users adjust font size using a slider, and toggle font styles (bold/italic) using checkboxes? |  |
| Are all user settings (background color, font size, font styles) applied globally and persisted between sessions? |  |
| Does the settings page provide real-time updates with previews for font and style changes? |  |
| Are images loaded efficiently without causing performance issues, even with a large number of topics or signs? |  |
| Can users interact with images, such as zooming in for better visibility of signs? |  |
| Does the system handle missing or corrupted image files gracefully by showing placeholders? |  |
| Does the layout remain clean, with proper spacing, alignment, and legibility for readability? |  |
| Is there clear error handling for issues like missing data, image errors, or faulty inputs? |  |
| Does the system load quickly without significant delays or lag when switching between pages or topics? |  |
| Does the system visually distinguish between completed and incomplete topics? |  |
| Is the reset button functional, deleting all relevant data files (scores, flagged questions) with success confirmation? |  |
| Does the system prevent crashes or unexpected behavior from faulty inputs or user actions? |  |
| Can flagged questions be viewed separately for easy access? |  |
| Are all forms and pages centered on the screen when opened? |  |
| Does the system handle high data volumes (e.g., many topics or signs) without performance issues? |  |
| Can additional content, such as multimedia (videos, animations), be added without major redesigns? |  |
| Is the system flexible enough to integrate new topics, signs, or complex features (e.g., quizzes, assessments)? |  |
| Does the overall user interface and experience align with personalization, usability, and consistency goals? |  |

**Prototype 3 - Mock Test Page**

**Design**

**Overview:**   
The functionality of Mock Test Page will be similar to that of the practice page as laid out above but it will have the added countdown timer feature. The timer is necessary in order to emulate the examination that users are preparing for. The users will be able to navigate through each question where the user can flag important questions, and review their performance at the end in the form of a ‘Mock Examination’ as highlighted in the analysis section of my project. This is a key feature for my stakeholders.

**Flowchart Representation:**

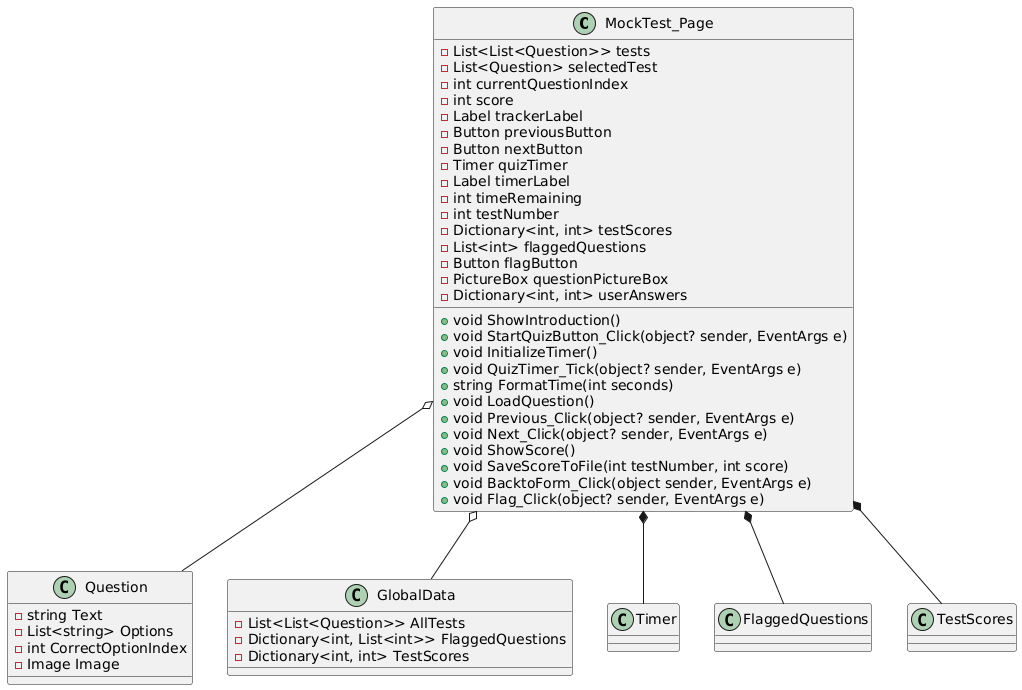
**A diagram of a process

Description automatically generated**

**Decomposition into Computable Sections:**

|  |  |
| --- | --- |
| **Section** | **Justification (suitable for computation because…)** |
| Implementing Timer | The countdown timer keeps track of the test duration and automatically ends the test once the allotted time has run out mirroring the conditions of a real exam. |
| Updating Timer | The timer needs to be updated every second to display the time the user has remaining and end the test when this time hits zero regardless of questions left. Reflecting the examination once again. |
| Flagging Questions | Allows users to flag questions they want to review later and navigate through these flagged questions. |
| Displaying User’s Choices and Results | After completing the test, the app needs to display the user’s selections and correct answers. |
| Displaying Review of Incorrect Answers | Providing feedback to the user’s by highlighting all of the incorrect answers in red and the correct answers in green. |

**States:**

****

Encapsulation

* It is necessary to point out that data is enclosed in such fields as tests, selectedTest, currentQuestionIndex, and flaggedQuestions.
* Many functions including ShowIntroduction, LoadQuestion, SaveScoreToFile etc. are Analyzed functional methods.

Abstraction

* LoadQuestion and SaveScoreToFile are example of anti-slices as they encapsulate logic of presenting questions and handling scores.

Inheritance

* MockTest\_Page derives from Form, it gets all the properties and methods of the Form class including Controls, Size, Hide.

Polymorphism

* Form level event handlers Previous\_Click, Next\_Click also show polymorphism through method overriding paring.
* Handling of answers in RadioButton can be managed dynamically by use of Tag property.

Summary

* Being a bad program all four of the pillars of OOP are implemented.
* They both help in improving on the degree of modularity.
* Takes over the Windows Forms abilities and enforces polymorphism on the fly.

### ***Step 1: Adding a Timer Feature***

**Feature:**  
Implement a countdown timer to display the remaining time for the Mock Test.

**Pseudocode:**

WHEN InitializeTimer:

Set timeRemaining = 30 \* 60 // 30 minutes

Start quizTimer every 1 second

Display timeRemaining on timerLabel

WHEN quizTimer\_Tick:

IF timeRemaining > 0:

timeRemaining = -1

Update timerLabel with remaining time

ELSE:

Stop quizTimer

Show final score

ENDWHILE

**Reason:**  
The test needs to utilise a countdown timer to keep the user aware of the time remaining in order to complete the quiz within the time limit.

**Approach:**

|  |
| --- |
| **Upon page loading, create and start the timer.** |
| **Update the time every second and display it in a label.** |
| **Stop the timer when the time runs out and show the final score.** |

### ***Step 2: Stopping the Quiz When Time Runs Out***

**Feature:**  
Finish the test automatically when the timer reaches 0 and show the score.

**Pseudocode:**

WHEN quizTimer\_Tick:

IF timeRemaining == 0:

Stop quizTimer

Call ShowScore() to display the final score

ENDWHILE

**Reason:**  
This will ensure the test ends immediately when the timer expires, preventing any further answers or changes to any answers given by the user.

**Approach:**

* Monitor the timer during each increment and check for when time runs out.
* Call the ShowScore method to display the results once time is up.

### ***Step 3: Flagging Questions for Review***

**Feature:**  
Implement the ability for the user to flag certain questions that they can review later as needed for a Mock Test.

**Pseudocode:**

WHEN FlagButton\_Click

IF currentQuestionIndex is not in flaggedQuestions:

Add currentQuestionIndex to flaggedQuestions

ELSE

Remove currentQuestionIndex from flaggedQuestions

**Reason:**  
Flagging questions will allow the user to mark the questions they want to revisit later, improving their ability to understand and revisit difficult questions.

**Approach:**

|  |
| --- |
| **Initialise a list flaggedQuestions to store flagged question indices.** |
| **Update the flag status whenever the "Flag" button is clicked.** |

### ***Step 6: Skipping Flagged Questions***

**Feature:**  
Allow the user to skip flagged questions and answer the next one - without selecting an answer just like the real examination.

**Pseudocode:**

WHEN NextButton\_Click

IF currentQuestionIndex is flagged

currentQuestionIndex = +1

LOAD Next Question

ELSE IF selectedAnswer is valid

Save the answer and proceed

**Reason:**  
This feature will make it so that users aren’t forced to answer flagged questions immediately, giving them the ability to skip and return later.

**Approach:**

|  |
| --- |
| **Check if the current question is flagged before navigating further.** |
| **Skip flagged questions if the user decides to move forward without answering.** |

### ***Step 7: Tracking User Answers***

**Feature:**  
Track and store answers given by the user for review after the test has been completed.

**Pseudocode:**

WHEN NextButton\_Click

Save selectedanswer in userAnswers[currentQuestionIndex]

**Reason:**  
Tracking answers allows users to review their responses at the end of the test, highlighting correct and incorrect choices for performance review.

**Approach:**

|  |
| --- |
| **Store each selected answer in a dictionary (userAnswers).** |
| **Update this dictionary each time the user answers a question.** |

### ***Step 8: Displaying Correct Answers After Completion***

**Feature:**  
At the end of the quiz, show the correct answers alongside the user's inputted selections during the mock examination.

**Pseudocode:**

FUNCTION ShowCorrectAnswers:

FOR LOOP question in selectedTest.Questions:

IF userAnswers[questionIndex] != question.CorrectOptionIndex:

Highlight the user's answer as incorrect

Highlight the correct answer as correct

ELSE:

Highlight the user's answer as correct

**Reason:**  
Displaying the correct answers for each question that the user did not answer correctly after the quiz helps users learn from their mistakes.

**Approach:**

|  |
| --- |
| **Iterate over the answers and highlight them (are they incorrect or correct?).** |
| **Display feedback after the quiz is completed to show the correct answers to the questions.** |

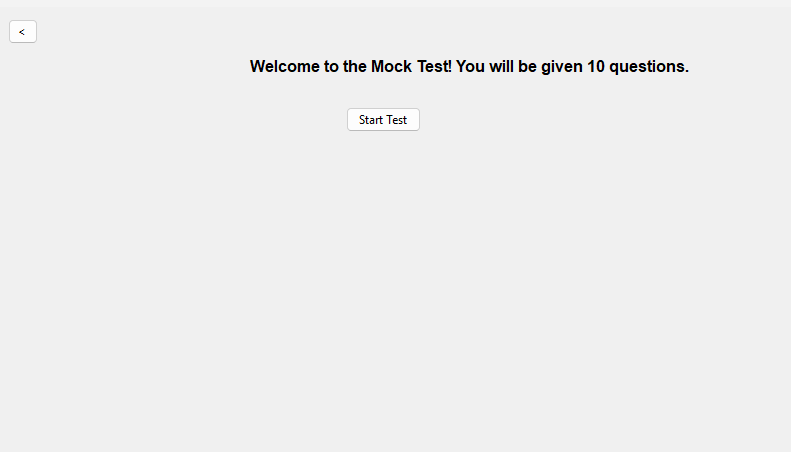
**Development**

***Overview:***

The Mock Test Page simulates a time-limited quiz, with a 57-minute countdown timer. Users can flag difficult questions for later review, and navigate through the test even without answering flagged questions. The test tracks answers, highlighting incorrect ones in red and correct ones in green in the results. A checkbox must be selected to confirm the user has read the instructions before starting the test. The page shows the time remaining and progress, and detailed feedback is provided on incorrect answers after the test ends.

***Development and Debugging (broken down into chronological iterations/updates):***

**Iteration 1:**



Everything is the same as the **Practice Page** except there is a timer now.  
**Problem**: How to Implement a CountDown?

**Solution:** I will implement the countdown timer by initializing a Timer in the StartQuizButton\_Click method.

private void InitializeTimer()

{

if (quizTimer == null)

{

quizTimer = new Timer();

quizTimer.Interval = 1000; // 1-second intervals

quizTimer.Tick += QuizTimer\_Tick; // Event handler when timer ticks

}

timeRemaining = 30; // Set the initial countdown time to 30 seconds

quizTimer.Start(); // Start the timer

timerLabel = new Label

{

Text = $"Time Left: {timeRemaining} seconds",

AutoSize = true,

Font = new Font("Arial", 10, FontStyle.Bold),

ForeColor = Color.Red,

Location = new Point(10, 10)

};

Controls.Add(timerLabel); // Add the timer label to the form

}

**Iteration 2:**

**Problem:** How will the timer would be Updated? This timer should Stop the Test when the timer runs out:

**Solution:** I will create the QuizTimer\_Tick method, which is triggered every second. It will update the timeRemaining value and display the remaining time on the timerLabel. If the timer reaches zero, the test will stop, and the score will be shown.

private void QuizTimer\_Tick(object? sender, EventArgs e)

{

timeRemaining--; // Decrease the time remaining by 1 second

if (timeRemaining <= 0)

{

quizTimer.Stop(); // Stop the timer when it reaches 0

ShowScore(); // Show the score when the time runs out

}

else

{

timerLabel.Text = $"Time Left: {timeRemaining} seconds"; // Update the label

}

}

The timer starts when the test begins with a countdown of 30 seconds. The quizTimer\_Tick event handler decreases the remaining time every second and updates the UI. Once the countdown reaches zero, the test is stopped by calling quizTimer.Stop(), and the ShowScore method is invoked to display the final score.

**Obstacle:** Severity Code Description Project File Line Suppression State Details Error (active) CS0104 'Timer' is an ambiguous reference between 'System.Windows.Forms.Timer' and 'System.Threading.Timer' WinFormsApp1 **Fix:** The error occurs because both System.Windows.Forms.Timer and System.Threading.Timer are referenced, and the compiler doesn't know which one to use. To resolve this ambiguity, I need to explicitly qualify the Timer class with its namespace. For example, if I'm working with a Windows Forms application, I will use:

System.Windows.Forms.Timer quizTimer = new System.Windows.Forms.Timer();

This will ensure that the correct Timer class is used in the context of my Windows Forms application.

private System.Windows.Forms.Timer quizTimer; // Timer object

In the class of IntializeTimer we need to change the qualification

private void InitializeTimer()

{

if (quizTimer == null)

{

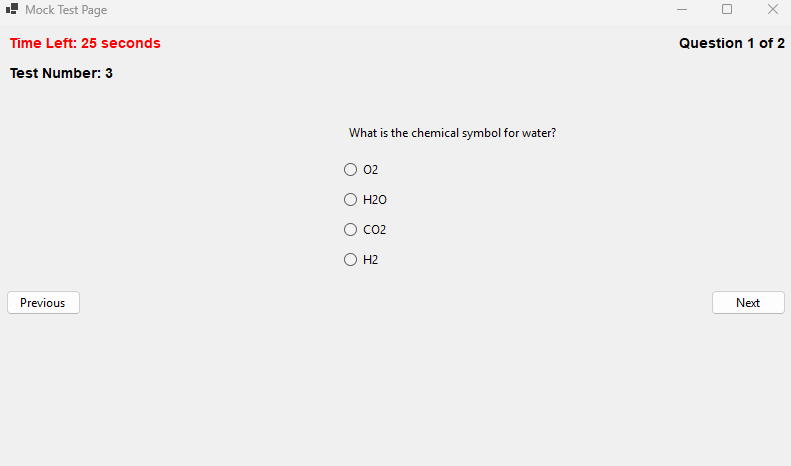
quizTimer = new System.Windows.Forms.Timer(); // Explicit namespace usage

quizTimer.Interval = 1000; // 1 second intervals

quizTimer.Tick += QuizTimer\_Tick;

}

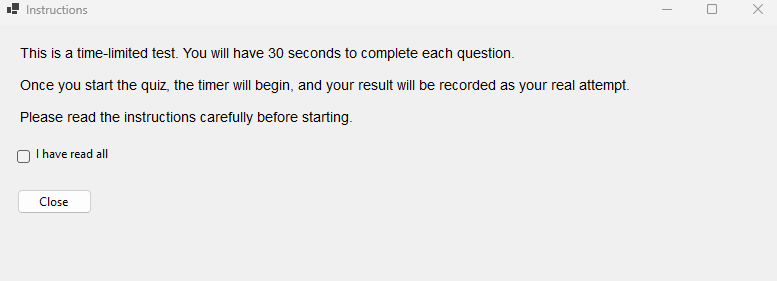
}



**Iteration 3:**

**Problem:** I need to make sure that I know this is a time-limited test and that I can't start until I have read the instructions carefully. To achieve this, I will clearly display a message stating that the test is time-limited, and ensure the "Start Test" button is only enabled once I check the "I have read all of the instructions" checkbox.

**Solution**: A new form is opened when the "Mock Test" button is clicked on the main page, containing the instructions that I must read before I can start the quiz. The Mock Test page form won't open until I have checked the "I have read all of the instructions" checkbox.

