COMP1551 Application Development Coursework

Educational quiz game

Maks Domas Smirnov – ID: 000979299

Table of Contents

[Design diagrams 3](#_Toc894280)

[Use-case diagram 3](#_Toc894281)

[Class diagram 4](#_Toc894282)

[Design explanation 5](#_Toc894283)

[Bugs/Weaknesses/Strengths 5](#_Toc894284)

[Appendix 6](#_Toc894285)

[Source Code 6](#_Toc894286)

[MainMenu class 6](#_Toc894287)

[LeaderboardMenu class 7](#_Toc894288)

[FinishMenu class 8](#_Toc894289)

[DBSingleton class 9](#_Toc894290)

[Answer class 13](#_Toc894291)

[Score class 14](#_Toc894292)

[Question class 14](#_Toc894293)

[InputAnswerQuestion class 15](#_Toc894294)

[MultipleChoiceQuestion class 16](#_Toc894295)

[MusicQuestion class 17](#_Toc894296)

[PictureQuestion class 19](#_Toc894297)

[YesOrNoQuestion class 20](#_Toc894298)

[Self-assessment form 21](#_Toc894299)

[User documentation 23](#_Toc894300)

[Application specification 23](#_Toc894301)

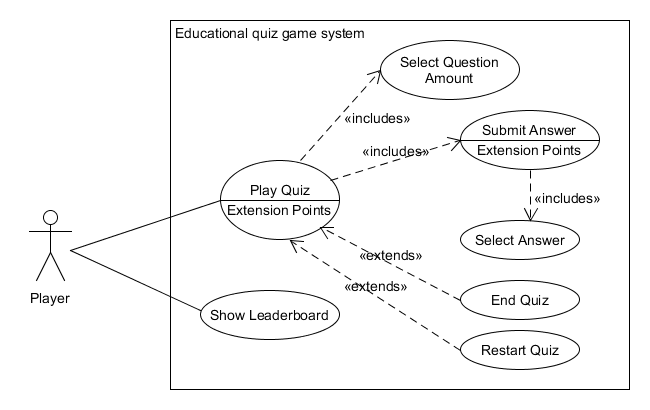
[Screenshots of the working application with explanations 23](#_Toc894302)

[References 25](#_Toc894303)

