horizontal line

**Sean Moylan G00299424**

**Eoghan O’Connor G00330972**

Quiz Me

**April 2020**

# OVERVIEW

This game was created as part of our Gesture based UI project carried out in 4th year Software Development. QuizMe is a quiz game that uses speech recognition to challenge the player with varying levels of difficulty.

# Purpose of the application

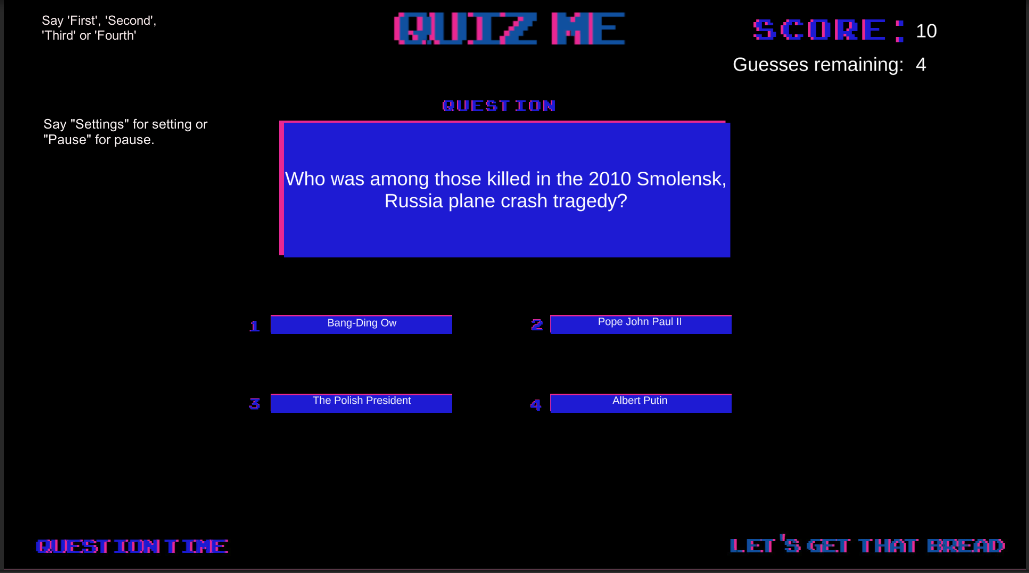
The application is very simple to use for users of all abilities. When the player starts the app they are greeted by a retro style game consisting of a simple black, purple and blue.



The user prompt in the top left shows the user what phrases are recognised by the scene. The “You said: ” prompt shows the user what phrase has been said. When the player says “Play” they are brought to the Level Difficulty scene.



The Level Difficulty scene allows the user to select the difficulty of the questions. All questions are from an open trivia database (<https://opentdb.com/>). The api allows you to request 50 questions at a time of varying difficulty and a wide range of topics. We decided this app should be available at all times, regardless of internet connection, to the user so we created three JSON files with 100 questions from the general knowledge section in each of the three difficulties. Same as the menu page the user is presented with a similar design to continue the retro design language as well as a brief prompt to what phrases the application will recognise and the phrases printed on the screen for the user to see. When the Player decides the level they’re brought to the Game scene.



The Game Scene continues the retro design, Here the user can see the user prompt in the top left of the scene for how to play the game. The question is presented in the centre of the scene with the answers below. The words selected for picking the option the user wants were picked after testing multiple different combinations of words, The selection depicted above was the most accurate. Once the user picks a phrase to select a question, they are met with the following prompt.



This prompt makes sure the user phrase has been recognised correctly.

# Gestures identified as appropriate for this application

After going through a few ideas from both sides, we decided to start work on some sort of quiz game. We both decided to research different aspects of gesture based UI, when we returned we discussed using an Xbox Kinetic and have the user physically point at the answer, or perform a certain physical movement to select the answer. We also looked into Myo armbands that will allow the user to hold up the corresponding amount of fingers in relation to what number the answers are labeled with.