  
**Iteration 4:**  
Increasing the Timer from 30 sec to 57 Minutes by changing the Initialize Timer Code

private void InitializeTimer()

{

if (quizTimer == null)

{

quizTimer = new System.Windows.Forms.Timer();

quizTimer.Interval = 1000; // 1-second intervals

quizTimer.Tick += QuizTimer\_Tick;

}

timeRemaining = 57 \* 60; // 57 minutes in seconds

quizTimer.Start();

...//remains the same

}

Added new function to change the format

private string FormatTime(int seconds)

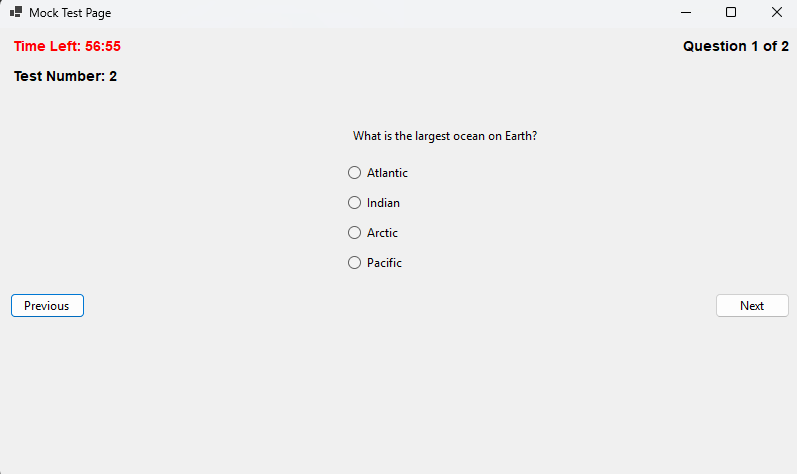
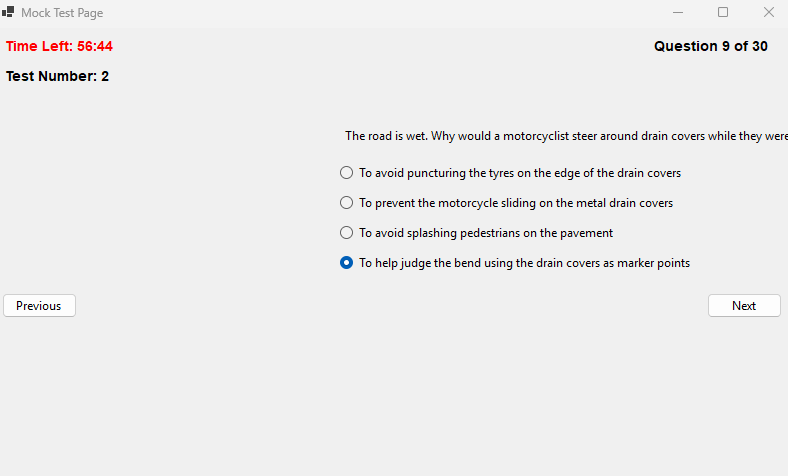
{

int minutes = seconds / 60;

int remainingSeconds = seconds % 60;

return $"{minutes:D2}:{remainingSeconds:D2}";

}

******Adding real traffic questions by putting 30 Questions in 3 mock test each  
  


**Iteration 5:**

**Problem:** If a user wants to mark a question they dont understand how would they?  
**Solution**: I will introduce a "Flag" button that allows me to flag questions I want to know the answer to. These flagged questions will then be displayed on the progress page. I can flag and unflag questions as needed. Additionally, I will introduce a list to save the selected flagged questions for easy tracking.

private List<int> flaggedQuestions = new List<int>();

Making a new Class for it.

private void Flag\_Click(object? sender, EventArgs e)

{

// Toggle flag for the current question

if (flaggedQuestions.Contains(currentQuestionIndex))

{

flaggedQuestions.Remove(currentQuestionIndex);

flagButton.Text = "Flag";

}

else

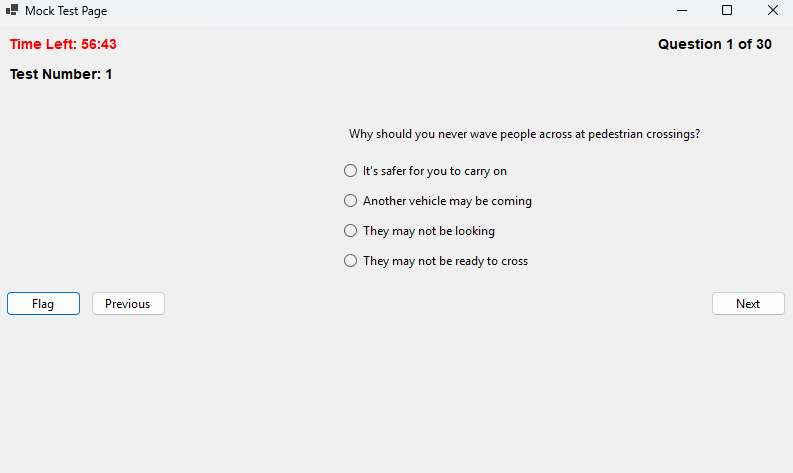
{

flaggedQuestions.Add(currentQuestionIndex);

flagButton.Text = "Unflag";

}

}

****

Made Questions Global as well so it could be accessed by the Progress page(Flagged questions Form)

public static List<List<Question>> AllTests { get; set; } = new List<List<Question>>

{

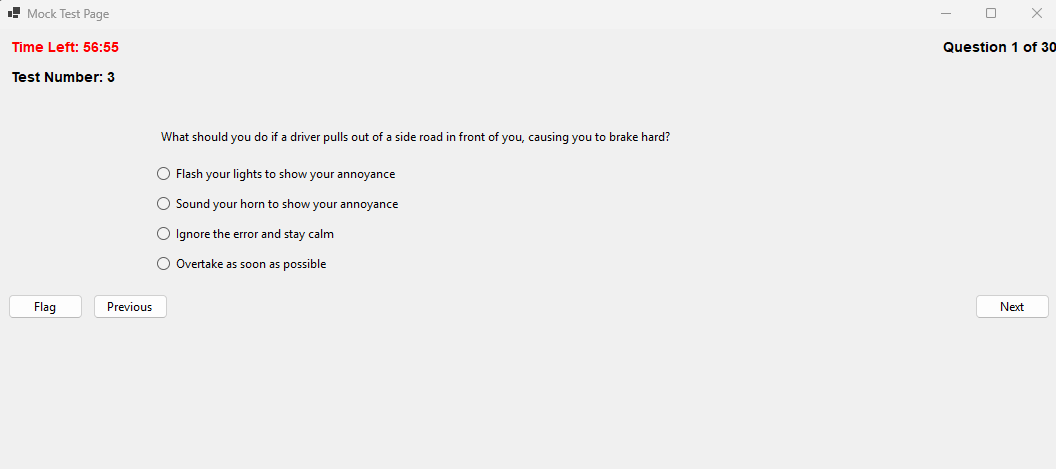
new List<Question>

{

new Question{Text = "Why should you never wave people across at pedestrian crossings?",Options = new List<string>{"It's safer for you to carry on","Another vehicle may be coming","They may not be looking","They may not be ready to cross"},CorrectOptionIndex = 1},

}

}

**  
Problem:** when the user is flagging the question they are still forced to select an option to move ahead  
**Solution:** Change the Next Click Class to include an if which checks if the question is flagged or not

private void Next\_Click(object? sender, EventArgs e)

{

// Check if the current question is flagged

if (flaggedQuestions.Contains(currentQuestionIndex))

{

// Proceed to the next question without requiring an answer

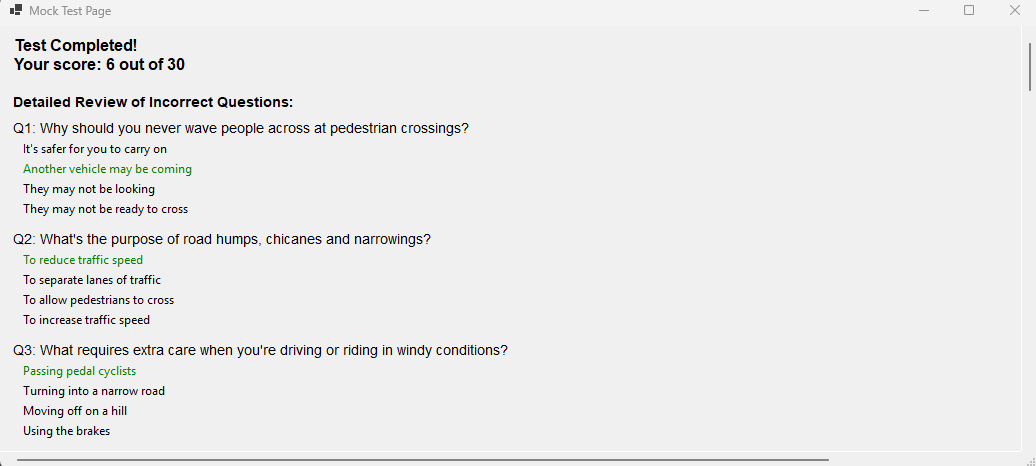
currentQuestionIndex++;

LoadQuestion();

return;

}

}

The Show screen also needs to show what questions the user has gotten right or wrong  
 **Problem:** Showing all the questions without showing user choice

**Solution:** Make and implement a Dictionary which stores the users choice

private Dictionary<int, int> userAnswers = new Dictionary<int, int>();

implement it in next click class so the users choice can be stored

// Store the user's answer

userAnswers[currentQuestionIndex] = selectedOption;

I will change the ShowScore class to incorporate all of the new changes by creating a loop that goes through all of the incorrect questions answered by me. In this loop, red will represent the user’s choice that was wrong, while green will represent the correct answer. This will provide a clear visual representation of the mistakes made and the correct answers.

// Display detailed results for incorrect answers

Label reviewLabel = new Label

{

Text = "Review of Incorrect Questions:",

AutoSize = true,

Font = new Font("Arial", 10, FontStyle.Bold),

Location = new Point(10, yPosition)

};

scrollablePanel.Controls.Add(reviewLabel);

yPosition = reviewLabel.Bottom + 10;

// Loop through the questions and display only incorrect answers

for (int i = 0; i < selectedTest.Count; i++)

{

Question question = selectedTest[i];

int userAnswer = -1; // Default value for no answer

bool isCorrect = false;

// Determine user's selected answer

if (userAnswers.ContainsKey(i))

{

userAnswer = userAnswers[i];

isCorrect = userAnswer == question.CorrectOptionIndex;

}

// Skip correctly answered questions

if (isCorrect)

continue;

// Display the question

Label questionLabel = new Label

{

Text = $"Q{i + 1}: {question.Text}",

AutoSize = true,

Font = new Font("Arial", 10),

Location = new Point(10, yPosition)

};

scrollablePanel.Controls.Add(questionLabel);

yPosition = questionLabel.Bottom + 5;

// Display the options

for (int j = 0; j < question.Options.Count; j++)

{

Label optionLabel = new Label

{

Text = question.Options[j],

AutoSize = true,

Location = new Point(20, yPosition)

};

// Highlight the user's incorrect answer in red

if (j == userAnswer)

{

optionLabel.ForeColor = Color.Red;

}

// Highlight the correct answer in green

else if (j == question.CorrectOptionIndex)

{

optionLabel.ForeColor = Color.Green;

}

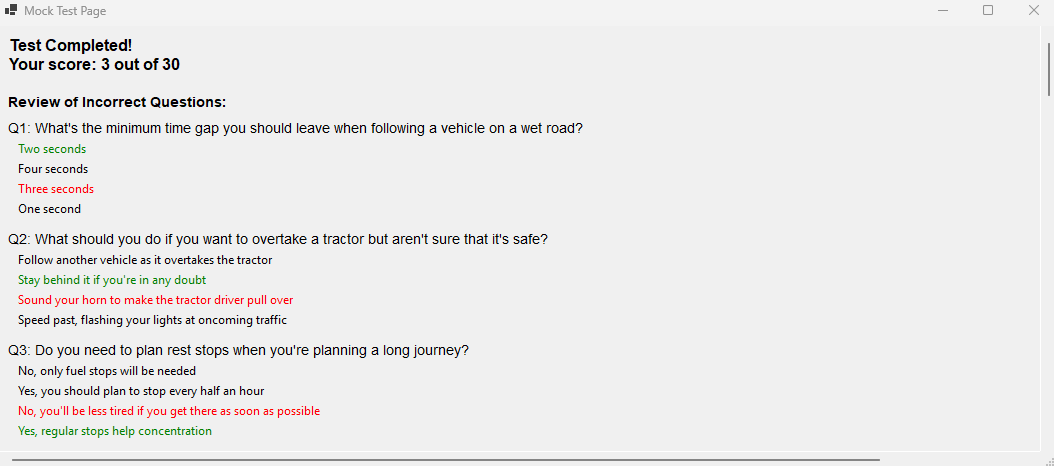
scrollablePanel.Controls.Add(optionLabel);

yPosition = optionLabel.Bottom + 5;

}

yPosition += 10; // Add spacing between questions

}

****

***Review:***

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| Does the system allow users to mark topics as completed? |  |
| Does the system persist the checkbox state even when the user navigates away from the page? |  |
| When can a users click on a particular topic, what do they see as regards all the signs connected to it? |  |
| Does every traffic sign have a clear image and a description? |  |
| Does the progress of each topic update, at the instance when the user or the system, completes or sets the topic to the incomplete status? |  |
| Is it possible for the user to flow from one page to another (topics, progress, settings, signs)? |  |
| On the page showing the progress, must all the topics that were finished be indicated with perfect icons? |  |
| Is it easy to use the software to select topics, to look for signs whereas checkboxes must be manipulative? |  |
| Are users able to remove their findings on specific or on all specific topics in case they want a fresh start? |  |
| Are there any options that allow a user to see their performance over the course of the whole program on a single page? |  |
| Are checkboxes and progress indicators (checkmarks, completion percentages) clearly visible and easy to interact with? |  |
| Can users dynamically change the background color using a ColorDialog in the settings page? |  |
| Can users adjust font size using a slider, and toggle font styles (bold/italic) using checkboxes? |  |
| Are all user settings (background color, font size, font styles) applied globally and persisted between sessions? |  |
| Does the settings page provide real-time updates with previews for font and style changes? |  |
| Are images loaded efficiently without causing performance issues, even with a large number of topics or signs? |  |
| Can users interact with images, such as zooming in for better visibility of signs? |  |
| Does the system handle missing or corrupted image files gracefully by showing placeholders? |  |
| Does the layout remain clean, with proper spacing, alignment, and legibility for readability? |  |
| Is there clear error handling for issues like missing data, image errors, or faulty inputs? |  |
| Does the system load quickly without significant delays or lag when switching between pages or topics? |  |
| Does the system visually distinguish between completed and incomplete topics? |  |
| Is the reset button functional, deleting all relevant data files (scores, flagged questions) with success confirmation? |  |
| Does the system prevent crashes or unexpected behavior from faulty inputs or user actions? |  |
| Can flagged questions be viewed separately for easy access? |  |
| Are all forms and pages centered on the screen when opened? |  |
| Does the system handle high data volumes (e.g., many topics or signs) without performance issues? |  |
| Can additional content, such as multimedia (videos, animations), be added without major redesigns? |  |
| Is the system flexible enough to integrate new topics, signs, or complex features (e.g., quizzes, assessments)? |  |
| Does the overall user interface and experience align with personalization, usability, and consistency goals? |  |

**Prototype 4 - Progress Page**

**Design**

**Overview:**

### The **Progress Page** visually tracks a user's test performance using progress bars and percentages. Progress tracking is necessary for my program to be successful, in order to help users improve their retention ability and tackle key areas where they are not succeeding. This page is therefore designed to dynamically display progress for both mock tests and practice tests. Key features will include:

* **Progress Bars**: Represent scores as a percentage for each test.
* **Dynamic Percentage Calculation**: Adjusts for varying numbers of questions per test.
* **Global Score Tracking**: Integrates scores from both the **Mock Test Page** and **Practice Page**.
* **Completed Topics Tracking**: Shows completed topics with real-time updates.

### **Flowchart Representation:**

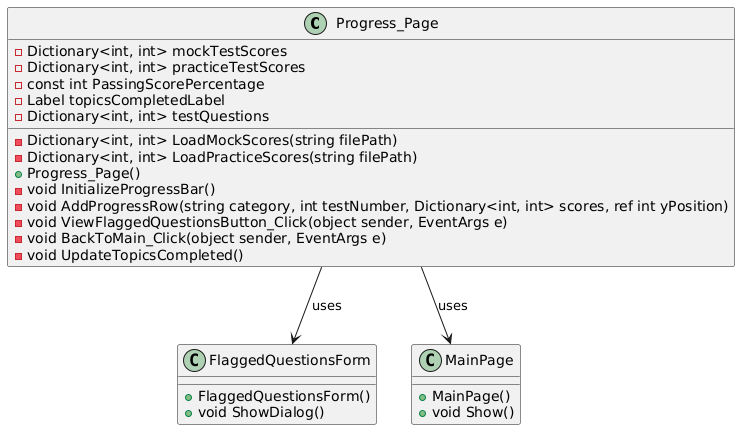
**A diagram of progress bar

Description automatically generated**

### **Decomposition into Computable Sections:**

|  |  |
| --- | --- |
| **Section** | **Justification (suitable for computation because…)** |
| **Displaying Progress for Practice Tests** | Adds the practice test scores to a global score dictionary and displays them alongside the mock test progress for user accessibility. |
| **Global Score Tracking** | Allows the program to store and retrieves these scores from a global dictionary for both mock tests and practice tests – essential for progress tracking which is a key feature necessary for my stakeholders. |
| **Label Identification for Test Types** | Adds labels for example "Mock Test Progress" and "Practice Test Progress" for clarity when displaying progress in examinations. |
| **Completed Topics Tracking** | Uses a **timer** to monitor and update the list of topics completed by the user for accurate progress tracking for the end user. |
| **Dynamic UI Updates** | Creates progress bars and labels dynamically, updating content from the latest data for scores and topics. |

### **States:**

****

Encapsulation

* MockTestScores and practiceTestScores are private variables meaning only this class can access them.
* To draw attention to logical work that has been hidden, methods LoadMockScores and LoadPracticeScores encapsulate.

Abstraction

* AddProgressRow and UpdateTopicsCompleted are examples of using GUI and business logic abstractions.

Inheritance

* Progress\_Page class is a subclass of Form to gain GUI operations.

Polymorphism

* Polymorphism is shown by event handlers BackToMain\_Click, ViewFlaggedQuestionsButton\_Click which work with specific actions.

Summary

* Good use of polymorphism, good use of waranty.
* Uses inheritance with GUI features in mind and polymorphism with user interaction in mind.

### ***Step 1: Tracking User Progress and Scores***

To track user progress (tests taken and scores), I will use a graphical progress bar and labels. The progress percentage will be calculated dynamically based on the number of questions in each test.

**Pseudocode:**

Define testScores as Dictionary

SET testScores = { " KEY " : " test number ",   
" VALUE " : “test score” }

Define testQuestions as Dictionary

SET testScores = { " KEY " : " test number ",   
" VALUE " : “total number of questions” }

FUNCTION SaveScore(testNumber, score)

IF testScores does not contain key testNumber  
 THEN

testScores[testNumber] = score

END IF

END FUNCTION

FUNCTION CalculateScorePercentage(testNumber)

IF testScores contains key testNumber AND testQuestions contains key testNumber  
 THEN

RETURN (testScores[testNumber] \* 100) / testQuestions[testNumber]

ELSE

RETURN 0

END IF

**Reason:**  
Tracking progress requires storing scores for each test in a global dictionary. Dynamic calculation ensures the correct percentage even if tests have varying question counts.

**Approach:**

* Save user scores globally using dictionaries.
* Calculate the percentage score dynamically using the formula:  
  (score / total\_questions) \* 100.

### ***Step 2: Displaying Progress Bars and Labels***

I will display a progress bar for each test, showing the user's score as a percentage. A label next to the progress bar will display the exact percentage or indicate if the test has not been attempted.