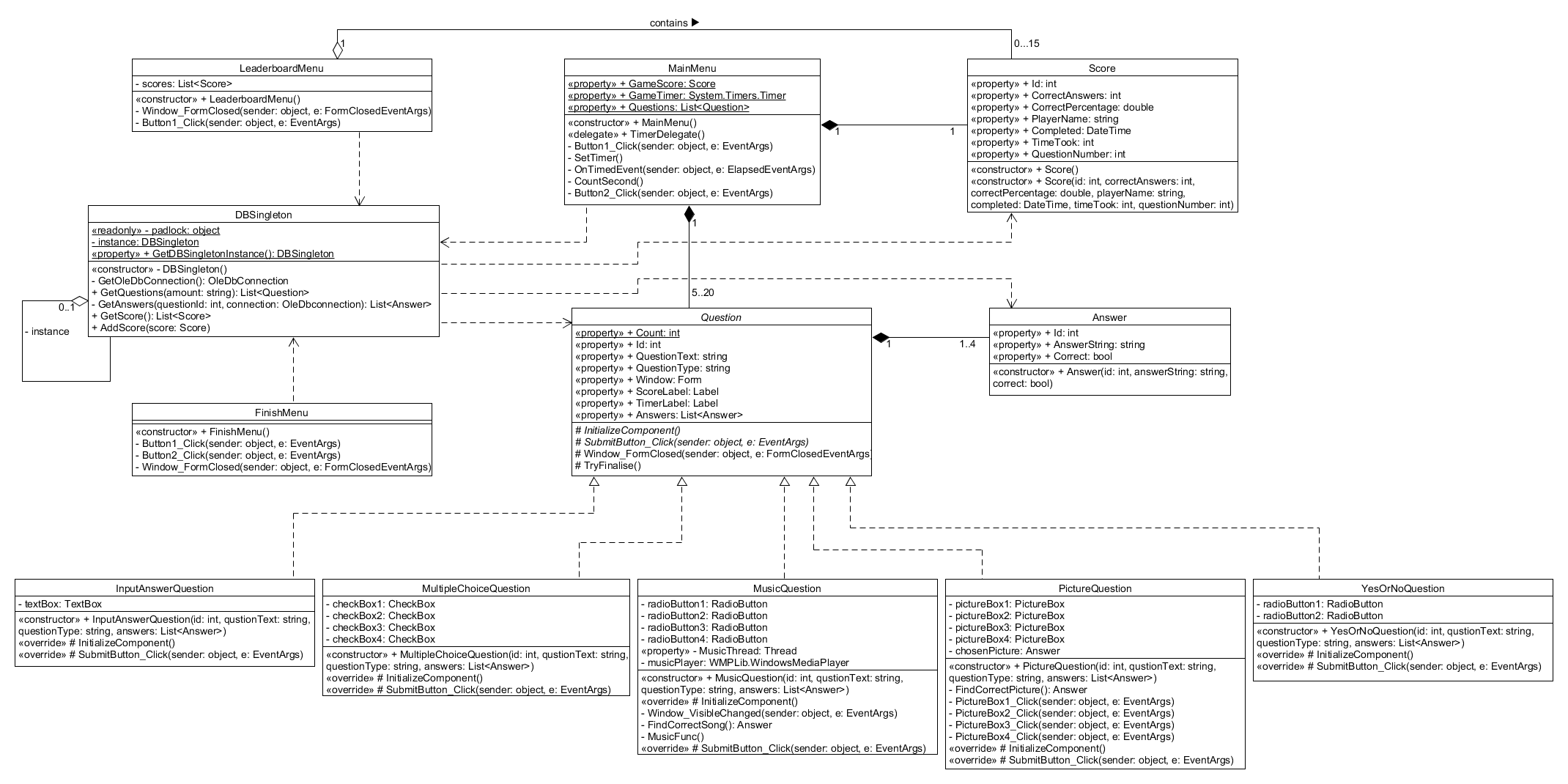
# Design diagrams

All diagrams will be included in the project folder so that they can be observed in closer detail if there is a need.

## Use-case diagram



## Class diagram



## Design explanation

Starting with the Use Case Diagram, as the program is relatively simple, there are only two use cases a player can interact with from the start. Show Leaderboard use case provides the player with the overview of the best scores of the quiz in a text format. The main use-case of the program is the Play Quiz. For the player to continue with the quiz, he must select the desired number of questions for the quiz using the Select Question Amount use case. Afterwards to be able to submit an answer using the Submit Answer use case, the player must first select the appropriate answers using the Select Answer use case. After all the questions have been answered, the player may close the application using the Exit Quiz use case or try to solve another quiz using the Restart Quiz use case.

The Class Diagram was designed with Object Oriented Concepts in mind. The entities have been abstracted and named to include only the necessary classes for the quiz. Considering the concepts Polymorphism and Inheritance, all the menu classes (MainMenu, LeaderboardMenu, FinishMenu) inherit from a .NET provided Form class. Also, there is an abstract Question class and 5 different question types which inherit some implementations from the abstract Question class and override some abstract methods from it. The MainMenu class contains a generic List of type Question. As the Question class is abstract it cannot be instantiated, but the List can be filled up with the different implementations of the abstract class. There is also an example of overloading. In the class Score there are two constructors with different signatures which are used in different situations. Speaking off Encapsulation, most variables are safely encapsulated in auto-properties. An example of a Singleton Pattern also exists in a DBSingleton class, the one responsible for all activities related to the interaction with the database. The notion of Model-View Separation was adhered to as much as possible.

# Bugs/Weaknesses/Strengths

A lot of effort was put into this program to try to create something easily manageable, upgradable and implementable. Nonetheless, some issues persist.

1. The program was not created to amaze the human eye. As such, a more pleasant user interface could have been developed.
2. In the Leaderboard, even though string formatting has been used the columns of the text still do not line-up nicely as expected.
3. While trying to solve the questions of type Music, in very rare instances the background music will not start playing for some time and/or will not stop playing even if the player has moved on to the next question. Nonetheless, after a small pause, the issue disappears so it does not appear to be very bothering to the playr.
4. While trying to solve the questions of type Picture, the player must click on the picture which he thinks answers the question. After clicking on the picture, the player does not receive any visual feedback, even though the picture which was clicked on will now be considered as selected. It could be useful to add an outline to the borders of the picture which was selected to indicate to the player that the click was submitted. For example, this could be implemented with an animation which shows a glowing halo around the picture.
5. Immediately after the player moves on to the next question, the label which contains the timer sometimes flickers as if it appears on the question a fraction of a second late.
6. Unfortunately, the order of the questions is not randomised which means that the quiz will contain questions in the same order, because the SQL query (SELECT TOP [amount of questions]) returns the rows in the same order every time. The good thing about the program is that if the rows from the database were returned in a randomised order the program will function as it should. This is because the order of appearance of the questions is not hardcoded and the program does not care which type of question comes after which type of question.

# Appendix

## Source Code

### MainMenu class

// MainMenu class acts as the starting point for the quiz. Implements the Form class.

public partial class MainMenu : Form

{

//Delegate for the timer.

public delegate void TimerDelegate();

//GameScore variable is the score for the quiz that is currently being solved.

public static Score GameScore { get; set; } = null;

//GameTimer variable is a timer for the current quiz. Stops when the quiz is finished.

public static System.Timers.Timer GameTimer { get; set; } = null;

//A list of questions for the current quiz.

public static List<Question> Questions { get; set; } = null;

public MainMenu()

{

InitializeComponent();

}

//Button to begin a new quiz

private void Button1\_Click(object sender, EventArgs e)

{

//Checks if a user has selected the preferred number of questions for the quiz.

if (comboBox1.SelectedItem != null)

{

GameScore = new Score();

//Retrieves the selected number of questions from the database.

Questions = DBSingleton.GetDBSingletonInstance.GetQuestions(comboBox1.SelectedItem.ToString());

//Sets the MainMenu Form to invisible.

this.Visible = false;

//Initialization of the Text property of ScoreLabel and TimerLabel

Questions[Question.Count].ScoreLabel.Text = "Question " + Questions[Question.Count].Id + " out of " + Questions.Count;

Questions[Question.Count].TimerLabel.Text = "Seconds taken already: " + GameScore.TimeTook + "s.";

//Sets the Window of the next question to visible.

