

Colour – Blindness Test

An Ishihara Test used to identify people with red-green colour blindness.

Ishihara Test Unity Program Architecture

The Ishihara test program files were made with three different scenes. First, the Persistent scene is responsible for storing the question-and-answer data. Next, the Menu Screen scene displays the instructions to the Ishihara test and contains a button which transitions to the Game scene. The Game scene contains four panels: the NewUserPanel, PatientIdPanel, QuestionPanel, and RoundOverPanel. The NewUserPanel enables the nurses to register the details of new patients into the system. After which, the patient ID number is entered in by the nurses in the PatientIdPanel to ensure that the results of the test are registered to the specific patient. Once this is done, the QuestionPanel displays the 17 multiple choice questions. Lastly, the RoundOverPanel saves the results to the MongoDB database when the Menu button is clicked. A simplified overview of the program architecture can be seen in Figure 1.

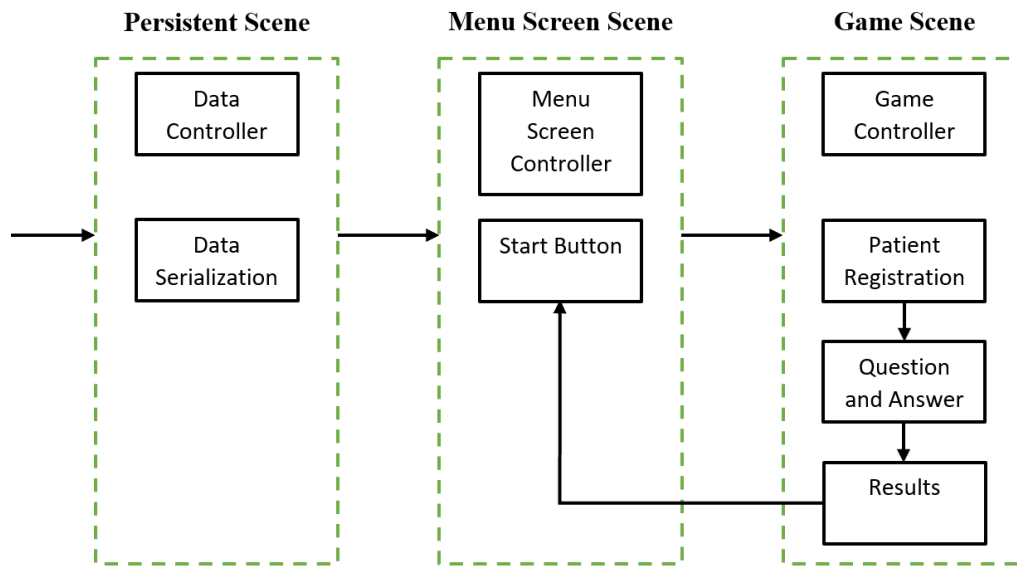


Figure 1: Ishihara Test Unity Program Architecture

Persistent Scene

Upon start-up, the program enters the Persistent scene. This scene contains the DataController gameobject with the DataController script attached to it. The relevant snippet of the C# script can be seen in Figure 2. This script helps transition to the MenuScreen scene using SceneManager; and stores the question and answers data. Moreover, the script prevents the DataController gameobject from being destroyed when transitioning to different scenes by allowing it to persist in a separate scene called DontDestroyOnLoad. This scene was designed this way to ensure that there is only one instance of the DataController when the program is running.

```

public RoundData[] allRoundData;

// Use this for initialization
void Start ()
{
    DontDestroyOnLoad (gameObject);

    SceneManager.LoadScene ("MenuScreen");
}

public RoundData GetCurrentRoundData()
{
    return allRoundData [0];
}

```

Figure 2: DataController Script Snippet

Under the Unity inspector, the questions and answers can be modified easily as shown in Figure 3. The program was designed this way to make handling a large amount of question data easy to configure. The relevant Booleans can also be assigned to different answers, which are used to tally the Ishihara test scores during the Game scene. This was achieved by using [System.Serializable] and linking the AnswerData, QuestionData and RoundData scripts.