**Pseudocode:**

FUNCTION DisplayProgressBarWithPercentage(testNumber)

percentage = CalculateScorePercentage(testNumber)

Minimum(progressBar) = 0

Maximum(progressBar) = 100

Value(progressBar) = percentage

Define percentageLabel as Label

IF percentage > 0 THEN   
 Text(percentageLabel) = (percentage)   
 ELSE   
 OUTPUT "Not Attempted"   
 ENDIF

END FUNCTION

**Reason:**  
Progress bars will allow my user to see a visual representation of their progress and labels help provide clarity with exact percentages, which is also needed for improvement and boosting ‘morale’ in their performance as indicated by my stakeholders for this project.

**Approach:**

|  |
| --- |
| **Insert the Progress Bar to display all scores visually for the user’s convenience.** |
| **Calculate percentages of tests and update the according labels alongside progress bars.** |

### ***Step 3: Integrating Mock and Practice Test Scores***

I will try to make sure that the progress displays to include both Mock and Practice tests by creating individual dictionaries for both.

**Pseudocode:**

Create Class GlobalData

MockTestScores = DICTIONARY

PracticeTestScores = DICTIONARY

FUNCTION ShowScore(testType, testNumber, score)

IF testType == "Mock"

MockTestScores[testNumber] = score

ELSE IF testType == "Practice"

PracticeTestScores[testNumber] = score

END IF

End FUNCTION

FUNCTION DisplayTestProgress()

Define mockProgressLabel AS Label  
 Display Progress Bars for Mock Scores

Define practiceProgressLabel AS Label  
 Display Progress Bars for Practice Scores

End FUNCTION

**Reason:**  
Tracking Mock and Practice tests separately will ensure even more clarity for the evaluation of progress on the Progress page for the user and indicate possible discrepancy between Timed test & normal practice questions – to help boost performance in the real exam

**Approach:**

|  |
| --- |
| **Host two separate dictionaries in my program to store Mock and Practice Test Scores** |
| **Display both progress metrics individually in their own distinct sections for easier viewing.** |

### ***Step 4: Handling Topics Completed with a Timer***

To track completed topics, I will add a timer that checks for user updates every second and dynamically displays the completed topics.

To automatically track topics that have been completed in the program, I aim to add a timer that will check for user actions updates every second and therefore dynamically display the completed topics.

**Pseudocode:**

**Define updateTimer as a Timer**

**Define topicsCompletedLabel as a label**

FUNCTION InitializeTopicsCompleted()

Interval(updateTimer) = 1 second  
 Tick(updateTimer)  
 Call UpdateTopicsCompleted() function

FUNCTION UpdateTopicsCompleted()

completedTopics = CompletedTopicsCount()  
 IF completedTopics > 0   
 topicsCompletedLabel = ("Topics Completed: ", completedTopics)  
 ELSE   
 topicsCompletedLabel = "Topics Completed: None"

END FUNCTION

**Reason:**  
Using a timer ensures that updates are fetched and displayed dynamically without requiring user intervention.

**Approach:**

|  |
| --- |
| **Create a Timer that checks for updates every second during execution of my program.** |
| **Update the topics label based on the count of completed topics by user.** |

### ***Step 5: Final Workflow Summary***

The finalized workflow for the **Progress Page** will consist of the following:

1. **Tracking Progress:** Save user’s scores and calculate percentages for each test taken.
2. **Displaying Progress Bars:** Progress bars and labels for Mock and Practice tests will be shown separately for better UX.
3. **Integrating Practice Scores:** Implement and use dictionaries to track Mock and Practice test scores throughout the execution of the entire program.
4. **Dynamic Updates:** Use a timer to track topics that have been completed and update this in the display in real-time.

**Key Features Implemented:**

* Graphical progress bars for visual progress tracking.
* Dynamic percentage calculation based on total test questions.
* Separate tracking for Mock and Practice tests.
* Real-time topic completion updates using a timer.

**Development**

***Overview:***

The project tracks and displays user progress in mock and practice tests using dynamic progress bars. It calculates progress based on the number of correct answers and the total questions per test, with scores stored globally. The interface also allows for flagging questions and displays a timer to track completed topics, updating the user's progress continuously. This system enhances user engagement by providing real-time feedback on test performance and completed content.

***Development and Debugging (broken down into chronological iterations/updates):***

**Iteration 1:**

**Problem:** How can I track the progress of the user, including how many tests they have taken and their scores on those tests (graphically)? How will the score percentage be calculated? **Solution:** To implement a graphical progress bar for the progress page, I can use the ProgressBar control in Windows Forms and a Label to display the score percentage. I can create a dynamic progress bar for each test, where the value of the progress bar reflects the user's score as a percentage. I will calculate the score percentage based on the number of correct answers (e.g., 10 questions per test, or adjust this to the actual number of questions in the test).

Next to each progress bar, a Score Display label will show the score percentage or indicate if the test has not been attempted. When navigating to the Progress\_Page, I will pass a dictionary containing the scores for each test to ensure all progress is displayed correctly.

private void InitializeProgressBar()

{

Controls.Clear();

Label titleLabel = new Label

{

Text = "Progress Overview",

Font = new Font("Arial", 14, FontStyle.Bold),

AutoSize = true,

Location = new Point((ClientSize.Width - 200) / 2, 20)

};

Controls.Add(titleLabel);

int yPosition = titleLabel.Bottom + 30;

// Display progress bar

ProgressBar progressBar = new ProgressBar

{

Minimum = 0,

Maximum = 100,

Value = testScores.ContainsKey(testNumber)

? (testScores[testNumber] \* 100) / 10

: 0, // Assuming each test has 10 questions

Size = new Size(200, 20),

Location = new Point(100, yPosition)

};

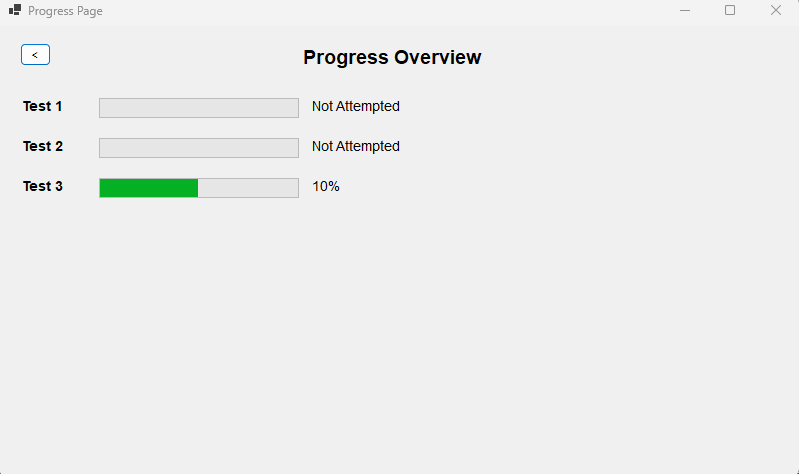
Controls.Add(progressBar);

}

**Problem:** How to Integrate it with the Mock test page?  
**Solution:** To integrate with my MockTest\_Page, updating the ShowScore method so it can save the score in a global object. Then, pass the scores to the Progress\_Page.

private Dictionary<int, int> testScores = new Dictionary<int, int>(); // Global dictionary to track scores

// Save the score for the current test if (!testScores.ContainsKey(testNumber)) testScores[testNumber] = score;



**Iteration 2:**

To Dynamically calculate the progress bar based on the number of questions in the test we do the following:

Dictionary<int, int> testQuestions = new Dictionary<int, int>

{

{ 1, 10 }, // Test 1 has 10 questions

{ 2, 8 }, // Test 2 has 8 questions

{ 3, 12 } // Test 3 has 12 questions

};

// Display progress bar

ProgressBar progressBar = new ProgressBar

{

Minimum = 0,

Maximum = 100,

Value = testScores.ContainsKey(testNumber) && testQuestions.ContainsKey(testNumber)

? (testScores[testNumber] \* 100) / testQuestions[testNumber]

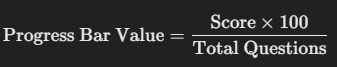
: 0, // Default to 0 if the test number is not found

Size = new Size(200, 20),

Location = new Point(100, yPosition)

};

If the test score for the current test (testNumber) exists in testScores and the total number of questions for that test exists in testQuestions, calculate the percentage:



Otherwise, default to 0. This logic ensures the progress bar adjusts according to the actual number of questions in the test rather than a fixed assumption of 10.  
  
**Problem:** The text next to the progress bar is 10%/20% it is being calculated by Multiplying the score by 10,Assuming that the number of question is 10  
the percentage should be calculated with the dynamic question logic now

Text = testScores.ContainsKey(testNumber)

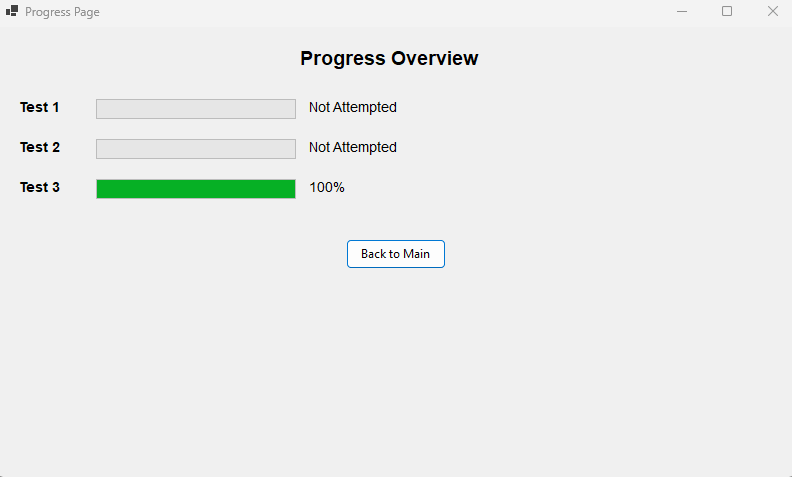
? $"{testScores[testNumber] \* 10}%"

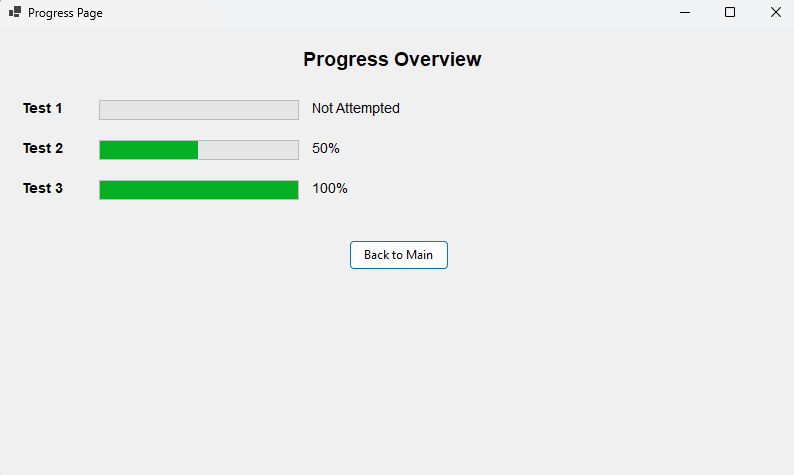
: "Not Attempted",

**Solution:** I can use a Label that dynamically calculates and displays the percentage based on the testScores and testQuestions.   
Calculate Percentage percentage is dynamically calculated now using the formula

****

If the score or total number of questions is missing, it defaults to 0%.  
The progress bar value is now set using the percentage.  
A Label is added to display the percentage next to the progress bar.  
The logic works for any number of questions and any score.





**Iteration 3:**

It only shows the progress of the mock test right now I want to change so that so it also shows the progress of practice page. I have to add A variable to score practice score globally

public static *class* GlobalData

{

public static Dictionary<int, int> TestScores { *get*; *set*; } = new Dictionary<int, int>();

public static Dictionary<int, int> PracticeScores { *get*; *set*; } = new Dictionary<int, int>(); // Practice test scores

}

I then have to store the scores as well

private void ShowScore()

{

// Clear the form

Controls.Clear();

// Save the score globally with the test index

if (selectedTest != null)

{

int testIndex = tests.IndexOf(selectedTest) + 1; // Test index starts from 1

GlobalData.PracticeScores[testIndex] = score; // Save the score for this test

}

}

I also changed the label to better identify the tests and show progress under

// Add mock test progress

Label mockProgressLabel = new Label

{

Text = "Mock Test Progress",

..//remains the same

};

foreach (*var* testNumber in testQuestions.Keys)

{

AddProgressRow("Test", testNumber, mockTestScores, ref yPosition);

}

// Add practice test progress

Label practiceProgressLabel = new Label

{

Text = "Practice Test Progress",

..//remains the same

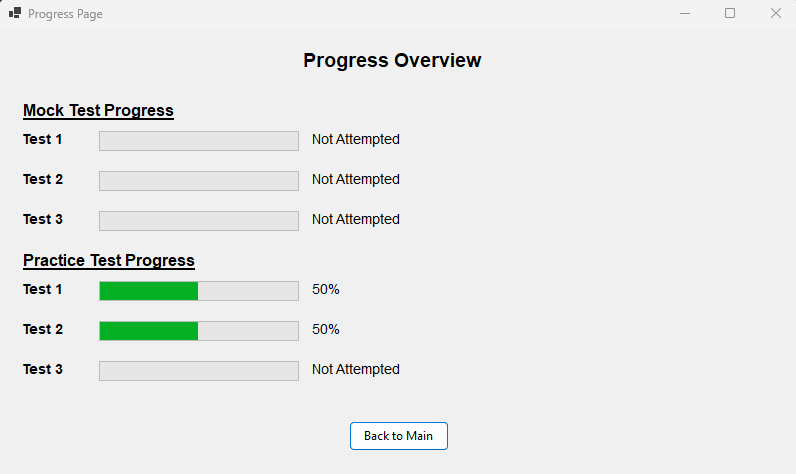
};

foreach (*var* testNumber in testQuestions.Keys)

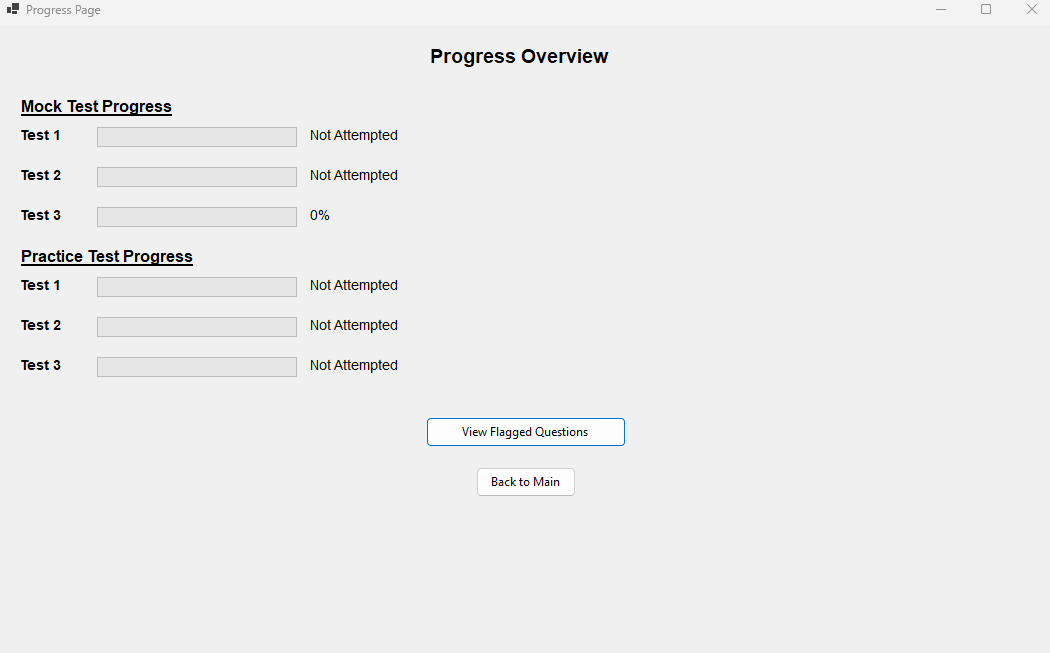
{

AddProgressRow("Test", testNumber, practiceTestScores, ref yPosition);

}

****Iteration 4:**

I will show the Flagged Questions here under a Flag Question Button, This Button would open a new form on which every flagged question alongside its answer would be there.