Questions[Question.Count].Window.Visible = true;

SetTimer();

}

else {

MessageBox.Show("Please select the number of questions.");

}

}

//Initialisation of the timer which ticks every second

private void SetTimer()

{

GameTimer = new System.Timers.Timer(1000);

GameTimer.Elapsed += OnTimedEvent;

GameTimer.AutoReset = true;

GameTimer.Enabled = true;

}

//The event is called every time the timer elapses

private void OnTimedEvent(object source, ElapsedEventArgs e)

{

TimerDelegate handler = CountSecond;

Invoke(handler);

}

// Method for the timer delegate

private void CountSecond()

{

//Increment time of the quiz

GameScore.TimeTook++;

//Change the TimerLabel of the current question to reflect the time spent.

Questions[Question.Count].TimerLabel.Text = "Seconds taken already: " + GameScore.TimeTook + "s.";

}

//Method to show leaderboards in a separate form.

private void Button2\_Click(object sender, EventArgs e)

{

Visible = false;

LeaderboardMenu leaderboardMenu = new LeaderboardMenu();

}

}

### LeaderboardMenu class

// LeaderboardMenu class implements Form and shows the score from previous quiz sessions.

public partial class LeaderboardMenu : Form

{

//List of all previous scores.

private List<Score> scores;

public LeaderboardMenu()

{

//Retrives all scores from the database

scores = DBSingleton.GetDBSingletonInstance.GetScores();

InitializeComponent();

//Sets the text for the label to display all the scores.

label2.Text = string.Format("{0, 5}", "Nr") + string.Format("{0, 20}", "Player name")

+ string.Format("{0, 20}", "Percentage") + string.Format("{0, 20}", "Time taken (s)" + "\n\r");

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

{

label2.Text += string.Format("{0, 5}", i + 1) + string.Format("{0, 20}", scores[i].PlayerName)

+ string.Format("{0, 20}", scores[i].CorrectPercentage) + string.Format("{0, 20}", scores[i].TimeTook) + "\n\r";

}

Visible = true;

}

private void Window\_FormClosed(object sender, FormClosedEventArgs e)

{

Application.Exit();

}

//Pressing the Go Back button actually restarts the whole application instead of switching to the MainMenu.

private void Button1\_Click(object sender, EventArgs e)

{

Application.Restart();

}

}

### FinishMenu class

//FinishMenu class implements Form and is showed when all the questions have been answered.

public partial class FinishMenu : Form

{

public FinishMenu()

{

InitializeComponent();

label1.Text = MainMenu.GameScore.CorrectAnswers.ToString() + " out of " + MainMenu.Questions.Count.ToString() + " correct answers.\n" +

"You completed the quiz in " + MainMenu.GameScore.TimeTook + "seconds.\nPlease enter your name below to save the score.";

}

//Method to save the score and play another quiz.

private void Button1\_Click(object sender, EventArgs e)

{

//Checks if the textbox is not empty and is not too long.

if (textBox1.Text == "" || textBox1.TextLength > 10)

{

MessageBox.Show("Please input a name of at least one and no more than 10 characters.");

}

else

{

//Set the player name to the text in the textbox.

MainMenu.GameScore.PlayerName = textBox1.Text;

//Add the score of the currently completed quiz to the database.

DBSingleton.GetDBSingletonInstance.AddScore(MainMenu.GameScore);

Application.Restart();

}

}

//Method to save the score and exit the application.

private void Button2\_Click(object sender, EventArgs e)

{

//Checks if the textbox is not empty and is not too long.

if (textBox1.Text == "" || textBox1.TextLength > 10)

{

MessageBox.Show("Please input a name of at least one and no more than 10 characters.");

}

else

{

//Set the player name to the text in the textbox.

MainMenu.GameScore.PlayerName = textBox1.Text;

//Add the score of the currently completed quiz to the database.

DBSingleton.GetDBSingletonInstance.AddScore(MainMenu.GameScore);

Application.Exit();

}

}

private void Window\_FormClosed(object sender, FormClosedEventArgs e)

{

Application.Exit();

}

}

### DBSingleton class

// A Database class which uses a Singleton pattern to only allow a single instance of itself and is used for interaction

//with a database.

internal sealed class DBSingleton

{

//A lock object to make the accessing of a single instance thread safe.

private static readonly object padlock = new object();

//A Singleton instance of the class.

private static DBSingleton instance = null;

//Property used for the retrieval of the Singleton instance.

public static DBSingleton GetDBSingletonInstance

{

get

{

lock (padlock)

{

if (instance == null)

{

instance = new DBSingleton();

}

return instance;

}

}

}

// Constructor. Used only once.

private DBSingleton()

{

}

// Method returns a database connection.

private OleDbConnection GetOleDbConnection()

{

String connection = @"Provider=Microsoft.JET.OLEDB.4.0;

Data Source =" + Path.Combine(AppDomain.CurrentDomain.BaseDirectory, @"DB\AppDevProjectDB.mdb");

OleDbConnection myConnection = new OleDbConnection(connection);

return myConnection;

}

//Method returns a List of all questions in a database.

public List<Question> GetQuestions(string amount)

{

List<Question> questions = new List<Question>();

OleDbConnection connection = GetOleDbConnection();

//Selects the amount of questions selected by the user from a database.