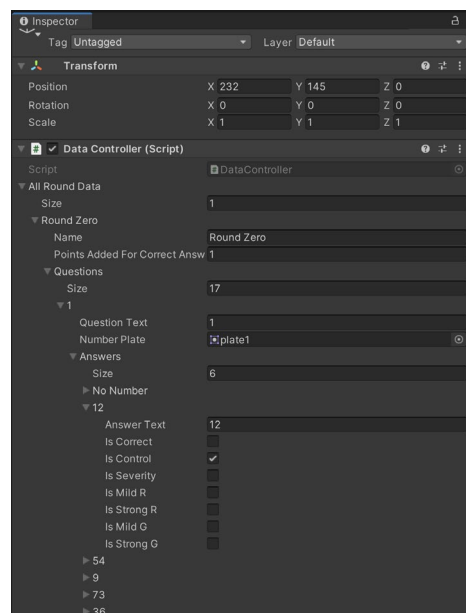


Figure 3: Inspector for DataController gameobject

The RoundData script enables the amount of points added when an answer is clicked to be set in the inspector (Figure 4). The QuestionData script enables the Ishihara plates to be assigned to each question (Figure 5). Lastly, the AnswerData allows the answer text options to be set (Figure 6). It also creates seven boolean values which are assigned to the answers in the following ways:

- i. isCorrect was assigned to correct answers
- ii. isControl was assigned to the correct answer for the demonstration plate
- iii. isSeverity was assigned to answers that indicated total colour blindness
- iv. isMildR assigned to answers that indicated red-weakness
- v. isStrongR was assigned to answers that indicated red-blindness
- vi. isMildG assigned to answers that indicated green-weakness
- vii. isStrongG was assigned to answers that indicated green-blindness

```
[System.Serializable]
public class RoundData
{
    public string name;
    public int pointsAddedForCorrectAnswer;
    public QuestionData[] questions;
}
```

Figure 4: RoundData Script Snippet

```
[System.Serializable]
public class QuestionData
{
    public string questionText;
    public UnityEngine.Sprite numberPlate;
    public AnswerData[] answers;
}
```

Figure 5: QuestionData Script Snippet

```
[System.Serializable]
public class AnswerData
{
    public string answerText;
    public bool isCorrect;
    public bool isControl;
    public bool isSeverity;
    public bool isMildR;
    public bool isStrongR;
    public bool isMildG;
    public bool isStrongG;
}
```

Figure 6: AnswerData Script Snippet

Menu Screen Scene

When the Ishihara test program is run, the scene changes to the Menu Screen scene almost instantaneously (Figure 7). This scene gives the instructions for the test; and contains the MenuScreenController gameobject with the MenuScreenController script attached to it. The start button was assigned the StartGame() OnClick function, which transitions to the Game scene when clicked.

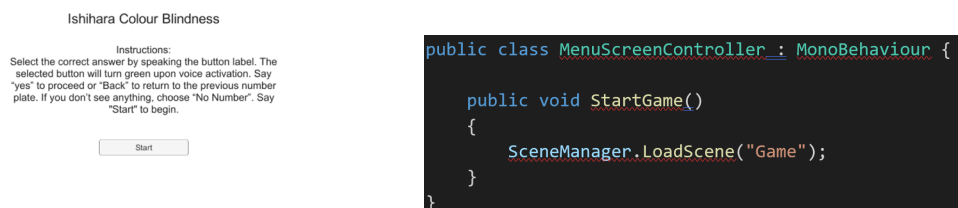
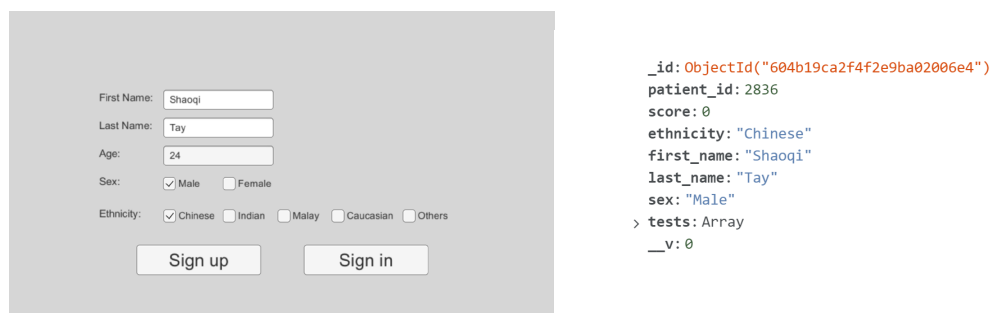