******

**Iteration 5:**

**Problem:** After adding the checkbox for completed topics, how can I track what the user has completed? And if they go back and complete it later, how will that be updated?  
**Solution**: I will introduce a timer that checks what the user has selected to be completed. This timer will run every second. If the user hasn't completed any topics, it will display "None."  
add a timer and the topics completed

topicsCompletedLabel = new Label

{

Text = "Topics Completed: None",

Font = new Font("Arial", 12, FontStyle.Bold),

AutoSize = true,

Location = new Point(20, yPosition )

};

Controls.Add(topicsCompletedLabel);

System.Windows.Forms.Timer updateTimer = new System.Windows.Forms.Timer

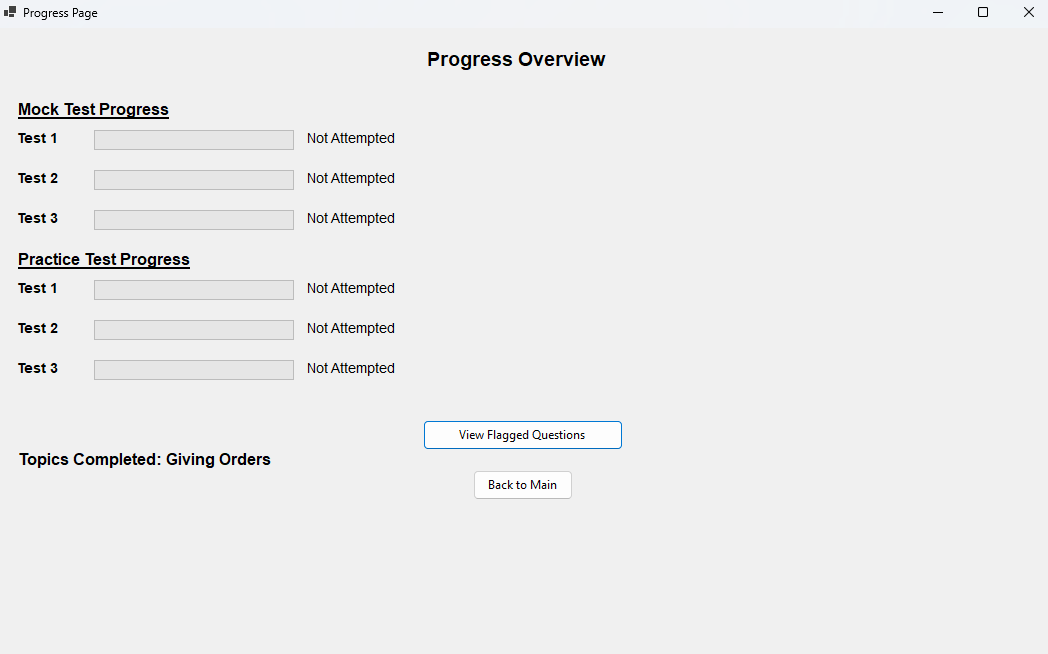
{

Interval = 1000 // Check for updates every second

};

updateTimer.Tick += (s, e) => UpdateTopicsCompleted();

updateTimer.Start();

***  
  
Review:***

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| Does the system allow users to mark topics as completed? |  |
| Does the system persist the checkbox state even when the user navigates away from the page? |  |
| When can a users click on a particular topic, what do they see as regards all the signs connected to it? |  |
| Does every traffic sign have a clear image and a description? |  |
| Does the progress of each topic update, at the instance when the user or the system, completes or sets the topic to the incomplete status? |  |
| Is it possible for the user to flow from one page to another (topics, progress, settings, signs)? |  |
| On the page showing the progress, must all the topics that were finished be indicated with perfect icons? |  |
| Is it easy to use the software to select topics, to look for signs whereas checkboxes must be manipulative? |  |
| Are users able to remove their findings on specific or on all specific topics in case they want a fresh start? |  |
| Are there any options that allow a user to see their performance over the course of the whole program on a single page? |  |
| Are checkboxes and progress indicators (checkmarks, completion percentages) clearly visible and easy to interact with? |  |
| Can users dynamically change the background color using a ColorDialog in the settings page? |  |
| Can users adjust font size using a slider, and toggle font styles (bold/italic) using checkboxes? |  |
| Are all user settings (background color, font size, font styles) applied globally and persisted between sessions? |  |
| Does the settings page provide real-time updates with previews for font and style changes? |  |
| Are images loaded efficiently without causing performance issues, even with a large number of topics or signs? |  |
| Can users interact with images, such as zooming in for better visibility of signs? |  |
| Does the system handle missing or corrupted image files gracefully by showing placeholders? |  |
| Does the layout remain clean, with proper spacing, alignment, and legibility for readability? |  |
| Is there clear error handling for issues like missing data, image errors, or faulty inputs? |  |
| Does the system load quickly without significant delays or lag when switching between pages or topics? |  |
| Does the system visually distinguish between completed and incomplete topics? |  |
| Is the reset button functional, deleting all relevant data files (scores, flagged questions) with success confirmation? |  |
| Does the system prevent crashes or unexpected behavior from faulty inputs or user actions? |  |
| Can flagged questions be viewed separately for easy access? |  |
| Are all forms and pages centered on the screen when opened? |  |
| Does the system handle high data volumes (e.g., many topics or signs) without performance issues? |  |
| Can additional content, such as multimedia (videos, animations), be added without major redesigns? |  |
| Is the system flexible enough to integrate new topics, signs, or complex features (e.g., quizzes, assessments)? |  |
| Does the overall user interface and experience align with personalization, usability, and consistency goals? |  |

**Prototype 5 - Flagged Questions Page**

**Design**

**Overview:**

The **Flagged Questions Page** is designed to display all the questions that a user has flagged during a test. It extracts flagged question indices, retrieves the corresponding question details, and displays the question text and the correct answer in a clean, scrollable interface.

### **Decomposition into Computable Sections:**

|  |  |
| --- | --- |
| **Section** | **Justification (suitable for computation because…)** |
| **Retrieving Flagged Questions** | Loops through all flagged question indices to fetch the respective question details. |
| **Accessing Test Data** | Retrieves the flagged questions using the GlobalData.AllTests dictionary to ensure all tests are accessible. |
| **Displaying Flagged Questions** | Dynamically creates labels for each question, displaying both the question text and the correct answer. |
| **Scrollable UI Creation** | Adds the dynamically generated question labels to a scrollable panel for an organized display. |
| **Dynamic Positioning of Labels** | Adjusts the yPosition to space out the displayed questions properly within the scrollable panel. |

### **States:**

**A diagram of a question

Description automatically generated with medium confidence**

Encapsulation

* Houses flagged questions’ data and code for showing flagged questions.
* Starting configurations of UI settings are done holistically with limited access.

Inheritance

* Uses form class to inherit so as to reuse their functions and add functionalities to them.

Polymorphism

* Other real time event-driven methods that you can see polymorphism are closeButton.Click.

Abstraction

* Others encapsulates data fetching and exception handling from users to provide a simplified user interphase.

Summary

* A clear distinction between the program’s organization and the graphical user interface.
* Call super class functions to inherit primary controls and implements polymorphism to handle dynamism in UI.
* Hides what it expects to compute for the output from the user.

### **Steps to Implement the Flagged Questions Page:**

1. **Setup Data Structure**
   * This will guarantee that the positions of the marked questions are kept in an appropriate format for example a dictionary with a list of the positions of the questions marked as the value and the test number as the key.
2. **Accessing Flagged Questions**
   * Use the test number to access flagged question position.
   * Retrieve the corresponding questions and their correct answers from Global Data.
3. **Dynamic UI Creation**
   * Iterate through the questions that have been flagged and create labels for each question and it’s corresponding correct answer.
   * Add these labels to a **panel**.
4. **Scrollable UI Integration**
   * To manage situations where there are several marked queries, arrange all generated content inside the panel.

### **Final Steps and Functionality:**

1. **Store Flagged Positions**: Ensure flagged questions are tracked during the test.
2. **Pass Flagged Questions**: Provide flagged positions when navigating to the Flagged Questions Page.
3. **Retrieve Data**: Fetch flagged questions from Global Data.
4. **Display Data**: Labels are dynamically created and positioned to show the text of the question and the appropriate response.
5. **Scrollable UI**: Make sure a scrollable interface is available for viewing all of the questions.

**Development**

***Overview:***

In Iteration 1, the goal is to display all flagged questions from a test on a separate page. This involves iterating through a collection of flagged questions, retrieving the question text and its correct answer, and dynamically displaying them in a scrollable panel for easy viewing. Each flagged question is represented by a label showing its text and the correct answer, and the layout adjusts the position of each question as it is added to the panel.

***Development and Debugging (broken down into chronological iterations/updates):***

**Iteration 1:**  
For getting all the flagged questions and showing them on one page

foreach (int questionIndex in test.Value)

{

Question question = GlobalData.AllTests[test.Key - 1][questionIndex]; // Get the flagged question

Label questionLabel = new Label

{

Text = $"Q: {question.Text}\nA: {question.Options[question.CorrectOptionIndex]}",

Font = new Font("Arial", 10, FontStyle.Regular),

AutoSize = true,

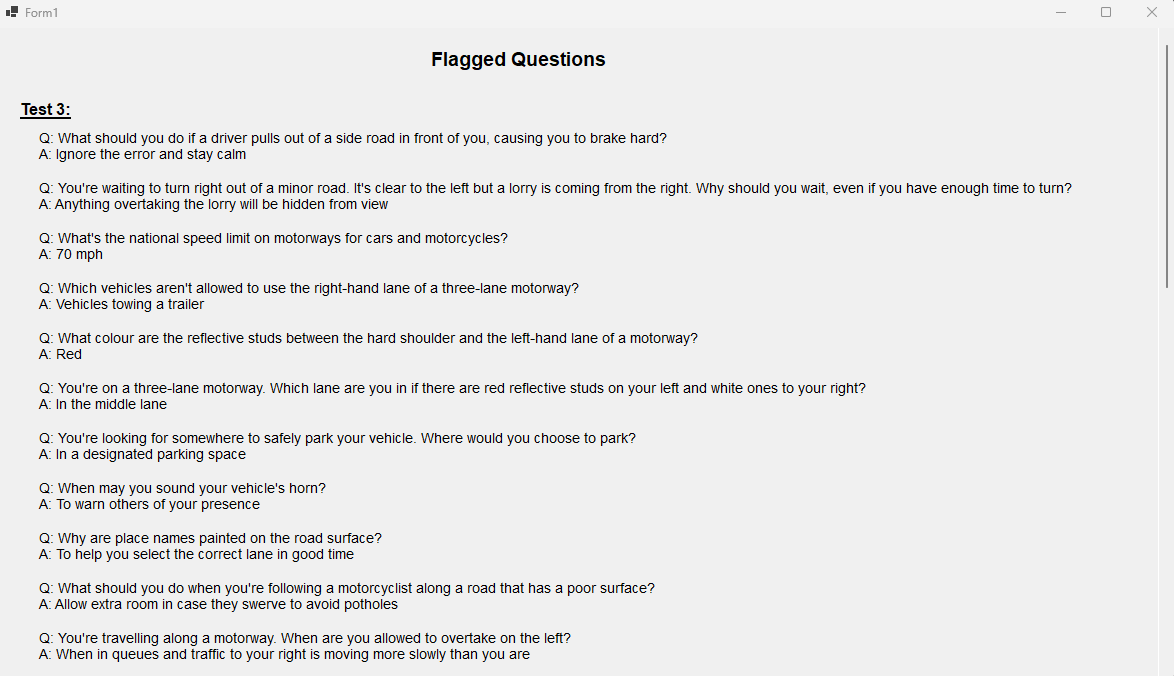
Location = new Point(40, yPosition)

};

scrollablePanel.Controls.Add(questionLabel);

yPosition += 50;

}



***Review:***

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| Does the system allow users to mark topics as completed? |  |
| Does the system persist the checkbox state even when the user navigates away from the page? |  |
| When can a users click on a particular topic, what do they see as regards all the signs connected to it? |  |
| Does every traffic sign have a clear image and a description? |  |
| Does the progress of each topic update, at the instance when the user or the system, completes or sets the topic to the incomplete status? |  |
| Is it possible for the user to flow from one page to another (topics, progress, settings, signs)? |  |
| On the page showing the progress, must all the topics that were finished be indicated with perfect icons? |  |
| Is it easy to use the software to select topics, to look for signs whereas checkboxes must be manipulative? |  |
| Are users able to remove their findings on specific or on all specific topics in case they want a fresh start? |  |
| Are there any options that allow a user to see their performance over the course of the whole program on a single page? |  |
| Are checkboxes and progress indicators (checkmarks, completion percentages) clearly visible and easy to interact with? |  |
| Can users dynamically change the background color using a ColorDialog in the settings page? |  |
| Can users adjust font size using a slider, and toggle font styles (bold/italic) using checkboxes? |  |
| Are all user settings (background color, font size, font styles) applied globally and persisted between sessions? |  |
| Does the settings page provide real-time updates with previews for font and style changes? |  |
| Are images loaded efficiently without causing performance issues, even with a large number of topics or signs? |  |
| Can users interact with images, such as zooming in for better visibility of signs? |  |
| Does the system handle missing or corrupted image files gracefully by showing placeholders? |  |
| Does the layout remain clean, with proper spacing, alignment, and legibility for readability? |  |
| Is there clear error handling for issues like missing data, image errors, or faulty inputs? |  |
| Does the system load quickly without significant delays or lag when switching between pages or topics? |  |
| Does the system visually distinguish between completed and incomplete topics? |  |
| Is the reset button functional, deleting all relevant data files (scores, flagged questions) with success confirmation? |  |
| Does the system prevent crashes or unexpected behavior from faulty inputs or user actions? |  |
| Can flagged questions be viewed separately for easy access? |  |
| Are all forms and pages centered on the screen when opened? |  |
| Does the system handle high data volumes (e.g., many topics or signs) without performance issues? |  |
| Can additional content, such as multimedia (videos, animations), be added without major redesigns? |  |
| Is the system flexible enough to integrate new topics, signs, or complex features (e.g., quizzes, assessments)? |  |
| Does the overall user interface and experience align with personalization, usability, and consistency goals? |  |

**Prototype 6 - Traffic Signs Page**

**Design**

**Overview:**

The **Traffic Signs Page** allows users to mark topics as completed, track their progress, and study traffic signs with a combination of text and images. This page implements checkboxes for progress tracking, persistent state storage, navigation to topic-specific pages, and a grid to display images with descriptions.

### **Decomposition into Computable Section**

|  |  |
| --- | --- |
| **Section** | **Justification (suitable for computation because…)** |
| **Restoring Checkbox States** | Ensures that previously checked topics remain checked when reopening the Traffic Signs Page. |
| **Updating Checkbox States** | Tracks changes to checkboxes, updates the state in a persistent storage, and maintains completed topics. |
| **Navigating to Topic Pages** | Allows users to click on a topic and view all related traffic signs with images and text. |
| **Implementing DataGrid for Images/Text** | Displays traffic signs with their descriptions in a structured format using a scrollable DataGridView. |
| **Dynamically Adding Rows** | Ensures that new signs with images and text can be dynamically added to the grid |

### **Status:**

**A screenshot of a computer

Description automatically generated**

Encapsulation

* Continued data: CompletdTopics, CheckboxStates , and methods located in Traffic\_Signs\_page.
* Close to internals and hides file operations to the externally called methods.

Inheritance

* Subclassed from Form which uses event handling and user interface rendering.

Polymorphism

* Event handlers (Giving\_Order\_Complete\_CheckedChanged) are polymorphic.

Abstraction

* There are various methods like UpdateCompletedTopics, which encapsulate the file handling of the abstracts and the UI operations.

Summary

* Data hide and dialog box implementation.
* It inherits from Form for good design.
* Polymorphism does apply here in event handling and gets to abstract internal logic to simplify something.

### ***Step 1: Persistent Checkbox States***

**To persist checkbox states across page reloads, I will use a static dictionary.** This dictionary will hold the state (checked/unchecked) for each traffic sign category.

**Pseudocode:**

DEFINE checkboxstate as a static dictionary

{

"Giving Orders": FALSE,

"Warning Signs": FALSE,

"Direction Signs": FALSE,

"Information Signs": FALSE,

"Road Work Signs": FALSE

}

**Reason:**  
Storing checkbox states in a static dictionary will allow for persistence during navigation and make sure that the states remain consistent throughout execution.

**Approach:**

* Use a static dictionary to hold the state of each checkbox.
* The dictionary allows easy updates and retrieval of checkbox states.

### **Step 2: Restoring Checkbox States**

**On page load, checkbox states will be restored using the values from the dictionary.**

**Pseudocode:**

FUNCTION RestoreCheckboxStates()

Giving\_Order\_Complete.Checked = CheckboxStates["Giving Orders"]

Warning\_Signs\_Complete.Checked = CheckboxStates["Warning Signs"]

Direction\_Signs\_Complete.Checked = CheckboxStates["Direction Signs"]

Information\_Signs\_Complete.Checked = CheckboxStates["Information Signs"]

Road\_Work\_Complete.Checked = CheckboxStates["Road Work Signs"]

END FUNCTION

**Reason:**  
Restoring the state of checkboxes ensures a seamless user experience, so users don’t lose progress after navigating or reloading the page.

**Approach:**

* Retrieve values from the static dictionary.
* Set the corresponding checkbox state dynamically.

### ***Step 3: Updating Checkbox States and Completed Topics***

**Checkbox interactions will dynamically update both the dictionary and a list of completed topics.**

**Pseudocode:**

FUNCTION UpdateCompletedTopics(topic, isCompleted)

IF IsCompleted = TRUE

IF topic = NOT IN CompletedTopics

Add topic to CompletedTopics

ELSE

Delete topic from CompletedTopics

END IF

CheckboxStates[topic] = IsCompleted

END FUNCTION

**Reason:**  
Tracking completed topics alongside updating checkbox states ensures data consistency and enables progress tracking.

**Approach:**

* Update the CheckboxStates dictionary when a checkbox is clicked.
* Maintain a list of completed topics for progress tracking.

### ***Step 4: Navigation to Topic-Specific Pages***

**Clicking on a checkbox or button will navigate users to a detailed page for the corresponding topic.**

**Pseudocode:**

FUNCTION OrdersSignClick

CREATE nextForm AS new Orders\_Signs()

CALL nextForm.Show()

Hide current form

END FUNCTION

**Reason:**  
Navigating to specific pages allows users to dive deeper into detailed information about each traffic sign type.

**Approach:**

* Attach click events to each topic.
* Create and show the corresponding form while hiding the current page.

### ***Step 5: Displaying Traffic Signs in a DataGridView***

**Traffic signs and descriptions will be displayed dynamically using a DataGridView.**

**Pseudocode:**  
***Initialize Grid:***

FUNCTION InitializeGrid()

Define signsGridView

Define imageColumn

Add(imageColumn)(Columns(signsGridView))

Define infoColumn

Add(infoColumn)(Columns(signsGridView))

AddSignRow(signsGridView, "Signs with red circles.", PIC.png)

AddSignRow(signsGridView, "Entry to 20 mph zone", PIC.png)

AddSignRow(signsGridView, "End of 20 mph zone", PIC.png)

END FUNCTION

***Adding Rows:***

FUNCTION AddSignRow(grid, info, imagePath)

imageFullPath = ("Signs\_Giving\_Order", ImagePath)

Define SignImg AS Image   
 SignImg = FromFile(Image)(imageFullPath)  
END FUNCTION

**Reason:**  
A DataGridView provides a structured and visually appealing way to display traffic signs alongside their descriptions.

**Approach:**

* Use DataGridView with image and text columns.
* Dynamically add rows for each traffic sign using images stored locally.

### ***Final Workflow Summary***

The finalized workflow for the Traffic Signs Page will include:

1. **Tracking Checkbox States**: Use a static dictionary to persist checkbox states across page reloads.
2. **Restoring States**: Retrieve and set checkbox states dynamically on page load.
3. **Dynamic Updates**: Update the dictionary and completed topics list as checkboxes are interacted with.
4. **Navigation**: Enable navigation to detailed pages for each traffic sign topic.
5. **Displaying Signs**: Use a DataGridView to show traffic signs and descriptions.