Another option brought to our attention was to use buzz controllers along with a Raspberry Pi. Buzz! The Music Quiz was a game first released in October of 2005 on Playstation 2. The game came with a set of handheld controllers for the contestants to buzz to answer and then had 4 other buttons to select the multiple choice. Our idea was to create a game that would let you simply connect the Raspberry Pi to a TV, plug in the controllers and begin to play a quiz game with friends or family. Although we did not end up following this idea for our project, we definitely took ideas and inspiration from it.

Finally we looked at speech recognition and having the user completely control the game with their voice. Upon further discussion we decided voice was the best option due to its use ability and the lack of external hardware, in case you know a global pandemic takes place. Voice Recognition allows the user to speak a given set of words that will control the application, each word will launch different activities given the context in which the word is spoken. Once we decided on this we both further researched the different ways to implement voice recognition.

# Hardware used in creating the application

The hardware used in this application is minimal. We are using the microphone built into a laptop or a headphone microphone. When deciding what speech recognition engine to use we looked into Android and IOS speech recognition. We immediately disregarded IOS as only one of us has a Mac, Android speech recognition was a promising option. It is easy to implement and would open the application up to a very large user pool, however we decided to use Windows as the Unity Engine speech recognition is very well documented and having used unity in a module in semester 1 and enjoyed it we decided to use Unity. At present the application is Windows, however it would be easily refactored to work with multiple platforms.

# Architecture for the solution

The Architecture for quiz me is very simple. We didn’t want to add complexity where it was unnecessary. All questions of each category are in a separate JSON file, depending on the user preferred difficulty the corresponding JSON file is loaded into the QuestionGenerator class. JSON structure of the question is as follows.

"category":"Entertainment: Video Games",

"type":"multiple",

"difficulty":"easy",

"question":"Which game did \"Sonic The Hedgehog\" make his first appearance in?",

"correct\_answer":"Rad Mobile",

"incorrect\_answers":[

"Sonic The Hedgehog",

"Super Mario 64",

"Mega Man"]

Each question contains a category , A question type, difficulty, question, correct answer and a JSON array of incorrect answers. Each JSON file contains 100 questions.

QuestionClass

// Question class based on JSON format of stored questions

[System.Serializable]

public class Question

{

public string category { get; set; }

public string type { get; set; }

public string difficulty { get; set; }

public string question { get; set; }

public string correct\_answer { get; set; }

public string[] incorrect\_answers { get; set; }

}

This is the structure of the Question class in the Question Generator.

The initQuestion method loads all questions files and removes special characters using the Helper class which will be explained later.

void initQuestions()

{

// Method used to read .json files

// Calls loadQuestions to Deserialize json string

// Set the string paths to the 3 Json files

easyPath = Application.streamingAssetsPath + "/easy.json";

mediumPath = Application.streamingAssetsPath + "/medium.json";

hardPath = Application.streamingAssetsPath + "/hard.json";

// Open and read all json files

easyJson = File.ReadAllText(easyPath);

mediumJson = File.ReadAllText(mediumPath);

hardJson = File.ReadAllText(hardPath);

Helper.RemoveSpecialChars(easyJson);

Helper.RemoveSpecialChars(mediumJson);

Helper.RemoveSpecialChars(hardJson);

easy = loadQuestions(easyJson);

medium = loadQuestions(mediumJson);

hard = loadQuestions(hardJson);

// Convert ILists to arrays

easyQuestions = convertToArray(easy);

mediumQuestions = convertToArray(medium);

hardQuestions = convertToArray(hard);

// Passes in the selected difficulty from the previous scene

nextQuestion(Helper.difficulty);

}

The loadQuestion is used to deserialize json string to an IList of question objects. Next all questions are converted to arrays. In the convertToArrays method.

public static IList<Question> loadQuestions(string jsonString)

{

// Method for Deserializing json string to IList<Question> object

JsonTextReader reader = new JsonTextReader(new StringReader(jsonString));