string query = "SELECT TOP " + amount.ToString() + " \* FROM Question;";

OleDbCommand questionCommand = new OleDbCommand(query, connection);

try

{

connection.Open();

OleDbDataReader questionReader = questionCommand.ExecuteReader();

while (questionReader.Read())

{

int id = (int)questionReader["ID"];

string questionText = (string)questionReader["QuestionText"];

string questionType = (string)questionReader["QuestionType"];

Question newQuestion = null;

//A switch case is used to determine which kind of question has been extracted from a database.

switch (questionType)

{

case "InputAnswer":

newQuestion = new InputAnswerQuestion(id, questionText, questionType, GetAnswers(id, connection));

break;

case "MultipleChoice":

newQuestion = new MultipleChoiceQuestion(id, questionText, questionType, GetAnswers(id, connection));

break;

case "Music":

newQuestion = new MusicQuestion(id, questionText, questionType, GetAnswers(id, connection));

break;

case "Picture":

newQuestion = new PictureQuestion(id, questionText, questionType, GetAnswers(id, connection));

break;

case "YesOrNo":

newQuestion = new YesOrNoQuestion(id, questionText, questionType, GetAnswers(id, connection));

break;

}

//If an abnormal question type is discovered, an exception is thrown.

if (newQuestion != null)

{

questions.Add(newQuestion);

}

else

{

throw new Exception("newQuestion object is null.");

}

}

//Checking if there are any questions in the database at all.

if (questions.Count == 0)

{

throw new Exception("Database is empty.");

}

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine("Exception: " + ex);

}

finally

{

connection.Close();

}

return questions;

}

//Returns a list of answers by using a provided id of a specific question.

private List<Answer> GetAnswers(int questionId, OleDbConnection connection)

{

List<Answer> answers = new List<Answer>();

//Query to select all answers which have a specific question\_id.

string query = "SELECT \* FROM Answer WHERE Question\_Id = " + questionId + ";";

OleDbCommand answerCommand = new OleDbCommand(query, connection);

try

{

OleDbDataReader answerReader = answerCommand.ExecuteReader();

while (answerReader.Read())

{

int id = (int)answerReader["ID"];

string answerText = (string)answerReader["AnswerText"];

bool correct = (bool)answerReader["Correct"];

answers.Add(new Answer(id, answerText, correct));

}

//Checks if there are any answers at all.

if (answers.Count == 0)

{

throw new Exception("Database is empty.");

}

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine("Exception: " + ex);

}

return answers;

}

//Returns a list of all the scores stored in the database to show in the leaderboards.

public List<Score> GetScores()

{

List<Score> scores = new List<Score>();

OleDbConnection connection = GetOleDbConnection();

//Nested Select query used to retrieve top 15 scores based on percentage of correctly answered questions and the time it took

//to complete the quiz.

string query = "SELECT TOP 15 \* FROM (SELECT \* FROM Score ORDER BY CorrectPercentage DESC, TimeTook ASC, PlayerName DESC);";

OleDbCommand scoreCommand = new OleDbCommand(query, connection);

try

{

connection.Open();

OleDbDataReader scoreReader = scoreCommand.ExecuteReader();

while (scoreReader.Read())

{

int id = (int)scoreReader["ID"];

double correctPercentage = (int)scoreReader["CorrectPercentage"];

DateTime completed = (DateTime)scoreReader["Completed"];

string playerName = (string)scoreReader["PlayerName"];

int timeTook = (int)scoreReader["TimeTook"];

int correctAnswers = (int)scoreReader["CorrectAnswers"];

int questionNumber = (int)scoreReader["QuestionNumber"];

Score newScore = new Score(id, correctAnswers, correctPercentage,

playerName, completed, timeTook, questionNumber);

//Check if it is a suitable score.

if (newScore != null)

{

scores.Add(newScore);

}

else

{

throw new Exception("newScore object is null.");

}

}

//Check if there are any score at all.

if (scores.Count == 0)

{

throw new Exception("Database is empty.");

}

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine("Exception: " + ex);

}

finally

{

connection.Close();

}

return scores;

}

//Method is used to store the score of a completed quiz to a database.

public void AddScore(Score score)

{

OleDbConnection connection = GetOleDbConnection();

//Query used to find the highest id in the table.

string query = "SELECT ID FROM Score;";

OleDbCommand maxIDCommand = new OleDbCommand(query, connection);

int maxID = 0;

try

{

connection.Open();

OleDbDataReader maxIDReader = maxIDCommand.ExecuteReader();

while (maxIDReader.Read())

{

if (maxID < (int)maxIDReader["ID"])

{

maxID = (int)maxIDReader["ID"];

}

}

maxID++;

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine("Exception: " + ex);

}

finally

{

connection.Close();

}

//A query used to insert a new tuple into the table.

query = "INSERT INTO Score(ID, CorrectPercentage, Completed, PlayerName, TimeTook, CorrectAnswers, QuestionNumber)" +

"VALUES (" + maxID + " , " + score.CorrectPercentage + " , #" + score.Completed.Date + "# , '" +

score.PlayerName + "' , " + score.TimeTook + " , " + score.CorrectAnswers + " , " + score.QuestionNumber + ");";

OleDbCommand scoreCommand = new OleDbCommand(query, connection);

try

{

connection.Open();

scoreCommand.ExecuteNonQuery();