Figure 7: Menu Screen Scene (left) and MenuScreenController Script Snippet (right)

Game Scene

The first panel in the Game scene is the NewUserPanel (Figure 8). Its purpose is to enable the clinical staff to add new patients to the database. To do this, the patient's details such as name, age, sex, and ethnicity are recorded. When the "Sign Up" button is clicked, the patient's details are uploaded to MongoDB and a patient ID is assigned to them. Patients registered to the system will already have been assigned a patient ID number. In this case, the clinical staff would click the "Sign In" button instead.



The image shows a user registration form on the left and a JSON object on the right. The form has fields for First Name (Shaoqi), Last Name (Tay), Age (24), Sex (Male), and Ethnicity (Chinese). There are 'Sign up' and 'Sign in' buttons. The JSON object on the right represents the data stored in the database, including a unique _id, patient_id, score, ethnicity, first_name, last_name, sex, tests, and __v.

```
{
  "_id": "604b19ca2f4f2e9ba02006e4",
  "patient_id": 2836,
  "score": 0,
  "ethnicity": "Chinese",
  "first_name": "Shaoqi",
  "last_name": "Tay",
  "sex": "Male",
  "tests": Array,
  "__v": 0
}
```

Figure 8: NewUserPanel (left) and Patient ID Generation (right)

As shown in Figure 9, the SaveAndGoToPatientId() and the GoToPatientId() function is assigned to "Sign Up" and "Sign In" button respectively. Both these button transition to the PatientIdPanel. However, clicking the "Sign Up" button has the added function of converting the patient data into the JSON format through savePatient() and then uploading it to MongoDB in UploadPatient().

```

public void GoToPatientId()
{
    newUserDisplay.SetActive (false);
    idDisplay.SetActive (true);
}

0 references
public void SaveAndGoToPatientId()
{
    myFirstName = nameFirstInput.text;
    myLastName = nameLastInput.text;
    myAge = ageInput.text;
    CheckSexToggleOn();
    CheckEthnicityToggleOn();
    savePatient();
    UploadPatient();
    newUserDisplay.SetActive (false);
    idDisplay.SetActive (true);
}

```

Figure 9: Code Snippet for "Sign Up" and "Sign In" Button

In the PatientIdPanel, the patient ID is entered by the clinical staff and the “Start Test” button initiates the Ishihara test. It is crucial to ensure that the ID number corresponds to the patient, otherwise, the results will not be uploaded to the patient’s records at the end of the test. The EnterPatientId() function in Figure 10 is assigned to the “Start Test” button. It saves the patient ID number, activates the QuestionPanel, and starts the Ishihara test.

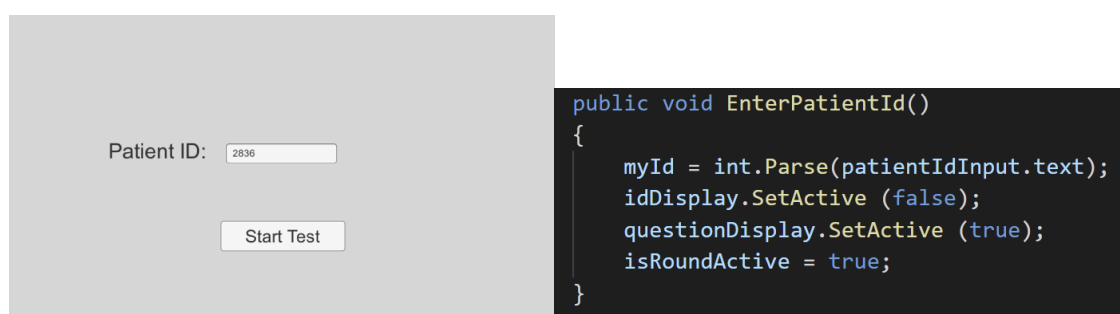


Figure 10: PatientIdPanel (left) and “Start Test” Button Code Snippet (right)

The QuestionPanel, displays the number plates and answer buttons. An example of the QuestionPanel can be seen in Figure 11.