IList<Question> listOfQs = JsonConvert.DeserializeObject<List<Question>>(jsonString);

return (List<Question>) listOfQs;

}

public Question[] convertToArray(IList<Question> questions)

{

// Converter for converting IList<Question> to Question[]

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

Question[] questionsArray = temp.ToArray();

return questionsArray;

}

The displayQuestion method puts the questions, correct answer and wrong answers onto the canvas so the user can see the options. The Helper class is again called to shuffle the possibleAnswers array.

private void displayQuestion(Question q)

{

// Display Question

question.SetText(q.question.ToString());

// Shuffle answers

List<string> possibleAnswers = new List<string>();

possibleAnswers.Add(q.incorrect\_answers[0]);

possibleAnswers.Add(q.incorrect\_answers[1]);

possibleAnswers.Add(q.incorrect\_answers[2]);

possibleAnswers.Add(q.correct\_answer);

// Debug.Log(possibleAnswers);

Helper.Shuffle(possibleAnswers);

int count = 1;

foreach (String ans in possibleAnswers)

{

switch (count)

{

case 1:

ans1.SetText(ans);

break;

case 2:

ans2.SetText(ans);

break;

case 3:

ans3.SetText(ans);

break;

case 4:

ans4.SetText(ans);

break;

default:

break;

}

count++;

}

}

**Speech Recognition**

We use two speech recognition classes, this is due to how the words have different contexts depending on where you user is in the game e.g, in the main menu if the player says play it means the player wishes to start a game where as in the game scene play is used to unpause the game . The SpeechRegMenu class deals with the main menu and level difficulty scenes and SpeechRecognitionEngine deals with voice input in the game scene. I will explain both classes at the same time and will address differences where appropriate.

Both SpeechRegMenu and SpeechRecognitionEngine declare the similar variables.

public class SpeechRecgMenu : MonoBehaviour

{

// Variables

public string[] keywords = new string[] { "Play", "Settings", "Easy", "Average", "Hard", "Pause" };

public ConfidenceLevel confidence = ConfidenceLevel.Low;

public Text results;

protected PhraseRecognizer recognizer;

protected string word = "";

public class SpeechRecognitionEngine : MonoBehaviour

{

public string[] keywords = new string[] { "First", "Second", "Third", "Fourth", "Play", "Settings", "Pause", "Yes", "No"};

public ConfidenceLevel confidence = ConfidenceLevel.Low;

public float speed = 1;

public Text results;

public Image target;

public int scoreInt;

private string finalAnswer;

protected PhraseRecognizer recognizer;

protected string word = " ";

public GameObject canvas;

public GameObject answerPanel;

public GameObject pausePanel;

public GameObject outcomePanel;

private QuestionGenerator qGen;

Speech Recognition declares a number of Game Objects but they are essentially the same. The keyword array is the words that will be added to the recognizer as well as the confidence level, the confidence level to how certain the programme is that a keyword has been said. The Recognizer is initialized in the Start() method.

private void Start()

{

// intailizes Keyword Recognizer

if (keywords != null)

{

recognizer = new KeywordRecognizer(keywords, confidence);

recognizer.OnPhraseRecognized += Recognizer\_OnPhraseRecognized;

recognizer.Start();

}

}

When a Phrase is recognised, a variable is set with that word which is used to decide the action that takes place, it also sets a Result text field on the canvas so the user can see the word.

private void Recognizer\_OnPhraseRecognized(PhraseRecognizedEventArgs args)

{

word = args.text;

results.text = "You said: <b>" + word + "</b>";

Debug.Log(word);

}

In the Update method, A switch statement controls what happens. In the SpeechRegMenu cases in Play, Easy, Average and Hard. The level difficulty is also set here.

switch (word)

{

case "Play":

SceneManager.LoadScene("LevelDifficulty");

break;

case "Settings":

//Debug.Log("Settings");

//SceneManager.LoadScene("Settings");

break;

case "Easy":

SceneManager.LoadScene("GameScene");

Helper.difficulty = 1;