**Development**

***Overview:***

This prototype aims to enhance a user's learning experience by tracking progress on traffic signs topics and displaying relevant sign images with descriptions. It includes two main features:

1. **Progress Tracking:**Users can mark topics as completed via checkboxes. Their progress is stored in a persistent dictionary, ensuring that their selections are remembered even when they navigate away from the page. The system dynamically updates the checkbox states to reflect the user's progress.
2. **Sign Study Interface:**For each traffic signs topic, a dedicated page displays related signs with images and descriptions. A DataGridView is utilized to show these signs in a scrollable format, with an image column and a text column for additional information. This interface allows users to visually study and learn the traffic signs.

The prototype efficiently tracks user progress and provides an interactive study interface for traffic signs, helping users easily learn and monitor their advancement.

***Development and Debugging (broken down into chronological iterations/updates):***

**Iteration 1:  
Problem:** How to know what topics are completed by the user?  
**Solution:** I will add a checkbox under every topic so that when the user completes a topic, they can check it. This allows the user to track their progress and mark topics as completed.

public static *List*<string> CompletedTopics { get; private set; } = new *List*<*string*>();

public Traffic\_Signs\_page()

{

InitializeComponent();

}

private *void* Giving\_Order\_Complete\_CheckedChanged(object sender, *EventArgs* e)

{

UpdateCompletedTopics("Giving Orders", Giving\_Order\_Complete.Checked);

}

private *void* Warning\_Signs\_Complete\_CheckedChanged(object sender, *EventArgs* e)

{

UpdateCompletedTopics("Warning Signs", Warning\_Signs\_Complete.Checked);

}

private *void* Direction\_Signs\_Complete\_CheckedChanged(object sender, *EventArgs* e)

{

UpdateCompletedTopics("Direction Signs", Direction\_Signs\_Complete.Checked);

}

private *void* Information\_Signs\_Complete\_CheckedChanged(object sender, *EventArgs* e)

{

UpdateCompletedTopics("Information Signs", Information\_Signs\_Complete.Checked);

}

private *void* Road\_Work\_Complete\_CheckedChanged(object sender, *EventArgs* e)

{

UpdateCompletedTopics("Road Work Signs", Road\_Work\_Complete.Checked);

}

private *void* UpdateCompletedTopics(string topic, bool isCompleted)

{

if (isCompleted)

{

if (!CompletedTopics.Contains(topic))

{

CompletedTopics.Add(topic);

}

}

else

{

CompletedTopics.Remove(topic);

}

}

**Obstacle:** when the user goes back to the menu and open traffic page again the checkbox becomes empty  
**Solution:** The issue arises because the state of the checkboxes (whether they are checked or not) is not being preserved when navigating away from the Traffic\_Signs\_page and reopening it. This happens because each time the form is opened, a new instance of Traffic\_Signs\_page is created, and the checkboxes are reset to their default state (unchecked).

To fix this, I will store the state of the checkboxes (whether they are checked or unchecked) in a persistent location, such as a global variable, a file, or a database. When the Traffic\_Signs\_page is reopened, I can retrieve the saved state and update the checkboxes accordingly.

public static *List*<string> CompletedTopics { get; private set; } = new *List*<*string*>();

public Traffic\_Signs\_page()

{

InitializeComponent();

}

private *void* Giving\_Order\_Complete\_CheckedChanged(object sender, *EventArgs* e)

{

UpdateCompletedTopics("Giving Orders", Giving\_Order\_Complete.Checked);

}

private *void* Warning\_Signs\_Complete\_CheckedChanged(object sender, *EventArgs* e)

{

UpdateCompletedTopics("Warning Signs", Warning\_Signs\_Complete.Checked);

}

private *void* Direction\_Signs\_Complete\_CheckedChanged(object sender, *EventArgs* e)

{

UpdateCompletedTopics("Direction Signs", Direction\_Signs\_Complete.Checked);

}

private *void* Information\_Signs\_Complete\_CheckedChanged(object sender, *EventArgs* e)

{

UpdateCompletedTopics("Information Signs", Information\_Signs\_Complete.Checked);

}

private *void* Road\_Work\_Complete\_CheckedChanged(object sender, *EventArgs* e)

{

UpdateCompletedTopics("Road Work Signs", Road\_Work\_Complete.Checked);

}

private *void* UpdateCompletedTopics(string topic, bool isCompleted)

{

if (isCompleted)

{

if (!CompletedTopics.Contains(topic))

{

CompletedTopics.Add(topic);

}

}

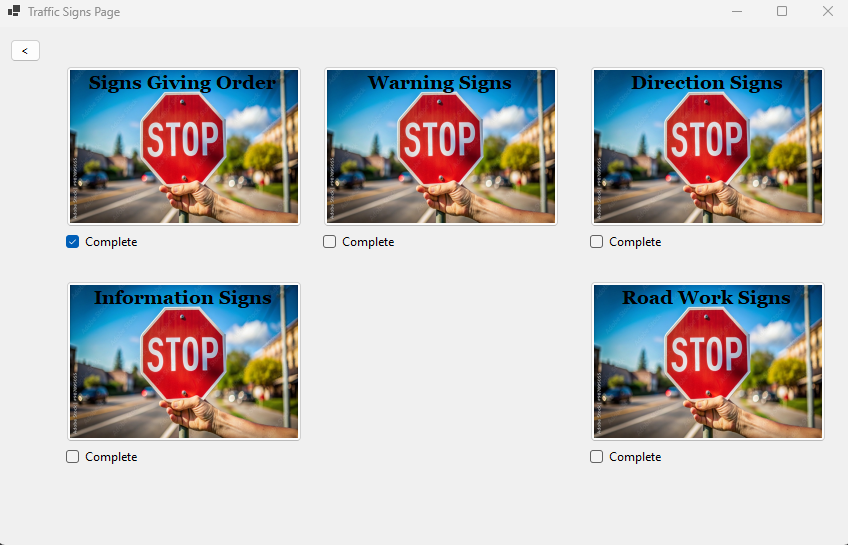
else

{

CompletedTopics.Remove(topic);

}

}

******

**Iteration 2:**

**Problem:** How to put pictures with text so the user can study signs from?  
**Solution:** I will add a new page under each topic so that the user can see what signs belong to that particular topic. When the user clicks on a topic, they will be directed to a page that displays all the relevant signs under that topic.

private void Orders\_Signs\_Click(object sender, EventArgs e)

{

Orders\_Signs nextForm = new Orders\_Signs();

nextForm.Show();

this.Hide();

}

**Problem:** How to put pictures with text?  
**Solution:** Implementation of a datagrid with scroll wheel

private void InitializeGrid()

{

// Create and configure DataGridView

DataGridView signsGridView = new DataGridView

{

Dock = DockStyle.Fill,

AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill,

RowTemplate = { Height = 100 }, // Adjust row height to fit images

AllowUserToAddRows = false,

ReadOnly = true

};

// Add columns

DataGridViewImageColumn imageColumn = new DataGridViewImageColumn

{

HeaderText = "Sign Image",

Name = "ImageColumn",

ImageLayout = DataGridViewImageCellLayout.Zoom // Adjust image display

};

signsGridView.Columns.Add(imageColumn);

DataGridViewTextBoxColumn infoColumn = new DataGridViewTextBoxColumn

{

HeaderText = "Information",

Name = "InfoColumn"

};

signsGridView.Columns.Add(infoColumn);

// Add data

AddSignRow(signsGridView, "Signs with red circles are mostly prohibitive.\r\nPlates below signs qualify their message.", "Blank.png");

AddSignRow(signsGridView, "Entry to 20 mph zone", "Entry\_to\_20\_mph\_zone.png");

AddSignRow(signsGridView, "End of 20 mph zone", "End\_of\_20\_mph\_zone.png");

if (signsGridView.Rows.Count > 1) // Ensure the row exists

{

signsGridView.Rows[0].DefaultCellStyle.Font = new Font("Arial", 10, FontStyle.Bold);

}

// Add DataGridView to form

Controls.Add(signsGridView);

}

private void AddSignRow(DataGridView grid, string info, string imagePath)

{

// Load image

string appDirectory = AppDomain.CurrentDomain.BaseDirectory;

string imageFullPath = System.IO.Path.Combine(appDirectory, "Signs\_Giving\_Order", imagePath);

if (!System.IO.File.Exists(imageFullPath))

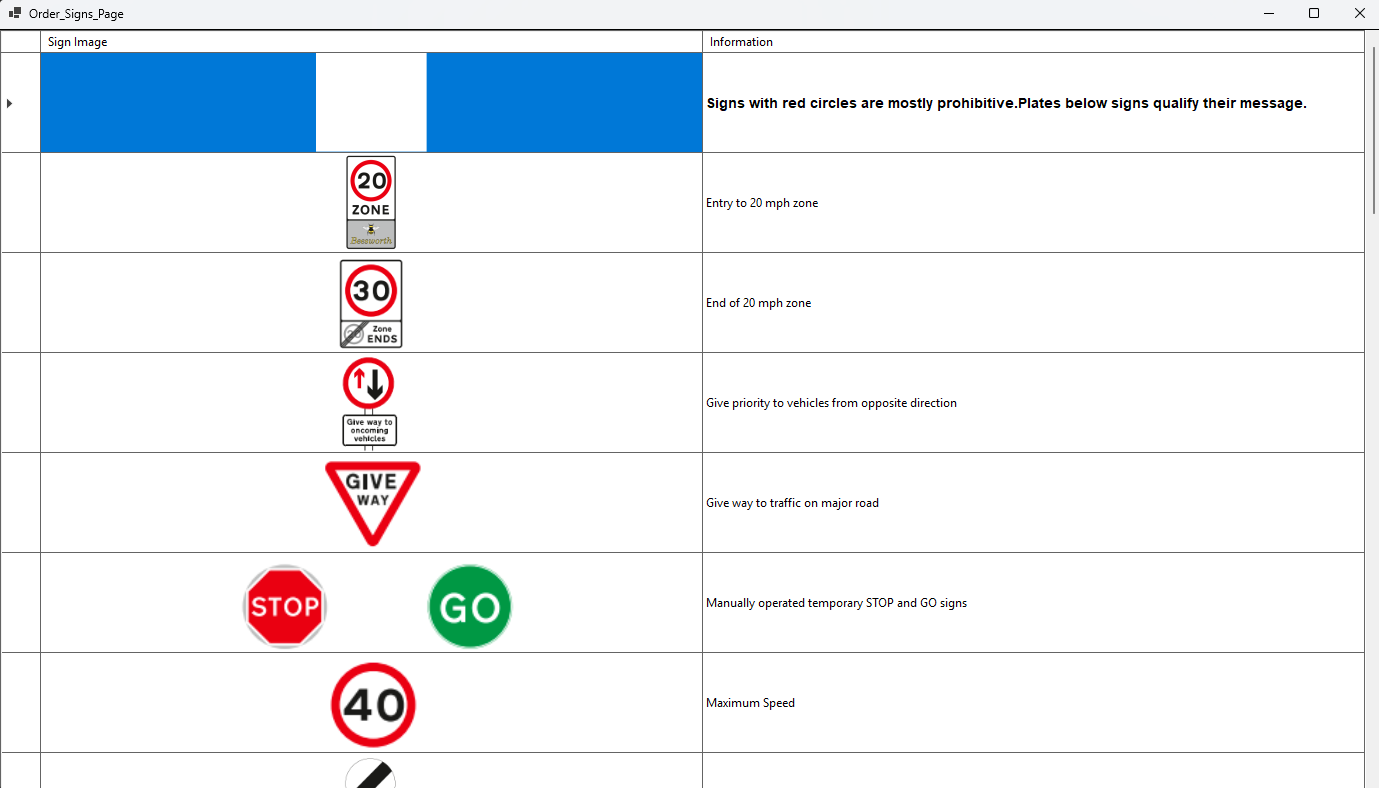
throw new System.IO.FileNotFoundException($"Image file not found: {imageFullPath}");

Image signImage = Image.FromFile(imageFullPath);

// Add row

grid.Rows.Add(signImage, info);

}

******

***Review:***

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| Does the system allow users to mark topics as completed? |  |
| Does the system persist the checkbox state even when the user navigates away from the page? |  |
| When can a users click on a particular topic, what do they see as regards all the signs connected to it? |  |
| Does every traffic sign have a clear image and a description? |  |
| Does the progress of each topic update, at the instance when the user or the system, completes or sets the topic to the incomplete status? |  |
| Is it possible for the user to flow from one page to another (topics, progress, settings, signs)? |  |
| On the page showing the progress, must all the topics that were finished be indicated with perfect icons? |  |
| Is it easy to use the software to select topics, to look for signs whereas checkboxes must be manipulative? |  |
| Are users able to remove their findings on specific or on all specific topics in case they want a fresh start? |  |
| Are there any options that allow a user to see their performance over the course of the whole program on a single page? |  |
| Are checkboxes and progress indicators (checkmarks, completion percentages) clearly visible and easy to interact with? |  |
| Can users dynamically change the background color using a ColorDialog in the settings page? |  |
| Can users adjust font size using a slider, and toggle font styles (bold/italic) using checkboxes? |  |
| Are all user settings (background color, font size, font styles) applied globally and persisted between sessions? |  |
| Does the settings page provide real-time updates with previews for font and style changes? |  |
| Are images loaded efficiently without causing performance issues, even with a large number of topics or signs? |  |
| Can users interact with images, such as zooming in for better visibility of signs? |  |
| Does the system handle missing or corrupted image files gracefully by showing placeholders? |  |
| Does the layout remain clean, with proper spacing, alignment, and legibility for readability? |  |
| Is there clear error handling for issues like missing data, image errors, or faulty inputs? |  |
| Does the system load quickly without significant delays or lag when switching between pages or topics? |  |
| Does the system visually distinguish between completed and incomplete topics? |  |
| Is the reset button functional, deleting all relevant data files (scores, flagged questions) with success confirmation? |  |
| Does the system prevent crashes or unexpected behavior from faulty inputs or user actions? |  |
| Can flagged questions be viewed separately for easy access? |  |
| Are all forms and pages centered on the screen when opened? |  |
| Does the system handle high data volumes (e.g., many topics or signs) without performance issues? |  |
| Can additional content, such as multimedia (videos, animations), be added without major redesigns? |  |
| Is the system flexible enough to integrate new topics, signs, or complex features (e.g., quizzes, assessments)? |  |
| Does the overall user interface and experience align with personalization, usability, and consistency goals? |  |

**New Update – Permanent Progress Storage:**  
This update implements a persistent state-saving mechanism that allows users to track their progress, scores, and flagged questions. The progress is stored in local text files, and this information is loaded when required, such as when the Progress Page is opened. This approach enables continuity in user experience, even after closing the application.

To save the history of user for previous test scores, completed topics and flagged questions to a .txt file which is then read by the progress page. ***For Practice Page:***

private *void* *ShowScore*()

{

if (selectedTest != null)

{

*int* testIndex = tests.*IndexOf*(selectedTest) + 1;

*GlobalData*.*PracticeScores*[testIndex] = score;

}

string filePath = "PracticeScores.txt";

*List*<string> fileLines = *File*.*Exists*(filePath) ? new *List*<string>(*File*.*ReadAllLines*(filePath)) : new *List*<string>();

string testIdentifier = $"Test: {testNameLabel.Text}";

string updatedScoreText = $"{testIdentifier}, Score: {score}/{selectedTest?.Questions.Count}";

*int* existingIndex = fileLines.*FindIndex*(line => line.*StartsWith*(testIdentifier));

if (existingIndex >= 0) fileLines[existingIndex] = updatedScoreText;

else fileLines.*Add*(updatedScoreText);

*File*.*WriteAllLines*(filePath, fileLines);

}

A screenshot of a computer

Description automatically generated

***For Traffic Signs Page:***

private void SaveData()

{

try

{

*var* uniqueTopics = new HashSet<string>(CompletedTopics.Select(topic => topic.Trim()));

using (StreamWriter writer = new StreamWriter("data.txt"))

{

writer.WriteLine(string.Join(",", uniqueTopics));

foreach (*var* entry in CheckboxStates) writer.WriteLine($"{entry.Key}:{entry.Value}");

}

}

catch (Exception ex) { Console.WriteLine("Error saving data: " + ex.Message); }

}

else File.WriteAllText(filePath, scoreLine);

A screenshot of a computer

Description automatically generated

**For Mock Test Page (*flagged questions):***

private void Flag\_Click(object? sender, EventArgs e)

{

string filePath = "Flagged\_Questions.txt";

Question currentQuestion = selectedTest[currentQuestionIndex];

string flaggedQuestionEntry = $"Test {testNumber}, Question {currentQuestionIndex + 1}:{Environment.NewLine}" +

$"Question: {currentQuestion.Text}{Environment.NewLine}" +

$"Answer: {currentQuestion.Options[currentQuestion.CorrectOptionIndex]}{Environment.NewLine}";

if (flaggedQuestions.Contains(currentQuestionIndex))

{

flaggedQuestions.Remove(currentQuestionIndex);

if (File.Exists(filePath))

{

*var* lines = File.ReadAllLines(filePath).ToList();

int startIndex = lines.FindIndex(line => line.StartsWith($"Test {testNumber}, Question {currentQuestionIndex + 1}:"));

if (startIndex != -1) lines.RemoveRange(startIndex, 3);

File.WriteAllLines(filePath, lines);

}

}

else

{

flaggedQuestions.Add(currentQuestionIndex);

if (!File.Exists(filePath)) File.WriteAllText(filePath, flaggedQuestionEntry + Environment.NewLine);

else if (!File.ReadAllLines(filePath).Contains($"Test {testNumber}, Question {currentQuestionIndex + 1}:"))

File.AppendAllText(filePath, flaggedQuestionEntry + Environment.NewLine);

}

}

**A screenshot of a computer

Description automatically generated**

**For Mock Test Page Scores:**

private void SaveScoreToFile(int testNumber, int score)

{

string filePath = "Mock\_Score.txt";

string scoreLine = $"Test: Test {testNumber}, Score: {score}/{selectedTest.Count}";

if (File.Exists(filePath))

{

*var* lines = File.ReadAllLines(filePath).ToList();

bool testFound = false;

for (int i = 0; i < lines.Count; i++)

{

if (lines[i].StartsWith($"Test: Test {testNumber}"))

{

lines[i] = scoreLine;

testFound = true;

break;

}

}

if (!testFound) lines.Add(scoreLine);

File.WriteAllLines(filePath, lines);

}

}

A screenshot of a computer

Description automatically generated

**For Progress Page (saving completed topics progress):**

private void UpdateTopicsCompleted()

{

string filePath = "data.txt";

if (!File.Exists(filePath)) return;

*var* completedTopics = new List<string>();

try

{

*var* lines = File.ReadAllLines(filePath);

foreach (*var* line in lines)