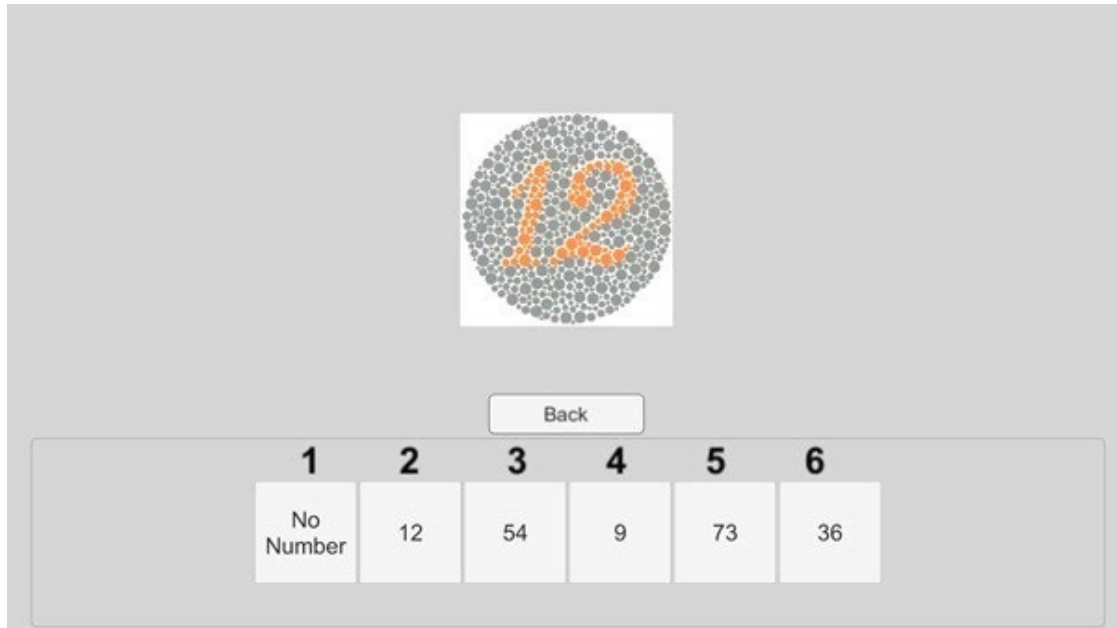


Figure 11: QuestionPanel

The answer buttons are stored in the `AnswerButtonAnswerPool` gameobject. It contains the `AnswerButton` prefab and `SimpleObjectPool` script that enables the answer buttons to be stored and displayed according to the question. The button used in `AnswerButton` prefab has the `HandleClick()` `OnClick` function assigned to it from the `AnswerButton` script (Figure 12). Whenever `HandleClick()` occurs, the Boolean values of the answer button is checked in sequence from `isControl` down to `isCorrect`. Since the test only goes to the next number plate when `isCorrect` is checked, `isCorrect` is arranged last to ensure that all the Booleans are checked before proceeding to the next question.

```

public void HandleClick()
{
    gameController.ControlButtonClicked (answerData.isControl);
    gameController.MildRButtonClicked (answerData.isMildR);
    gameController.StrongRButtonClicked (answerData.isStrongR);
    gameController.MildGButtonClicked (answerData.isMildG);
    gameController.StrongGButtonClicked (answerData.isStrongG);
    gameController.SeverityButtonClicked (answerData.isSeverity);
    gameController.AnswerButtonClicked (answerData.isCorrect);
}