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine("Exception: " + ex);

}

finally

{

connection.Close();

}

}

}

### Answer class

//A class to store the details of an answer of a particular question.

public class Answer

{

public Answer(int id, string answerString, bool correct) {

this.Id = id;

this.AnswerString = answerString;

this.Correct = correct;

}

public string AnswerString { get; set; }

public int Id { get; set; }

public bool Correct { get; set; }

}

### Score class

//Score class is used to store information about a players stats in a specific quiz session.

public class Score

{

public int Id { get; set; }

public int CorrectAnswers { get; set; }

public double CorrectPercentage { get; set; }

public string PlayerName { get; set; }

public DateTime Completed { get; set; }

public int TimeTook { get; set; }

public int QuestionNumber { get; set; }

// Constructor used to create a score when a new quiz is started.

public Score()

{

CorrectAnswers = 0;

TimeTook = 0;

}

// Constructor used to create scores to display for the leaderboard.

public Score(int id, int correctAnswers, double correctPercentage,

string playerName, DateTime completed, int timeTook, int questionNumber)

{

Id = id;

CorrectAnswers = correctAnswers;

CorrectPercentage = correctPercentage;

PlayerName = playerName;

Completed = completed;

TimeTook = timeTook;

QuestionNumber = questionNumber;

}

}

### Question class

// Abstract class for different question types

public abstract class Question

{

// Variable Count stores the number of the question the player is currently answering

public static int Count { get; set; } = 0;

public int Id { get; set; }

public string QuestionText { get; set; }

public string QuestionType { get; set; }

public Form Window { get; set; }

public Label ScoreLabel { get; set; }

public Label TimerLabel { get; set; }

public List<Answer> Answers { get; set; }

// Abstract method to initialise the UI Window variable of a specific question

protected abstract void InitializeComponent();

protected abstract void SubmitButton\_Click(object sender, EventArgs e);

protected void Window\_FormClosed(object sender, FormClosedEventArgs e)

{

Application.Exit();

}

//Method is called each time a question is answered to check if it was the last question of the quiz

//and proceed to the FinishMenu

protected void TryFinalise() {

//Checks if the question counter reached the end of the List of all questions

if (Count == MainMenu.Questions.Count - 1)

{

//Stop the game timer so that it is possible to save the value for the score.

MainMenu.GameTimer.Stop();

//Get the time of completion of the quiz.

MainMenu.GameScore.Completed = DateTime.Now;

//Calculating the percentage of correct answers.

MainMenu.GameScore.CorrectPercentage = ((double)MainMenu.GameScore.CorrectAnswers / MainMenu.Questions.Count) \* 100;

//Record the total number of questions in the quiz.

MainMenu.GameScore.QuestionNumber = MainMenu.Questions.Count;

//Show the FinishMenu

FinishMenu finishMenu = new FinishMenu();

Window.Visible = false;

finishMenu.Visible = true;

}

//If not the last question, continue.

else

{

//Increase the question counter.

Count++;

Window.Visible = false;

//Set the ScoreLabel of the next question to show the position in the quiz.

MainMenu.Questions[Question.Count].ScoreLabel.Text = "Question " + MainMenu.Questions[Question.Count].Id + " out of " + MainMenu.Questions.Count;

//Show the next question.

MainMenu.Questions[Question.Count].Window.Visible = true;

}

}

}

### InputAnswerQuestion class

//A question type where you have to input your own answer to the question into a textbox.

public class InputAnswerQuestion : Question

{

private TextBox textBox;

public InputAnswerQuestion(int id, string questionText, string questionType, List<Answer> answers)

{

Id = id;

QuestionText = questionText;

QuestionType = questionType;

Answers = answers;

InitializeComponent();

}

protected override void InitializeComponent()

{

Intentionally left blank to conserve space.

}

protected override void SubmitButton\_Click(object sender, EventArgs e)

{

//Check if the user provided an answer.

if (textBox.Text == "")

{

MessageBox.Show("Please input something into the text box.");

}

else

{

//If the answer is correct, increase the score.

if (textBox.Text.ToLower() == Answers[0].AnswerString.ToLower())

{

MainMenu.GameScore.CorrectAnswers++;

}

//Check if this question is the last one.

TryFinalise();

}

}

}

### MultipleChoiceQuestion class

//A question type where you are allowed multiple choices for answering the question.

public class MultipleChoiceQuestion : Question

{

private CheckBox checkBox1;

private CheckBox checkBox2;

private CheckBox checkBox3;

private CheckBox checkBox4;

public MultipleChoiceQuestion(int id, string questionText, string questionType, List<Answer> answers)

{

Id = id;

QuestionText = questionText;

QuestionType = questionType;

Answers = answers;

InitializeComponent();

}

protected override void InitializeComponent()

{

Intentionally left blank to conserve space.

}

protected override void SubmitButton\_Click(object sender, EventArgs e)

{

//Check if the user provided an answer.

if (!checkBox1.Checked && !checkBox2.Checked && !checkBox3.Checked && !checkBox4.Checked)

{

MessageBox.Show("Please check at least one box.");

}

else

{

//If the answer is correct, increase the score.

if (checkBox1.Checked == Answers[0].Correct && checkBox2.Checked == Answers[1].Correct &&

checkBox3.Checked == Answers[2].Correct && checkBox4.Checked == Answers[3].Correct)

{

MainMenu.GameScore.CorrectAnswers++;

}

//Check if this question is the last one.