{

*var* parts = line.Split(':');

if (parts.Length == 2 && bool.TryParse(parts[1], out bool isCompleted) && isCompleted)

completedTopics.Add(parts[0].Trim());

}

}

catch (Exception ex) { Debug.WriteLine($"Error reading or parsing file {filePath}: {ex.Message}"); }

}

// Progress Page: Load Mock Scores

private Dictionary<int, int> LoadMockScores(string filePath)

{

*var* scores = new Dictionary<int, int>();

if (!File.Exists(filePath)) return scores;

*var* lines = File.ReadAllLines(filePath);

foreach (*var* line in lines)

{

try

{

*var* parts = line.Split(new[] { "Test: ", ", Score: " }, StringSplitOptions.RemoveEmptyEntries);

if (parts.Length == 2

&& int.TryParse(parts[0].Replace("Test ", string.Empty), out int testNumber)

&& int.TryParse(parts[1].Split('/')[0], out int score))

{

scores[testNumber] = score;

}

}

catch { }

}

return scores;

}

// Progress Page: Load Practice Scores

private Dictionary<int, int> LoadPracticeScores(string filePath)

{

*var* scores = new Dictionary<int, int>();

if (!File.Exists(filePath)) return scores;

*var* lines = File.ReadAllLines(filePath);

foreach (*var* line in lines)

{

try

{

*var* parts = line.Split(new[] { "Test: ", ", Score: " }, StringSplitOptions.RemoveEmptyEntries);

if (parts.Length == 2

&& int.TryParse(parts[0].Replace("Test ", string.Empty), out int testNumber)

&& int.TryParse(parts[1].Split('/')[0], out int score))

{

scores[testNumber] = score;

}

}

catch { }

}

return scores;

}

// Flagged Questions Page: Display Flagged Questions

private void DisplayFlaggedQuestions()

{

string filePath = "Flagged\_Questions.txt";

if (!File.Exists(filePath)) return;

try

{

*var* lines = File.ReadAllLines(filePath);

string currentTest = string.Empty;

for (int i = 0; i < lines.Length; i++)

{

string line = lines[i].Trim();

if (line.StartsWith("Test") && line.Contains(", Question"))

{

currentTest = line; // Test identifier

}

else if (line.StartsWith("Question:"))

{

string questionText = line.Substring("Question:".Length).Trim();

string answerText = lines[++i].Substring("Answer:".Length).Trim();

}

}

}

catch (Exception ex) { Debug.WriteLine($"Error reading flagged questions: {ex.Message}"); }

}

***Review:***

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| Does the system allow users to mark topics as completed? |  |
| Does the system persist the checkbox state even when the user navigates away from the page? |  |
| When can a users click on a particular topic, what do they see as regards all the signs connected to it? |  |
| Does every traffic sign have a clear image and a description? |  |
| Does the progress of each topic update, at the instance when the user or the system, completes or sets the topic to the incomplete status? |  |
| Is it possible for the user to flow from one page to another (topics, progress, settings, signs)? |  |
| On the page showing the progress, must all the topics that were finished be indicated with perfect icons? |  |
| Is it easy to use the software to select topics, to look for signs whereas checkboxes must be manipulative? |  |
| Are users able to remove their findings on specific or on all specific topics in case they want a fresh start? |  |
| Are there any options that allow a user to see their performance over the course of the whole program on a single page? |  |
| Are checkboxes and progress indicators (checkmarks, completion percentages) clearly visible and easy to interact with? |  |
| Can users dynamically change the background color using a ColorDialog in the settings page? |  |
| Can users adjust font size using a slider, and toggle font styles (bold/italic) using checkboxes? |  |
| Are all user settings (background color, font size, font styles) applied globally and persisted between sessions? |  |
| Does the settings page provide real-time updates with previews for font and style changes? |  |
| Are images loaded efficiently without causing performance issues, even with a large number of topics or signs? |  |
| Can users interact with images, such as zooming in for better visibility of signs? |  |
| Does the system handle missing or corrupted image files gracefully by showing placeholders? |  |
| Does the layout remain clean, with proper spacing, alignment, and legibility for readability? |  |
| Is there clear error handling for issues like missing data, image errors, or faulty inputs? |  |
| Does the system load quickly without significant delays or lag when switching between pages or topics? |  |
| Does the system visually distinguish between completed and incomplete topics? |  |
| Is the reset button functional, deleting all relevant data files (scores, flagged questions) with success confirmation? |  |
| Does the system prevent crashes or unexpected behavior from faulty inputs or user actions? |  |
| Can flagged questions be viewed separately for easy access? |  |
| Are all forms and pages centered on the screen when opened? |  |
| Does the system handle high data volumes (e.g., many topics or signs) without performance issues? |  |
| Can additional content, such as multimedia (videos, animations), be added without major redesigns? |  |
| Is the system flexible enough to integrate new topics, signs, or complex features (e.g., quizzes, assessments)? |  |
| Does the overall user interface and experience align with personalization, usability, and consistency goals? |  |

**Prototype 7 – Settings Page:**

**Design**

**Overview:**

In this section, I will plan the Settings Page workflow, allowing users to adjust the background color, font size, and font style. The settings page will include controls such as a ColorDialog for changing the background color, a slider for adjusting the font size, and checkboxes for selecting font styles (bold/italic). I will also ensure smooth transitions between forms, with updates being applied globally across all forms.

**Decomposition to computable sections:**

|  |  |
| --- | --- |
| **Section** | **Justification (suitable for computation because…)** |
| Mapping ColorDialog to Change Background Color | The background color will be changed based on user selection. A ColorDialog is suitable for allowing users to select a color dynamically. |
| Implementing Font Size and Style Adjustments | Tracks changes to checkboxes, updates the state in a persistent storage, and maintains completed topics. |
| Centering Forms on Screen | The font size will be adjusted with a slider, and the font style (bold/italic) will be toggled with checkboxes. This ensures a dynamic user interface. |
| Resetting Data | The reset functionality needs to delete specific files to clear mock test scores and completed topics, ensuring data management is handled efficiently. |
| Finalized Settings Workflow | The finalized settings workflow will ensure smooth application of user preferences like background color, font size, and style. The reset button will handle data deletion and inform the user of the results. |

**States:**

**A screenshot of a computer program

Description automatically generated**

Encapsulation

* This class wraps settings with the help of static fields (GlobalBackgroundColor, etc.).
* Event handlers hides executable code specific to the view.

Inheritance

* Derived from Form class, thereby acquiring related graphical user interface abilities.

Polymorphism

* Preference of runtime polymorphism can be done in methods such as Change\_Color\_Click.

Abstraction

* Programmatically encapsulates font and color management of abstracts through a user friendly interface.

Summary

* Encapsulation leads to the achievement of modularity.
* Retains features of UI in order to use abstraction for better usability.

***Step 1: Mapping ColorDialog to Change Background Color***

To allow users to change the background color, I will implement a button that opens a ColorDialog. When the user selects a color, the program will update a global variable for the background color and notify the user about the update.

**Pseudocode:**

Procedure Change\_Color\_Click

Open a new ColorDialog

If the user selects a color in the ColorDialog

Set the global variable related to bg colour to the user’s selected color

Display a notification box (message box) to let the user know the bg color has been changed

End Procedure

**Reason:** This step allows users to make visual changes to an application by setting a new background color. Since this will be implemented as a global change, all the forms will then reflect that updated background on reopening - hopefully providing consistency within the application.

**Approach:**

* Add a button labeled "Change Background Color" to my Settings Page.
* When this button will be clicked, the button will then open the ColorDialog.
* User will select a color and this will be stored in a variable.
* A prompt will appear that informs the user that this change has taken effect.

***Step 2: Implementing Font Size and Style Adjustments***

Users will be able to adjust the font size using a slider, and choose bold or italic styles using checkboxes. The global font settings will be updated dynamically as the user interacts with the controls.

**Pseudocode:**

*Pseudocode for Font Size:*

Procedure FontSizeSlider\_Scroll:

Update the global font size based on the slider value

FontSize = Value(fontSizeSlider)

Update the font size of the preview label

End Procedure

*Pseudocode for Font Style (Bold/Italic):*

PROCEDURE FontStyle\_Change

fontStyle = Regular(FontStyle)

IF Checked(boldCheckBox)

fontStyle = Bold(FontStyle)

END IF

IF Checked(italicCheckBox)

fontStyle = Italic(FontStyle)

END IF

Update the font style globally for all pages

Update the font of the preview label in settings view

Display a message about the font style update (applies to all pages now for UI experience)

END PROCEDURE

**Reason:** The slider allows the user to easily change font size, while the checkboxes give them control over the font style. This ensures a personalized user interface. The preview label immediately updates so the user can see the changes in real-time.

Approach:

* A slider will allow users to adjust the font size.
* Two checkboxes will control whether the font is bold or italic.
* The font size and style will be updated dynamically, with a preview label showing the updated font.
* A message box will confirm the changes when the style is updated.

***Step 3: Centering Forms on Screen***

When any form is opened, I know that Windows Forms won’t automatically center this text. Therefore as I need it to be centered on the screen to enhance the user experience I will manually implement this code into every page.

**Code:**

this.StartPosition = FormStartPosition.CenterScreen;

**Reason:** Centering forms ensures a consistent and professional look across different screen sizes and resolutions. It also ensures that the form will be easily visible to the user when it is opened.

**Approach:**

* I will ensure that every form has this line of code to center it on the screen when it is loaded.

***Step 4: Resetting Data***

The reset functionality will allow users to delete test scores and completed topics by removing the relevant files. If any of the specified files exist, they will be deleted. The user will be informed of the outcome.

**Pseudocode:**

PROCEDURE reset\_Click

filesToDelete = []

SET filesToDelete = [

"PracticeScores.txt",

"Mock\_Score.txt",

"Flagged\_Questions.txt",

"data.txt"

]

filesDeleted = FALSE

filesExist = FALSE

FOR loop file IN filesToDelete

IF FileExists(file)

filesExist = TRUE

DeleteFile(file)

filesDeleted = TRUE

END IF

END FOR

IF filesDeleted = TRUE

OUTPUT "Data has been deleted."

ELSE

OUTPUT "No data to reset."

END IF

END PROCEDURE

**Reason:** This feature allows users to reset their data, which can be useful if they want to start fresh or clear unnecessary data. The user will receive feedback on whether the operation was successful.

**Approach:**

* The reset button will trigger the deletion of the specified files.
* If any files are deleted, a confirmation message will appear. If no files are found, a warning message will notify the user.

**Final Workflow Summary**

1. User Action: The program listens for button clicks or slider movements.
2. Settings Management: Upon interaction, the settings (background color, font size, font style) will be updated in real-time.
3. Form Behaviour: The forms in the program will open centered on the screen for a consistent UX as needed for a smooth consistent learning experience for my users without ‘clutter’.
4. Resetting Data: The reset button will clear all specified files.

**Development**

***Overview:***

The Settings Page is a centralized form designed to provide users with control over customization options for the application. It allows for changes to visual elements (background color, font size, and style) and allows users to reset stored progress.

***Development and Debugging (broken down into chronological iterations/updates):***

**Iteration 1:**

Setting button to change the background colour of each form for user visibility  
Creating a new Form Page of Settings:

private void Change\_Color\_Click(object sender, EventArgs e)

{

using (ColorDialog colorDialog = new ColorDialog())

{

if (colorDialog.ShowDialog() == DialogResult.OK)

{

GlobalBackgroundColor = colorDialog.Color; // Set the global background color

MessageBox.Show("Background color updated. It will apply to all forms when they are reopened.",

"Settings",

MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

}

}

**A yellow square with a white text

Description automatically generatedA white background with red text

Description automatically generatedA screenshot of a computer screen

Description automatically generated**

**Problem:** How can the user change the size of the font? **Solution:** Introduce a slider on the Settings Page So That The User can change the size of the font as well as choose if they want to make it bold/italic

private void FontSizeSlider\_Scroll(object sender, EventArgs e)

{

// Update the global font size based on the slider value

GlobalFontSize = fontSizes[fontSizeSlider.Value];

// Update the font size of the preview label

fontPreviewLabel.Font = new Font("Arial", GlobalFontSize, GlobalFontStyle);

}

private void FontStyleCheckBox\_CheckedChanged(object sender, EventArgs e)

{

FontStyle fontStyle = FontStyle.Regular;

if (boldCheckBox.Checked)

fontStyle |= FontStyle.Bold;

if (italicCheckBox.Checked)

fontStyle |= FontStyle.Italic;

GlobalFontStyle = fontStyle;

// Update the font style of the preview label

fontPreviewLabel.Font = new Font("Arial", GlobalFontSize, GlobalFontStyle);

MessageBox.Show($"Font style updated to {GlobalFontStyle}.", "Settings",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

**Problem:** Forms Opening anywhere on the screen  
**Solution**: Make The Forms on center when they are being loaded in runtime.

this.StartPosition = FormStartPosition.CenterScreen;

A screenshot of a computer

Description automatically generated

*A screenshot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generated*  
  
**Iteration 2:**

Adding a Reset Button which Deletes the mock test practice test scores as well as topics completed too.

private void reset\_Click(object sender, EventArgs e)

{

// File paths (update these with the correct file paths as needed)

string[] filesToDelete = {

"PracticeScores.txt",

"Mock\_Score.txt",

"Flagged\_Questions.txt",

"data.txt"

};

bool filesDeleted = false; // Track if any files were deleted

bool filesExist = false; // Track if any files existed

foreach (string file in filesToDelete)

{

try

{

if (System.IO.File.Exists(file))

{

filesExist = true;

System.IO.File.Delete(file);

filesDeleted = true; // Mark that at least one file was deleted

}

}

catch (Exception ex)

{

MessageBox.Show($"An error occurred while deleting '{file}': {ex.Message}", "Error",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return; // Exit the method on error

}

}

// Show appropriate message after processing all files

if (filesDeleted)

{

MessageBox.Show("Data has been deleted.", "Reset",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else if (!filesExist)

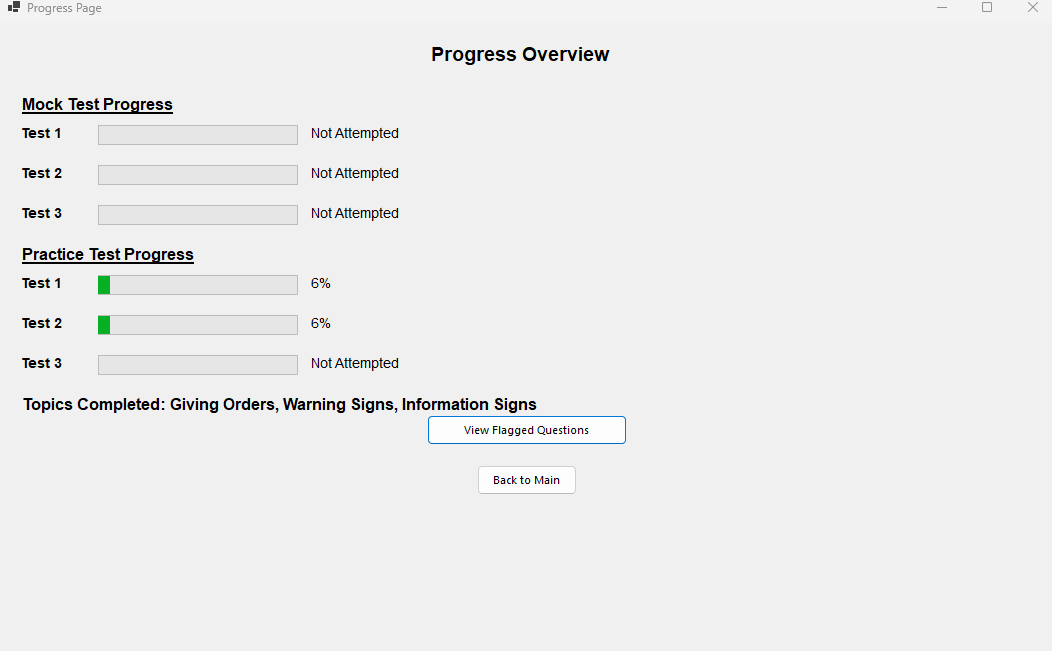
{

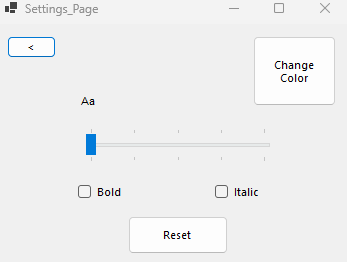
MessageBox.Show("No data to reset.", "Reset",

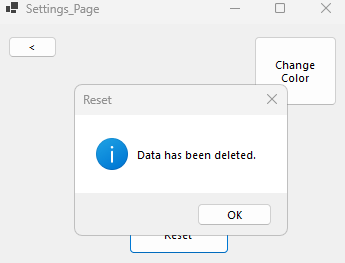
MessageBoxButtons.OK, MessageBoxIcon.Warning);