Debug.Log(Helper.difficulty);

break;

case "Average":

//Debug.Log("average");

SceneManager.LoadScene("GameScene");

Helper.difficulty = 2;

Debug.Log(Helper.difficulty);

break;

case "Hard":

// Debug.Log("hard");

SceneManager.LoadScene("GameScene");

Helper.difficulty = 3;

Debug.Log(Helper.difficulty);

break;

}

Similarly in SpeechRecognitionEngine a switch statement is used. This time the cases First, Second, Third and Fourth display the answerPanel and sets the final answer. Play continues a pause game and Pause sets the pausePanel to active. Yes calls the runCheck method and No returns the user from the to the game scene from the answerPanel.

switch (word)

{

case "First":

answerPanel.SetActive(true);

Debug.Log(qGen.ans1.text.ToString());

finalAnswer = qGen.ans1.text.ToString();

break;

case "Second":

answerPanel.SetActive(true);

Debug.Log(qGen.ans2.text.ToString());

finalAnswer = qGen.ans2.text.ToString();

break;

case "Third":

answerPanel.SetActive(true);

Debug.Log(qGen.ans3.text.ToString());

finalAnswer = qGen.ans3.text.ToString();

break;

case "Fourth":

answerPanel.SetActive(true);

Debug.Log(qGen.ans4.text.ToString());

finalAnswer = qGen.ans4.text.ToString();

break;

case "Play":

ContinueGame();

break;

case "Settings":

break;

case "Pause":

PauseGame();

break;

case "Yes":

if (answerPanel.activeSelf)

{

runCheck(finalAnswer);

}

break;

case "No":

answerPanel.SetActive(false);

break;

}

}

The runCheck method checks the answer, increment or decrement the score and or the guesses remaining.

private void runCheck(string ans)

{

answerPanel.SetActive(false);

outcomePanel.SetActive(true);

if (qGen.checkAnswer(ans))

{

// The answer was correct

qGen.scoreCount += 10;

qGen.nextQuestion(Helper.difficulty);

Invoke("", 1);

outcomePanel.SetActive(false);

}

else

{

qGen.remainingGuesses--;

Debug.Log("RemainingGuesses = "+ qGen.remainingGuesses);

Invoke("", 1);

outcomePanel.SetActive(false);

}

}

The Helper Class contains methods that are useful across the project.

public static class Helper

{

public static int difficulty;

public static GameObject pausemenuUI;

public static bool IsGamePaused = false;

// Used to shuffle correct answer with incorrect answers

public static void Shuffle<T>(this IList<T> list)

{

System.Random rng = new System.Random();

int n = list.Count;

while (n > 1)

{

n--;

int k = rng.Next(n + 1);

T value = list[k];

list[k] = list[n];

list[n] = value;

}

}

public static void RestartGame()

{

Application.LoadLevel(0);

}

public static string RemoveSpecialChars(this string data)

{

return HttpUtility.HtmlDecode(data);

}

}

# 

# 

# 

# Conclusion

Voice Recognition was not something familiar to us when we first took on this project. We feel we have gained more insight into voice recognition along with using Unity as our Engine to design the game in. We had both previously used unity in projects before but nothing along the lines of Voice Recognition and the game design we have created here.

Another great take away from this project was working with Json files as our storage. As frustrating as it is to run into errors for days on end it gave us a far greater understanding of how json serialization is done and how to get around some of the issues that might arise.

C# is also something both of us were not overly familiar with. Over the past few years most of our work has been done in languages such as Java, Python, JavaScript, C … etc. Coding in a language you are not overly confident with is always a productive learning curve and we feel we both gained better knowledge in the language on completion of this project.

In all we feel this project has helped us gain better understanding of Unity, Voice Recognition, C# and game development as a whole.

# Distribution of work.

The work in this project was well distributed between both of us. Sean dealt with the game mechanics such as answer checking, Scoring, Question loading and game progression. Eoghan dealt with speech recognition, all UI elements and sprite design.

# 