TryFinalise();

}

}

}

### MusicQuestion class

//A question type where you have to select a single author whose music is playing in the background.

public class MusicQuestion : Question

{

private RadioButton radioButton1;

private RadioButton radioButton2;

private RadioButton radioButton3;

private RadioButton radioButton4;

private Thread MusicThread { get; set; }

private WMPLib.WindowsMediaPlayer musicPlayer;

public MusicQuestion(int id, string questionText, string questionType, List<Answer> answers)

{

Id = id;

QuestionText = questionText;

QuestionType = questionType;

Answers = answers;

InitializeComponent();

}

protected override void InitializeComponent()

{

Intentionally left blank to conserve space.

}

//Event which is triggered every time the Visible property of the Window variable is changed.

private void Window\_VisibleChanged(object sender, EventArgs e)

{

//If the window is visible, then the thread responsible for playing music should start.

if (Window.Visible)

{

MusicThread.Start();

}

else

{

MusicThread.Abort();

}

}

//Method is used to find the correct answer from the list of all answers,

//so that it can be compared to the users selection.

private Answer FindCorrectSong()

{

bool found = false;

int count = 0;

while (!found)

{

if (Answers[count].Correct == true)

{

found = true;

}

else

{

count++;

}

}

return Answers[count];

}

//Method used by the thread to start playing music in the background.

private void MusicFunc()

{

musicPlayer = new WMPLib.WindowsMediaPlayer();

musicPlayer.URL = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, @"Music\" + FindCorrectSong().AnswerString + ".mp3");

musicPlayer.controls.play();

}

protected override void SubmitButton\_Click(object sender, EventArgs e)

{

//Check if the user provided an answer.

if (!radioButton1.Checked && !radioButton2.Checked && !radioButton3.Checked && !radioButton4.Checked)

{

MessageBox.Show("Please check at least one box.");

}

else

{

//If the answer is correct, increase the score.

if (radioButton1.Checked == Answers[0].Correct && radioButton2.Checked == Answers[1].Correct &&

radioButton3.Checked == Answers[2].Correct && radioButton4.Checked == Answers[3].Correct)

{

MainMenu.GameScore.CorrectAnswers++;

musicPlayer.controls.stop();

}

//Check if this question is the last one.

TryFinalise();

}

}

}

### PictureQuestion class

//A question type where the user has to select a single corresponding picture to answer the question

public class PictureQuestion : Question

{

private PictureBox pictureBox1;

private PictureBox pictureBox2;

private PictureBox pictureBox3;

private PictureBox pictureBox4;

private Answer chosenPicture;

public PictureQuestion(int id, string questionText, string questionType, List<Answer> answers)

{

Id = id;

QuestionText = questionText;

QuestionType = questionType;

Answers = answers;

chosenPicture = null;

InitializeComponent();

}

protected override void InitializeComponent()

{

Intentionally left blank to conserve space.

}

//Method used to find the correct answer in the list of answers

//to later compare it with the users selection.

private Answer FindCorrectPicture()

{

bool found = false;

int count = 0;

while (!found)

{

if (Answers[count].Correct == true)

{

found = true;

}

else

{

count++;

}

}

return Answers[count];

}

private void PictureBox1\_Click(object sender, EventArgs e)

{

chosenPicture = Answers[0];

}

private void PictureBox2\_Click(object sender, EventArgs e)

{

chosenPicture = Answers[1];

}

private void PictureBox3\_Click(object sender, EventArgs e)

{

chosenPicture = Answers[2];

}

private void PictureBox4\_Click(object sender, EventArgs e)

{

chosenPicture = Answers[3];

}

protected override void SubmitButton\_Click(object sender, EventArgs e)

{

//Check if the user provided an answer.

if (chosenPicture == null)

{

MessageBox.Show("Please select a picture.");

}

else

{

//If the answer is correct, increase the score.

if (chosenPicture.Id == FindCorrectPicture().Id)

{

MainMenu.GameScore.CorrectAnswers++;

}

//Check if this question is the last one.

TryFinalise();

}

}

}

### YesOrNoQuestion class

//A question type where the user has to select one option, either yes or no, to answer the question.

public class YesOrNoQuestion : Question

{

private RadioButton radioButton1;

private RadioButton radioButton2;

public YesOrNoQuestion(int id, string questionText, string questionType, List<Answer> answers)

{

Id = id;

QuestionText = questionText;

QuestionType = questionType;

Answers = answers;

InitializeComponent();

}

protected override void InitializeComponent()

{

Intentionally left blank to conserve space.

}

protected override void SubmitButton\_Click(object sender, EventArgs e)

{

//Check if the user provided an answer.

if (!radioButton1.Checked && !radioButton2.Checked)

{

MessageBox.Show("Please check at least one box.");

}

else

{

//If the answer is correct, increase the score.

if ((Answers[0].Correct && radioButton1.Checked) || (Answers[1].Correct && radioButton2.Checked))

{

MainMenu.GameScore.CorrectAnswers++;

}

//Check if this question is the last one.