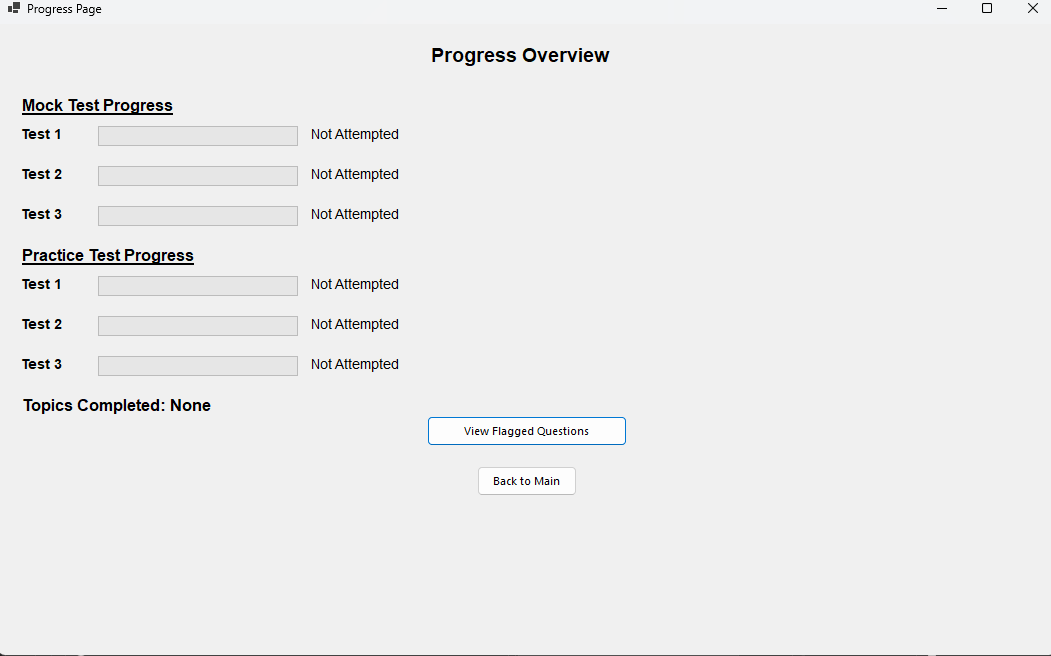
}

}

******

******

******

******

***Review:***

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| Does the system allow users to mark topics as completed? |  |
| Does the system persist the checkbox state even when the user navigates away from the page? |  |
| When can a users click on a particular topic, what do they see as regards all the signs connected to it? |  |
| Does every traffic sign have a clear image and a description? |  |
| Does the progress of each topic update, at the instance when the user or the system, completes or sets the topic to the incomplete status? |  |
| Is it possible for the user to flow from one page to another (topics, progress, settings, signs)? |  |
| On the page showing the progress, must all the topics that were finished be indicated with perfect icons? |  |
| Is it easy to use the software to select topics, to look for signs whereas checkboxes must be manipulative? |  |
| Are users able to remove their findings on specific or on all specific topics in case they want a fresh start? |  |
| Are there any options that allow a user to see their performance over the course of the whole program on a single page? |  |
| Are checkboxes and progress indicators (checkmarks, completion percentages) clearly visible and easy to interact with? |  |
| Can users dynamically change the background color using a ColorDialog in the settings page? |  |
| Can users adjust font size using a slider, and toggle font styles (bold/italic) using checkboxes? |  |
| Are all user settings (background color, font size, font styles) applied globally and persisted between sessions? |  |
| Does the settings page provide real-time updates with previews for font and style changes? |  |
| Are images loaded efficiently without causing performance issues, even with a large number of topics or signs? |  |
| Can users interact with images, such as zooming in for better visibility of signs? |  |
| Does the system handle missing or corrupted image files gracefully by showing placeholders? |  |
| Does the layout remain clean, with proper spacing, alignment, and legibility for readability? |  |
| Is there clear error handling for issues like missing data, image errors, or faulty inputs? |  |
| Does the system load quickly without significant delays or lag when switching between pages or topics? |  |
| Does the system visually distinguish between completed and incomplete topics? |  |
| Is the reset button functional, deleting all relevant data files (scores, flagged questions) with success confirmation? |  |
| Does the system prevent crashes or unexpected behavior from faulty inputs or user actions? |  |
| Can flagged questions be viewed separately for easy access? |  |
| Are all forms and pages centered on the screen when opened? |  |
| Does the system handle high data volumes (e.g., many topics or signs) without performance issues? |  |
| Can additional content, such as multimedia (videos, animations), be added without major redesigns? |  |
| Is the system flexible enough to integrate new topics, signs, or complex features (e.g., quizzes, assessments)? |  |
| Does the overall user interface and experience align with personalization, usability, and consistency goals? |  |

**Prototype 8 - Hazard Perception Test Page**

**Design**

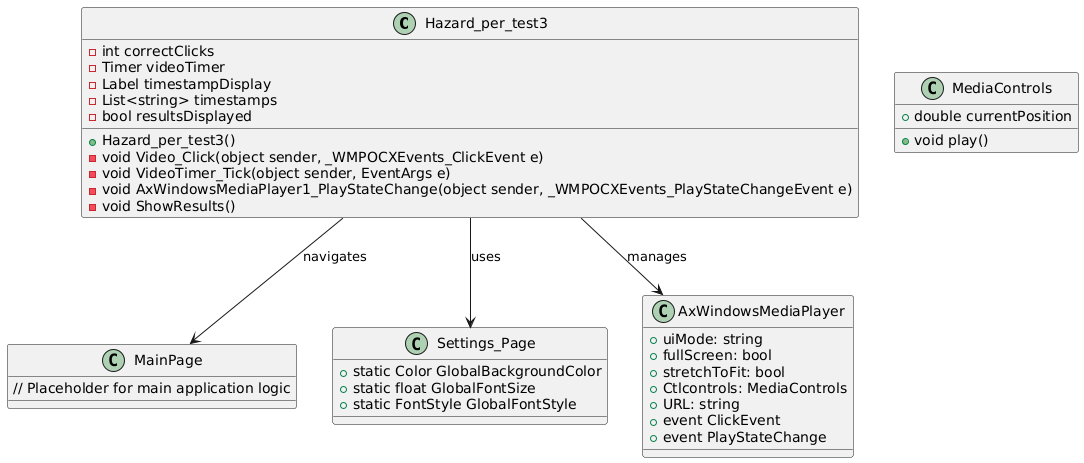
**Overview:**

In this section, I will plan the Hazard Perception Test feature for the application, allowing users to interact with a video. These videos will have have hazards which will need to be flagged similar to the real theory exam. As a result the user will need to flag specific timestamps and receive points for these correct flags.

**Decomposition to computable sections:**

|  |  |
| --- | --- |
| **Section** | **Justification (suitable for computation because…)** |
| Map the Video Playback to User Interaction | The video will play and pause based on user interaction, providing a base for flagging timestamps that can be identified as the hazards in the video. |
| Flagging Timestamps on Video | The user will be able to click specific points in the video, which will need real-time detection of the timestamp. |
| Awarding Points for Correct Flags | Points will be awarded for correctly flagged timestamps within a specified time range. |
| Displaying Real-Time Timestamp | The current video timestamp will be displayed dynamically during playback. |

**States:**

****

Encapsulation

* Casts test logic and other variables such as timestamps and correctClicks.
* Video\_Click is an event handler that performs a task on mouse click and VideoTimer\_Tick event handler accomplishes another.

Inheritance

* Subclasses from Form in a way that makes the management of Multiple Document Interfaces less redundant.

Polymorphism

* Encapsulates polymorphism in such events as PlayStateChange and ClickEvent.

Abstraction

* Of the structure of scoring and playback of the abstracts from the users.

Summary

* High use of internals encapsulation and implementation of inheritance.
* Uses the concept of abstraction for usability, and polymorphism for flexibility of operations.

***Step 1: Mapping Video Playback to User Interaction To allow users to interact with a video.***

I will use a Windows Media Player control to load and play the video. The video will be played as soon as the form loads, and the timestamp will be updated every second.

**Pseudocode:**

PROCEDURE Hazard\_perception\_test\_Load

SET URL(axWindowsMediaPlayer1) TO "Path of Video"

SET Interval(timer) TO 500

Start()(timer)

END PROCEDURE

PROCEDURE timer\_Tick

currentMedia(axWindowsMediaPlayer1)  
 SET currentTime TO VideoTime  
 SET Text(label1) TO currentTime

END IF  
END PROCEDURE

**Reason:** This step will make sure that the video starts playing as soon as the user enters the test, and the current timestamp is displayed for reference when a flag is placed.

**Approach:**

* Implement a Windows Media Player control.
* Start the video when the form loads.
* Update the timestamp on every tick of a timer.

***Step 2: Flagging Timestamps on Video Users will be able to flag specific timestamps by clicking on the form during playback. When a user clicks, the current timestamp of the video will be stored.***

**Pseudocode:**

PROCEDURE Form\_Click

currentTime = currentPosition(Ctlcontrols(axWindowsMediaPlayer1))

timestamp = FromSeconds(currentTime)(TimeSpan)

OUTPUT “Flagged Timestamp:”, timestamp

END PROCEDURE

**Reason:** This functionality allows the user to mark specific moments in the video making it interactive. The timestamp will then therefore be flagged when the user clicks the form.

**Approach:**

* Put in a click event handler for the form.
* Retrieve the current timestamp from the video and display this in a label format to the user.

***Step 3: Awarding Points for Correct Flags Users will be awarded points if they flag a timestamp within a specified time range (e.g., within the first 5 seconds of a relevant event in the video).***

**Pseudocode:**

Procedure AwardPoints (timestamp):

If timestamp >= targetStartTime And timestamp <= targetEndTime Then

Points = points +1

OUTPUT ”Correct flag! Points awarded."

Else

OUTPUT "Incorrect flag. No points awarded”  
 ENDIF

End Procedure

**Reason:** This can allow me to reward users for correctly identifying the hazards in the video as required in the real test.

**Approach:**

* Create a specific time window for valid flags to be raised by user input ( for example 5 seconds after an event).
* Check if the flagged timestamp falls within the valid time window range created.
* Award points accordingly to the time the flag is placed and display a message confirming this result for progress tracking.

***Step 4: Displaying Real-Time Timestamp During playback, the current timestamp will be continuously displayed on the form to give the user a reference point for flagging timestamps.***

**Pseudocode:**

PROCEDURE timer1\_Tick

currentTime TO currentPosition((axWindowsMediaPlayer1))

OUTPUT currentTime

END IF

END PROCEDURE

**Reason:** This will make sure that the user can see the current timestamp helping them decide when to flag a hazard in the video being played.

**Approach:**

* Use a timer to update the timestamp label every second.
* Display the timestamp in a label so users can refer to this easily.

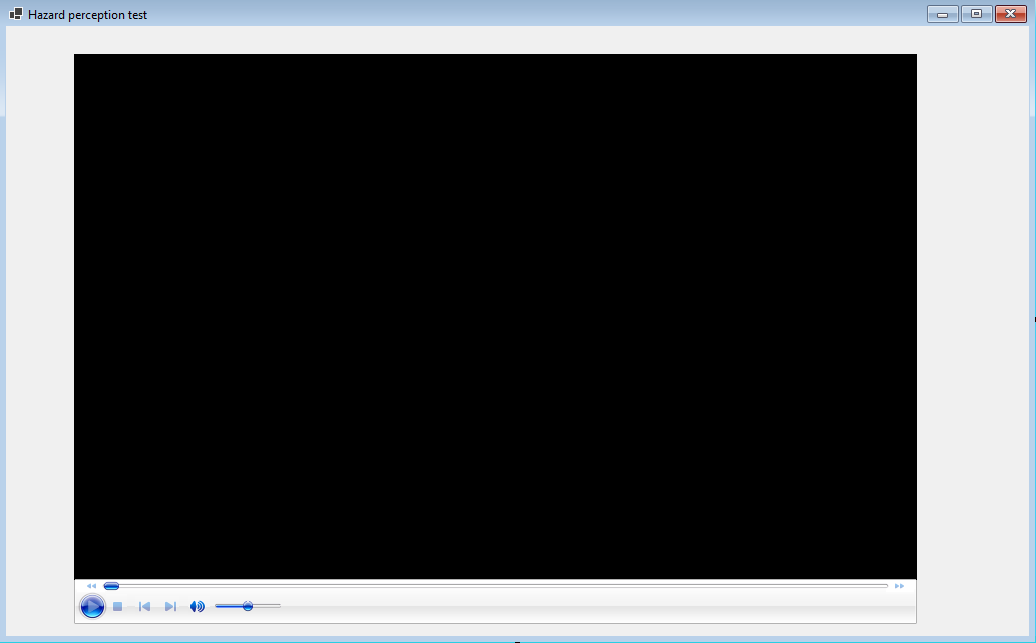
**Final Workflow Summary:**

1. User Action: The user can click on the Form when a video is being played in order to mark/flag a hazard to the program.
2. Timestamp Flagging: This timestamp will be captured and displayed to the user.
3. Points Awarding: Points will only be awarded if the user flags the timestamp within the designated time range.
4. Real-Time Timestamp Display: The timestamp is updated continuously as the video plays.

**Development**

***Overview:***

The Hazard Perception Test Page is an interactive application designed to test users' ability to detect and respond to hazards in a video-based environment. The user interacts with a video by flagging potential hazards at specific timestamps, earning points based on the timing and accuracy of their responses. The page has undergone several development iterations to improve functionality, user experience, and scoring logic.

***Development and Debugging (broken down into chronological iterations/updates):***  
**Iteration 1:**A Hazard Perception Video which plays a video where the user have to mark/flag where the upcoming obstacle is coming from  
**Problem:** How do add a video on the user form?  
**Solution:** Add Windows Media player from the tool box.  
  
  
  
Adding a Label for the Timestamp & adding a timer:

private void Hazard\_perception\_test\_Load(object sender, EventArgs e)

{

axWindowsMediaPlayer1.URL = "C:\\path\\to\\your\\video.mp4";

axWindowsMediaPlayer1.Ctlcontrols.play();

timer1.Interval = 500; timer1.Start();

}

private void timer1\_Tick(object sender, EventArgs e)

{

Update the label with the current timestamp if (axWindowsMediaPlayer1.currentMedia != null) { double currentTime = axWindowsMediaPlayer1.Ctlcontrols.currentPosition; label1.Text = TimeSpan.FromSeconds(currentTime).ToString(@"hh\:mm\:ss"); }

}

**Iteration 2:Problem:** Remove the ability for the user to interact with video ui and media control  
**Solution:** I can configure the axWindowsMediaPlayer control to hide the UI elements for playback control.

using System;

using System.Windows.Forms;

*namespace* WinFormsApp1

{

public partial *class* Hazard\_perception\_test : Form

{

public Hazard\_perception\_test()

{

InitializeComponent();

// Configure axWindowsMediaPlayer on form load

axWindowsMediaPlayer1.uiMode = "none"; // Hide user controls

}

private System.Windows.Forms.Timer timer1;

private System.Windows.Forms.Label label1;

private void axWindowsMediaPlayer1\_Enter(object sender, EventArgs e)

{

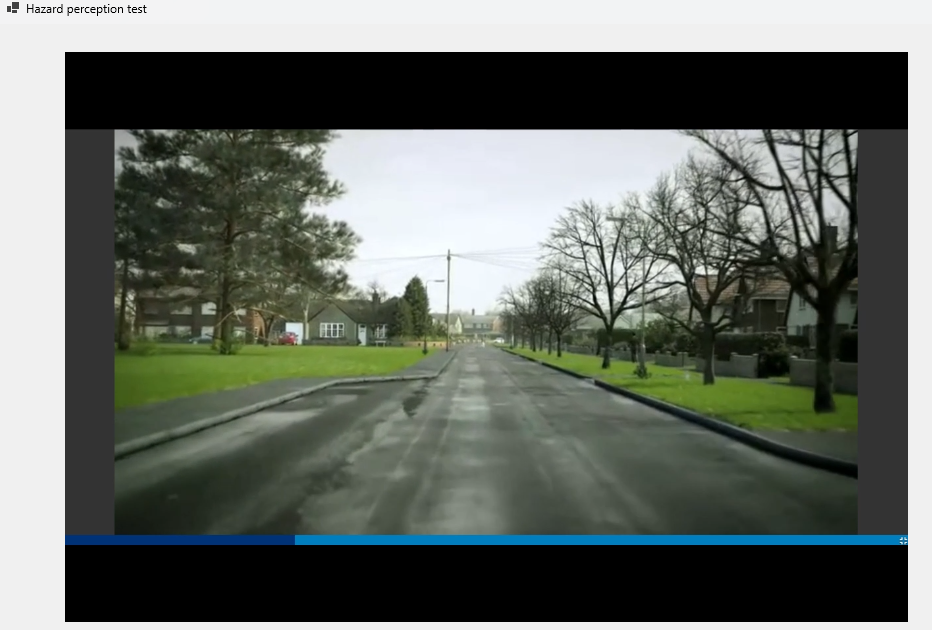
axWindowsMediaPlayer1.URL = "Video\_1.mp4";// Set the path to your video

axWindowsMediaPlayer1.Ctlcontrols.play(); // Play video automatically

}

}

}

******

**Iteration 3:Problem:** How to mark flags on the time stamp?   
**Solution:** Whenever a user clicks anywhere on the form make it so it flags that specific time stamp of the video and shows it to the user

using System;

using System.Windows.Forms;

*namespace* WinFormsApp1

{

public partial *class* Hazard\_perception\_test : Form

{

public Hazard\_perception\_test()

{

InitializeComponent();

// Configure axWindowsMediaPlayer on form load

axWindowsMediaPlayer1.uiMode = "none"; // Hide user controls

// Configure form's click event to flag timestamps

this.Click += Form\_Click;

}

private void axWindowsMediaPlayer1\_Enter(object sender, EventArgs e)

{

axWindowsMediaPlayer1.URL = "Video\_1.mp4";// Set the path to your video

axWindowsMediaPlayer1.Ctlcontrols.play(); // Play video automatically

}

private void Form\_Click(object sender, EventArgs e)

{

// Get the current position of the video

double currentTime = axWindowsMediaPlayer1.Ctlcontrols.currentPosition;

// Format the timestamp to minutes and seconds

TimeSpan timestamp = TimeSpan.FromSeconds(currentTime);

string formattedTime = timestamp.ToString(@"mm\:ss");

// Display the flagged time

MessageBox.Show($"Flagged Timestamp: {formattedTime}", "Flagged Time", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

**Problem:** How to award points?  
**Solution:** User earns points based on their clicks within the first 5 seconds of the video for now , and display a prompt with the final points when the video ends

using System;

using System.Windows.Forms;

*namespace* WinFormsApp1

{

public partial *class* Hazard\_perception\_test : Form

{

private int correctClicks;

private System.Windows.Forms.Timer videoTimer;

private Label timestampDisplay;

private System.Collections.Generic.List<string> timestamps;

public Hazard\_perception\_test()

{

InitializeComponent();

// Configure axWindowsMediaPlayer on form load

axWindowsMediaPlayer1.uiMode = "none"; // Hide user controls

// Initialize the timer

videoTimer = new System.Windows.Forms.Timer();

videoTimer.Interval = 1000; // Check every second

videoTimer.Tick += VideoTimer\_Tick;

// Configure form's click event to flag timestamps

this.Click += Form\_Click;

// Initialize the counter for correct clicks

correctClicks = 0;

// Initialize the label to display timestamps

timestampDisplay = new Label();

timestampDisplay.Location = new System.Drawing.Point(943, 28); // Adjust location as needed

timestampDisplay.Size = new System.Drawing.Size(200, 300); // Size of the display area

timestampDisplay.BorderStyle = BorderStyle.FixedSingle;

this.Controls.Add(timestampDisplay);

// List to store timestamps

timestamps = new System.Collections.Generic.List<string>();

}

private void axWindowsMediaPlayer1\_Enter(object sender, EventArgs e)