```

Figure 12: AnswerButton Script Snippet

The Ishihara test is primarily controlled by the GameController gameobject with the GameController script attached to it. Each of the Boolean values are checked, and if they are true, 1 point is added to their respective scores. As shown in Figure 13, isCorrect is checked last. Once this happens, the next set of number plate and answer buttons are displayed with questionIndex++ and ShowQuestion(). When the test is completed, EndRound() occurs which sets the active panel to RoundOverPanel.

```

public void AnswerButtonClicked(bool isCorrect)
{
    if (isCorrect)
    {
        playerScore += currentRoundData.pointsAddedForCorrectAnswer;
        scoreDisplayText.text = "Main Score: " + playerScore + " / 16";
        controlScoreDisplayText.text = "Control Score: " + controlScore + " / 1";
        redWeakScoreDisplayText.text = "Red-weakness: " + redMildScore + " / 2";
        redStrongScoreDisplayText.text = "Red-blindness: " + redStrongScore + " / 2";
        greenWeakScoreDisplayText.text = "Green-weakness: " + greenMildScore + " / 2";
        greenStrongScoreDisplayText.text = "Green-blindness: " + greenStrongScore + " / 2";
        totalBlindnessScoreDisplayText.text = "Total Colour Blindness: " + totalBlindnessScore + " / 6";
    }

    if (questionPool.Length > questionIndex + 1)
    {
        questionIndex++;
        ShowQuestion ();
    }
    else
    {
        EndRound();
    }
}

```

Figure 13: GameController Script Snippet

The RoundOverPanel displays various scores (Figure 14). The Main Score is dependent on the user's ability to match the correct numbers with the number plates. It is used to assess if the patient has colour vision deficiencies. A Main Score of 12 and above indicated normal vision while a Main Score below that suggests colour blindness. The Control Score gives an indication of the result's reliability. Since the demonstration plate number can be seen by users with total colour blindness, a Control Score of 0 could mean that the patient was either fibbing or visually impaired to the extent that their results cannot be representative of their colour vision. The red and green scores are tested during the sixteenth and seventeenth plate to evaluate the severity of the user's colour blindness. Lastly, the Total Colour Blindness score was tallied according to the number of responses which exclusively indicated total colour blindness.