TryFinalise();

}

}

}

## Self-assessment form

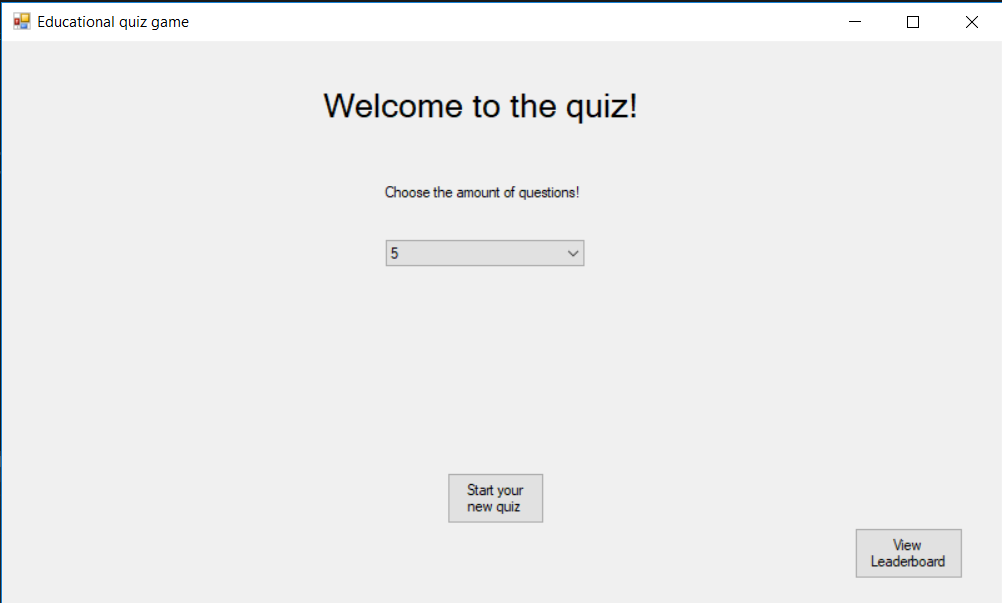
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | % | P | A | S | G | E | O | Comments |
| Object Oriented Design  (including UML diagrams and how the design is reflected into code and any other technical documentation included in the report) | 15 |  |  |  |  |  | x | All OO fundamentals are used in the program. Class Diagram reflects the code – names of fields and method signatures are the same. All associations and dependencies are clearly shown. |
| Use of desired features and concepts | | | | | | | | |
| Well named variables and, methods. Decisions, Iteration. Overloading | 10 |  |  |  |  |  | x | Proper encapsulation used throughout the program, auto properties. If statements and switch used. For and while loops used. Overloading used. |
| Objects and Classes. Collections | 10 |  |  |  |  |  | x | Entities abstracted to classes. Generic Lists are used in several places. |
| Events controls. Validation | 10 |  |  |  |  |  | x | Events used frequently in the program. Many different types of controls appear. User input validated to prevent errors. Exceptions in the DBSingleton class handled correctly. |
| Threads. Animations. | 10 |  |  |  |  | x |  | Threads are used for the background music in the Music question type. Delegates are used for the timer, which in itself is a thread. Unfortunately, no animations demonstrated. |
| Inheritance. Interfaces | 10 |  |  |  |  |  | x | All 3 menu classes inherit from Form. Abstract class for questions exists, 5 different question types implement it. No interfaces used as the scale of the program indicates no need for them. |
| Persistence – storing and reading data from a database | 10 |  |  |  |  |  | x | A separate class DBSingleton handles all interaction with the database. Nested queries used. |
| Patterns | 5 |  |  |  |  |  | x | Singleton pattern is used in the database class. Model-view separation has also been attempted. |
| Extra features (such as using web services) | 5 |  |  |  |  |  | x | Implemented background music. There is also a timer which ticks every second. |
| Acceptance Testing  (includes how well the app is presented and the ability to answer technical questions) | 10 |  |  |  |  |  | x | I believe I can demonstrate my program relatively well and can answer all the technical questions. |
| Accurate Self- Assessment | 5 |  |  |  |  |  | X | I genuinely believe I have assessed myself fairly and accurately. |
| Possible AO? |  |  |  |  |  |  |  |  |

## User documentation

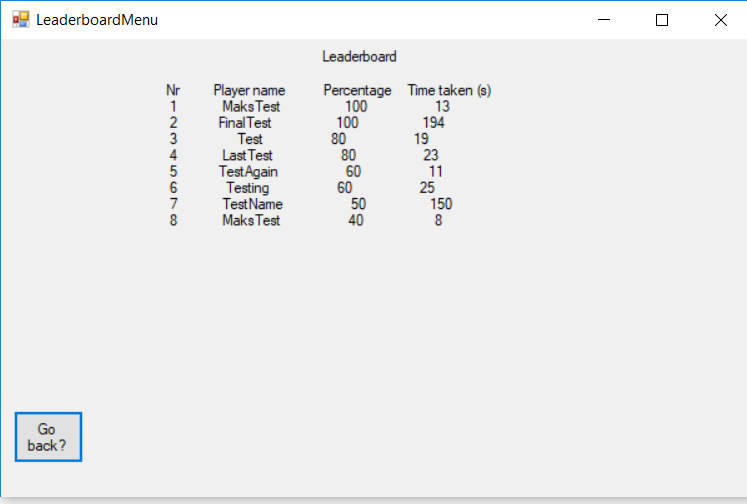
### Application specification

The application lets the player attempt an educational quiz and saves the score so that the player can compare his attempts with other players. The user may also view the best results of the quiz in a leaderboard. Before the player starts the quiz, he must select the number of questions (5, 10, 15 or 20) he would like the quiz to have. There are 5 different types of questions, each with their own way of solving it. The player must answer the questions until he has answered all of them. The player is presented with the number of the question he is currently solving and the timer which shows how many seconds has elapsed since the player has started the quiz. After the player answers all the questions, he is presented with a new window, which shows how many questions the player has answered correctly and how long it took the player to complete the quiz. The player is also given the option of saving his own score and playing another quiz or saving his own score and exiting the application.