{

axWindowsMediaPlayer1.URL = "Video\_1.mp4";// Set the path to your video

axWindowsMediaPlayer1.Ctlcontrols.play(); // Play video automatically

// Start the timer to monitor video end

videoTimer.Start();

}

private void Form\_Click(object sender, EventArgs e)

{

// Get the current position of the video

double currentTime = axWindowsMediaPlayer1.Ctlcontrols.currentPosition;

// Format the timestamp and add to the list

TimeSpan timestamp = TimeSpan.FromSeconds(currentTime);

string formattedTime = timestamp.ToString(@"mm\:ss");

timestamps.Add(formattedTime);

// Update the timestamp display

timestampDisplay.Text = string.Join("\n", timestamps);

}

private void VideoTimer\_Tick(object sender, EventArgs e)

{

// Check if the video has reached its end

if (axWindowsMediaPlayer1.Ctlcontrols.currentPosition >= axWindowsMediaPlayer1.currentMedia.duration)

{

// Video has ended, stop the timer

videoTimer.Stop();

// Gather all timestamps and categorize them as correct or incorrect

string resultText = "Flagged Timestamps:\n";

foreach (*var* timestamp in timestamps)

{

// Determine if the timestamp was flagged within the first 5 seconds of the video

double timestampInSeconds = TimeSpan.Parse(timestamp).TotalSeconds;

string correctness = timestampInSeconds <= 5 ? "Correct" : "Incorrect";

if (correctness == "Correct")

{

correctClicks++;

}

resultText += $"{correctness}: {timestamp}\n";

}

// Show the result as a message box

MessageBox.Show($"{resultText}Your total points: {correctClicks}", "Final Points", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

}

**Obstacle:** Comparison of time format is wrong   
**Solution:** Convert the hardcoded 5 seconds into a formatted timestamp string "00:05" and then compare it to the timestamps recorded during the video.

private void VideoTimer\_Tick(object sender, EventArgs e)

{

// Check if the video has reached its end

if (axWindowsMediaPlayer1.Ctlcontrols.currentPosition >= axWindowsMediaPlayer1.currentMedia.duration)

{

// Video has ended, stop the timer

videoTimer.Stop();

// Prepare the hardcoded timestamp for comparison (5 seconds)

string fiveSecondsTimestamp = "00:05";

double fiveSeconds = TimeSpan.Parse(fiveSecondsTimestamp).TotalSeconds;

// Gather all timestamps and categorize them as correct or incorrect

string resultText = "Flagged Timestamps:\n";

foreach (*var* timestamp in timestamps)

{

// Convert the timestamp into seconds

TimeSpan ts = TimeSpan.Parse(timestamp);

double timestampInSeconds = ts.TotalSeconds;

// Compare the timestamp to the "00:05" mark

string correctness = timestampInSeconds <= fiveSeconds ? "Correct" : "Incorrect";

if (correctness == "Correct")

{

correctClicks++;

}

resultText += $"{correctness}: {timestamp}\n";

}

// Show the result as a message box

MessageBox.Show($"{resultText}Your total points: {correctClicks}", "Final Points", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

**Iteration 4:Problem:** Want to show flags in front of time stamps   
**Solution:** To show a flag in front of the timestamps text to indicate flagged times, we can modify the Video\_Click method to add some visual indicator next to each timestamp in the timestampDisplay.

using AxWMPLib;

using System;

using System.Windows.Forms;

*namespace* WinFormsApp1

{

public partial *class* Hazard\_perception\_test : Form

{

private int correctClicks;

private System.Windows.Forms.Timer videoTimer;

private Label timestampDisplay;

private System.Collections.Generic.List<string> timestamps;

public Hazard\_perception\_test()

{

InitializeComponent();

// Configure axWindowsMediaPlayer on form load

axWindowsMediaPlayer1.uiMode = "none"; // Hide user controls

axWindowsMediaPlayer1.fullScreen = false;

axWindowsMediaPlayer1.stretchToFit = true;

// Initialize the timer

videoTimer = new System.Windows.Forms.Timer();

videoTimer.Interval = 1000; // Check every second

videoTimer.Tick += VideoTimer\_Tick;

// Initialize the counter for correct clicks

correctClicks = 0;

// Initialize the label to display timestamps

timestampDisplay = new Label();

timestampDisplay.Location = new System.Drawing.Point(943, 28); // Adjust location as needed

timestampDisplay.Size = new System.Drawing.Size(200, 300); // Size of the display area

timestampDisplay.BorderStyle = BorderStyle.FixedSingle;

this.Controls.Add(timestampDisplay);

// List to store timestamps

timestamps = new System.Collections.Generic.List<string>();

// Event handlers for clicks on the video player

axWindowsMediaPlayer1.ClickEvent += Video\_Click;

}

private void axWindowsMediaPlayer1\_Enter(object sender, EventArgs e)

{

axWindowsMediaPlayer1.URL = "Video\_1.mp4";// Set the path to your video

axWindowsMediaPlayer1.Ctlcontrols.play(); // Play video automatically

// Start the timer to monitor video end

videoTimer.Start();

}

private void Video\_Click(object sender, \_WMPOCXEvents\_ClickEvent e)

{

// Get the current position of the video

double currentTime = axWindowsMediaPlayer1.Ctlcontrols.currentPosition;

// Format the timestamp and add to the list with marker

TimeSpan timestamp = TimeSpan.FromSeconds(currentTime);

string formattedTime = timestamp.ToString(@"mm\:ss");

string timestampWithMarker = $"⚑ {formattedTime}";

timestamps.Add(timestampWithMarker);

// Update the timestamp display

timestampDisplay.Text = string.Join("\n", timestamps);

}

private void VideoTimer\_Tick(object sender, EventArgs e)

{

// Check if the video has reached its end

if (axWindowsMediaPlayer1.Ctlcontrols.currentPosition >= axWindowsMediaPlayer1.currentMedia.duration)

{

// Video has ended, stop the timer

videoTimer.Stop();

// Prepare the hardcoded timestamp for comparison (5 seconds)

string fiveSecondsTimestamp = "00:05";

double fiveSeconds = TimeSpan.Parse(fiveSecondsTimestamp).TotalSeconds;

// Gather all timestamps and categorize them as correct or incorrect

string resultText = "Flagged Timestamps:\n";

foreach (*var* timestamp in timestamps)

{

// Convert the timestamp into seconds

TimeSpan ts = TimeSpan.Parse(timestamp.Substring(2)); // Remove the ⚑ marker

double timestampInSeconds = ts.TotalSeconds;

// Compare the timestamp to the "00:05" mark

string correctness = timestampInSeconds <= fiveSeconds ? "Correct" : "Incorrect";

if (correctness == "Correct")

{

correctClicks++;

}

resultText += $"{correctness}: {timestamp}\n";

}

// Show the result as a message box

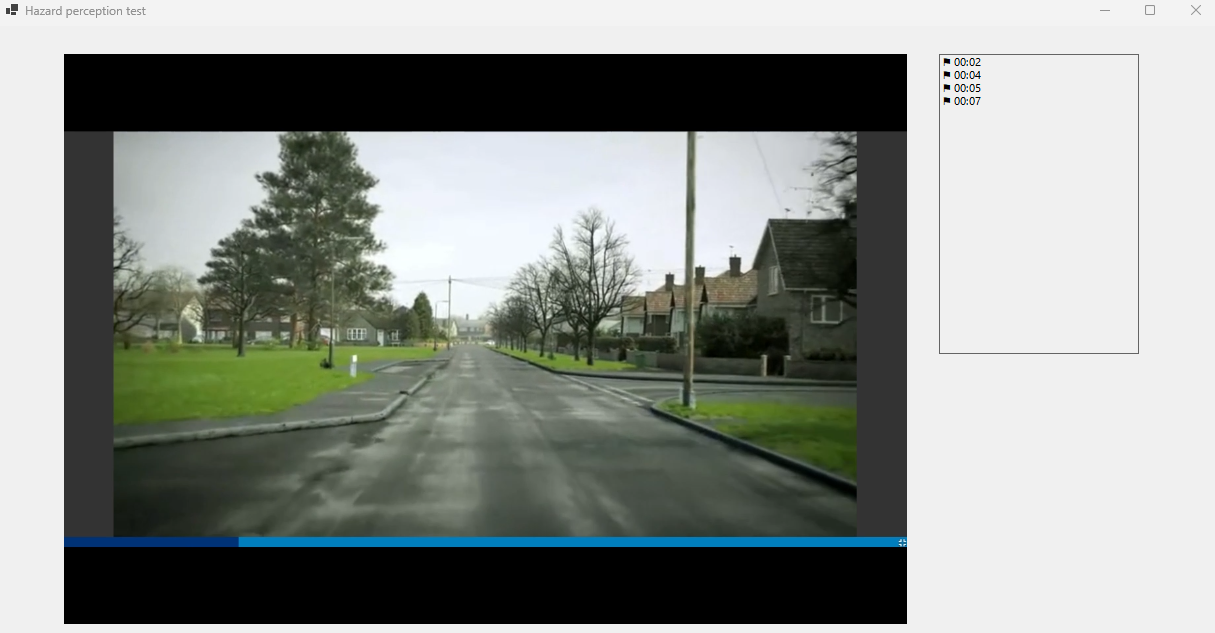
MessageBox.Show($"{resultText}Your total points: {correctClicks}", "Final Points", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

}

****Problem:** Want to change the time stamps  
**Solution:** Use if else and range to see if the click/flag lies within it

private *void* VideoTimer\_Tick(*object* *sender*, *EventArgs* *e*)

{

// Check if the video has reached its end

if (*axWindowsMediaPlayer1.Ctlcontrols*.currentPosition >= axWindowsMediaPlayer1.currentMedia.duration)

{

// Video has ended, stop the timer

*videoTimer.Stop*();

// Prepare the hardcoded timestamp for comparison (5 seconds)

*double* fiveSeconds = *TimeSpan.Parse*("00:05").*TotalSeconds*;

// Gather all timestamps and categorize them as correct or incorrect with points

string resultText = "Flagged Timestamps:\n";

*int* totalPoints = 0;

bool pointsAdded = false; // Track if any points have been added

foreach (var timestamp in timestamps)

{

// Extract time part from the formatted timestamp

string timePart = *timestamp.Substring*(2); // Remove the ⚑ marker

*TimeSpan* ts = *TimeSpan.Parse*(timePart);

*double* timestampInSeconds = *ts.TotalSeconds*;

// Compare timestamp against each specific time point for points

*int* points = 0;

if (timestampInSeconds <= *TimeSpan.Parse*("00:01").*TotalSeconds*)

{

points = 5;

}

else if (timestampInSeconds <= *TimeSpan.Parse*("00:02").*TotalSeconds*)

{

points = 4;

}

else if (timestampInSeconds <= *TimeSpan.Parse*("00:03").*TotalSeconds*)

{

points = 3;

}

else if (timestampInSeconds <= *TimeSpan.Parse*("00:04").*TotalSeconds*)

{

points = 2;

}

else if (timestampInSeconds <= *TimeSpan.Parse*("00:05").*TotalSeconds*)

{

points = 1;

}

// Only add the first set of points within each interval

if (points > 0 && !pointsAdded)

{

correctClicks += points;

pointsAdded = true; // Mark points as added

}

// Append the result to the display text

resultText += $"{(points > 0 ? "*Correct*" : "*Incorrect*")}: {timestamp} (+{points} points)\n";

}

// Show the result as a message box

*MessageBox.Show*($"{resultText}Your total points: {correctClicks}", "Final Points", *MessageBoxButtons*.OK, *MessageBoxIcon.Information*);

}

}

**Obstacle:** Time stamps Range not being implemented correctly  
**Solution:**

// Show the results in a new form with a "Next" button

Form resultsForm = new Form

{

Text = "Final Results",

Size = new System.Drawing.Size(342, 151),

StartPosition = FormStartPosition.CenterParent

};

Label resultsLabel = new Label

{

Text = $"Your total points: {correctClicks}",

Dock = DockStyle.Fill,

AutoSize = false,

TextAlign = ContentAlignment.TopLeft,

Padding = new Padding(10)

};

Button nextButton = new Button

{

Text = "Next",

Dock = DockStyle.Bottom,

Height = 40

};

nextButton.Click += (s, e) =>

{

resultsForm.Close();

// Open the next form with the new video

Hazard\_per\_test3 nextForm = new Hazard\_per\_test3(); // Replace with your next form logic

nextForm.Show();

this.Close();

};

resultsForm.Controls.Add(resultsLabel);

resultsForm.Controls.Add(nextButton);

resultsForm.ShowDialog();

**Obstacle:** Want to Add a form when the video ends so it can open the next video  
**Solution:**

// Show the results in a new form with a "Next" button

Form resultsForm = new Form

{

Text = "Final Results",

Size = new System.Drawing.Size(342, 151),

StartPosition = FormStartPosition.CenterParent

};

Label resultsLabel = new Label

{

Text = $"Your total points: {correctClicks}",

Dock = DockStyle.Fill,

AutoSize = false,

TextAlign = ContentAlignment.TopLeft,

Padding = new Padding(10)

};

Button nextButton = new Button

{

Text = "Next",

Dock = DockStyle.Bottom,

Height = 40

};

nextButton.Click += (s, e) =>

{

resultsForm.Close();

// Open the next form with the new video

Hazard\_per\_test3 nextForm = new Hazard\_per\_test3(); // Replace with your next form logic

nextForm.Show();

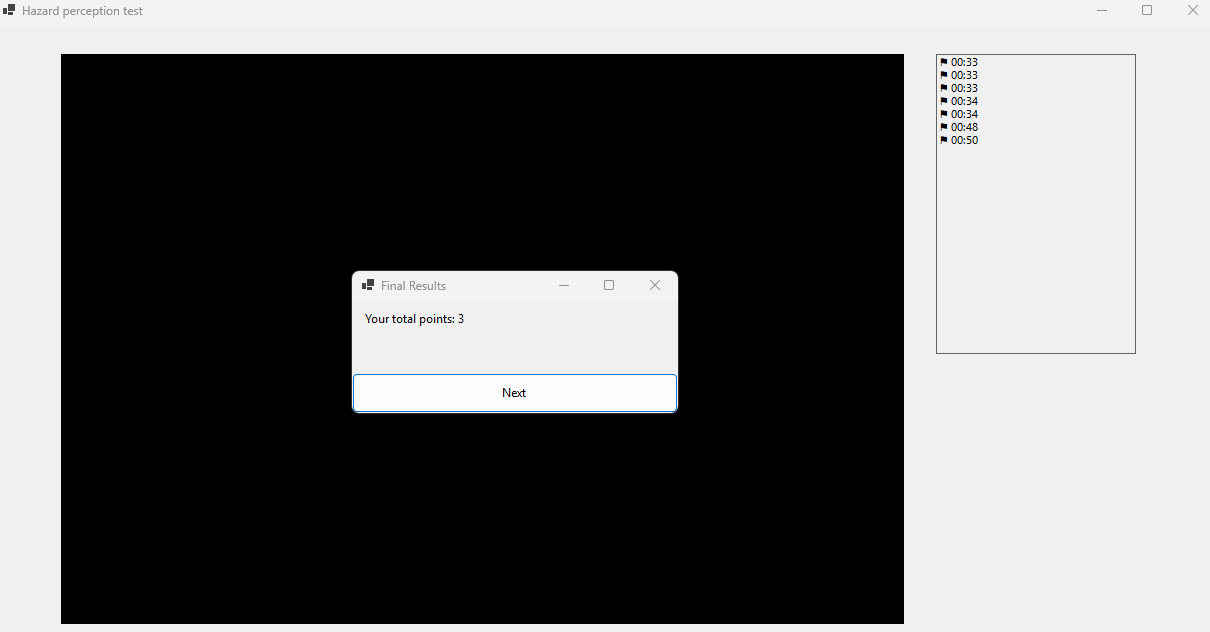
this.Close();

};

resultsForm.Controls.Add(resultsLabel);

resultsForm.Controls.Add(nextButton);

resultsForm.ShowDialog();

***  
  
Review:***

|  |  |
| --- | --- |
| **Success Criteria** | **Result** |
| Does the system allow users to mark topics as completed? |  |
| Does the system persist the checkbox state even when the user navigates away from the page? |  |
| When can a users click on a particular topic, what do they see as regards all the signs connected to it? |  |
| Does every traffic sign have a clear image and a description? |  |
| Does the progress of each topic update, at the instance when the user or the system, completes or sets the topic to the incomplete status? |  |
| Is it possible for the user to flow from one page to another (topics, progress, settings, signs)? |  |
| On the page showing the progress, must all the topics that were finished be indicated with perfect icons? |  |
| Is it easy to use the software to select topics, to look for signs whereas checkboxes must be manipulative? |  |
| Are users able to remove their findings on specific or on all specific topics in case they want a fresh start? |  |
| Are there any options that allow a user to see their performance over the course of the whole program on a single page? |  |
| Are checkboxes and progress indicators (checkmarks, completion percentages) clearly visible and easy to interact with? |  |
| Can users dynamically change the background color using a ColorDialog in the settings page? |  |
| Can users adjust font size using a slider, and toggle font styles (bold/italic) using checkboxes? |  |
| Are all user settings (background color, font size, font styles) applied globally and persisted between sessions? |  |
| Does the settings page provide real-time updates with previews for font and style changes? |  |
| Are images loaded efficiently without causing performance issues, even with a large number of topics or signs? |  |
| Can users interact with images, such as zooming in for better visibility of signs? |  |
| Does the system handle missing or corrupted image files gracefully by showing placeholders? |  |
| Does the layout remain clean, with proper spacing, alignment, and legibility for readability? |  |
| Is there clear error handling for issues like missing data, image errors, or faulty inputs? |  |
| Does the system load quickly without significant delays or lag when switching between pages or topics? |  |
| Does the system visually distinguish between completed and incomplete topics? |  |
| Is the reset button functional, deleting all relevant data files (scores, flagged questions) with success confirmation? |  |
| Does the system prevent crashes or unexpected behavior from faulty inputs or user actions? |  |
| Can flagged questions be viewed separately for easy access? |  |
| Are all forms and pages centered on the screen when opened? |  |
| Does the system handle high data volumes (e.g., many topics or signs) without performance issues? |  |
| Can additional content, such as multimedia (videos, animations), be added without major redesigns? |  |
| Is the system flexible enough to integrate new topics, signs, or complex features (e.g., quizzes, assessments)? |  |
| Does the overall user interface and experience align with personalization, usability, and consistency goals? |  |