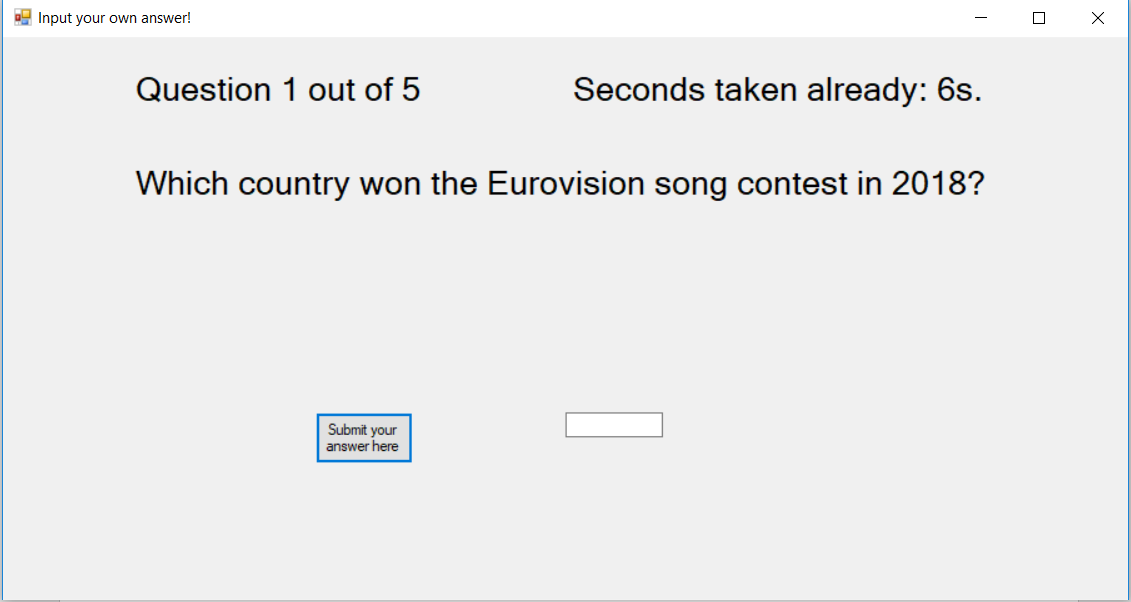
### Screenshots of the working application with explanations



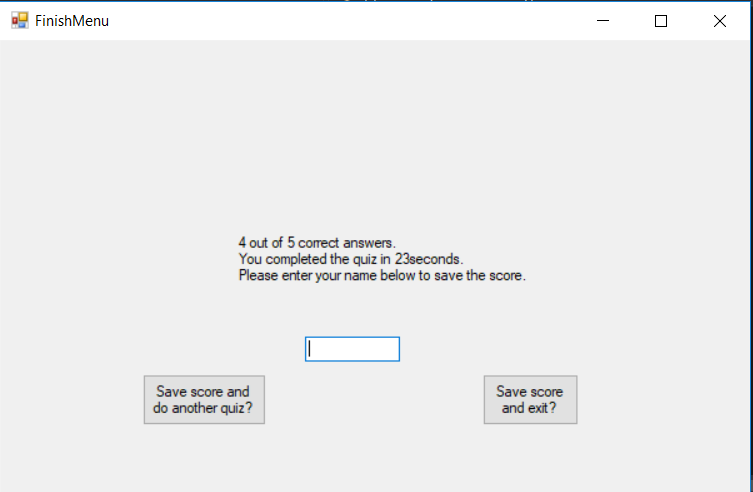
After the player starts the application he is presented with this menu. From here he can view the leaderboard or start a new quiz (this requires the amount of questions to be selected from the combobox, otherwise an error message will be displayed).



If the player selects to view the leaderboards, a new window appears showing the best scores in a text format. An option to go back appears in the bottom left corner, which when pressed will bring the player back to the main menu.



After starting the quiz, the player is presented with the question. Each new question will show how many questions are left until the end of the quiz (top left corner). Each question also shows the timer for the whole quiz (top right corner). All questions must be attempted, otherwise if the player tries to submit an answer without providing one, the program will not continue and will present an error message.



After answering all the questions, a new menu appears which shows the number of correct answers and the time it took the player to complete the quiz. To save the score and do another quiz or to save the score and exit the application the player must provide the name, otherwise an error message will occur.

## References

1. UMLet software has been used for the creation of the diagrams.
2. Different pictures and music for the questions have been found using Google searches. There is no intent to monetize this application or use the pictures and music maliciously.
3. Ideas for different questions and answers have been suggested by my fiancé Livija Murmaite.