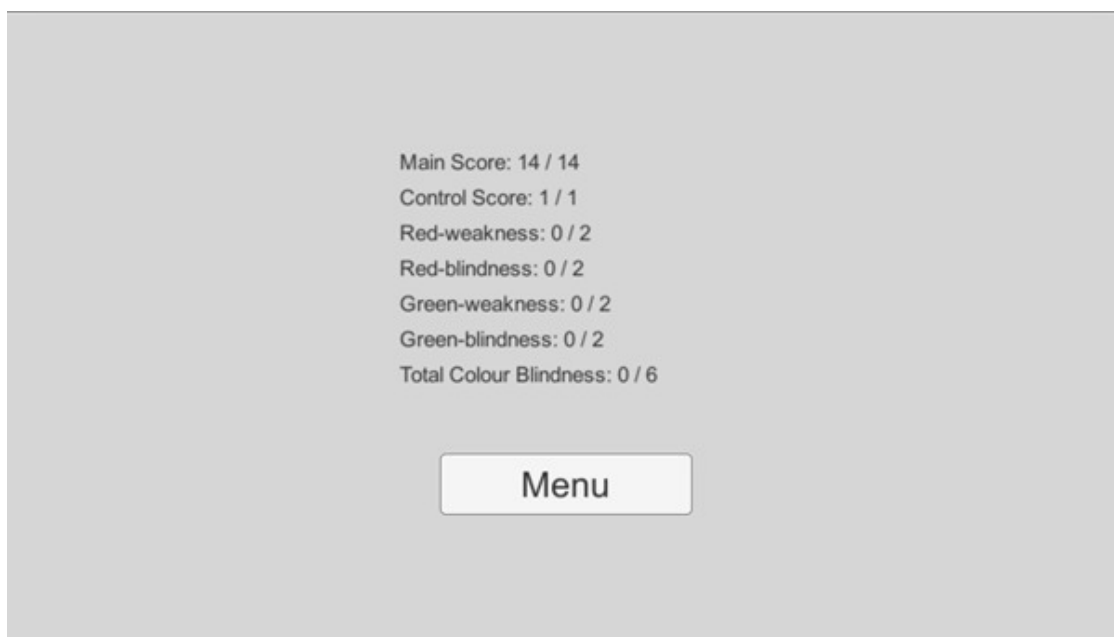


Figure 14: RoundOverPanel

Upon clicking the “Menu” button, the results are converted into JSON format using `saveIshiharaTest()` and subsequently uploaded to the MongoDB database using `UploadIshiharaTest()`. After which, the scene changes to the Menu Screen scene. An example of the Ishihara test results, as viewed in MongoDB, is presented in Figure 15.

```

public void ReturnToMenu()
{
    //Save();
    saveIshiharaTest();
    UploadIshiharaTest();

    SceneManager.LoadScene("MenuScreen");
}

```

```

_id: ObjectId("604b19ca2f4f2e9ba02006e4")
patient_id: 2836
score: 0
ethnicity: "Chinese"
first_name: "Shaoqi"
last_name: "Tay"
sex: "Male"
tests: Array
  0: Object
    test_name: "Ishihara Test"
    test_datetime: "12/03/21, 15:40"
    test_results: Array
      0: Object
        main_score: Object
          total: 14
          obtained: 14
        control_score: Object
          total: 1
          obtained: 1
        red_weakness_score: Object
          total: 2
          obtained: 0
        red_deficiency_score: Object
          total: 2
          obtained: 0
        green_weakness_score: Object
          total: 2
          obtained: 0
        green_deficiency_score: Object
          total: 2
          obtained: 0

```

Figure 15: "Menu" Button Code Snippet (left) and MongoDB Test Results (right)

Database Creation

There are two reasons for creating the database. Firstly, using a central database would enable the ophthalmologists to work efficiently as the test results are saved automatically into a cloud database to enable easy retrieval of patient data. Secondly, automatically saving results prevents errors from manifesting when results are recorded and saved manually by the ophthalmologists.

Two different methods of saving patient data were developed for this project. I developed a way to save the patient data locally in a specified folder as a TXT file. However, this method has since been replaced with another MongoDB database method. This method utilises JSON serialization to format the patient's information and send it to MongoDB using a POST function.

Local Saving

In the local saving method, the patient's data is formatted in the string "content" and stored by specifying the folder and file name in the string "path" (Figure 16). The patient's information can be accessed remotely if the specified file was sent to a cloud storage folder such as OneDrive. An example of a saved file can be seen in Figure 17.

```

private void Save()
{
    myId = int.Parse(patientIdInput.text);
    myFirstName = nameFirstInput.text;
    myLastName = nameLastInput.text;
    myAge = ageInput.text;
    CheckSexToggleOn();
    CheckEthnicityToggleOn();

    string path = "C:/Users/taysh/OneDrive/Desktop/GreenDeficiency/Results/" + myId + ".test.txt";
    if (!File.Exists(path))
    {
        File.WriteAllText(path, "Colour Blindness Test Results \n");
    }

    string content = DateTime.Now + "\n" + "Name: " + myFirstName + " " + myLastName + "\n" + "Age: " + myAge + "\n" + "Sex: " + mySex + "\n"
    + "Ethnicity: " + myEthnicity + "\n" + "Main Score: " + playerScore + "/14 \n" + "Control Score: " + controlScore + "/1 \n" + "Red Weakness Score: " +
    redMildScore + "/2 \n" + "Red Deficiency Score: " + redStrongScore + "/2 \n" + "Green Weakness Score: " + greenMildScore + "/2 \n"
    + "Green Deficiency Score: " + greenStrongScore + "/2 \n" + "Total Colour Blindness Score: " + totalBlindnessScore + "/6 \n";

    File.AppendAllText(path, content);
}

```

Figure 16: Local Saving Code Snippet

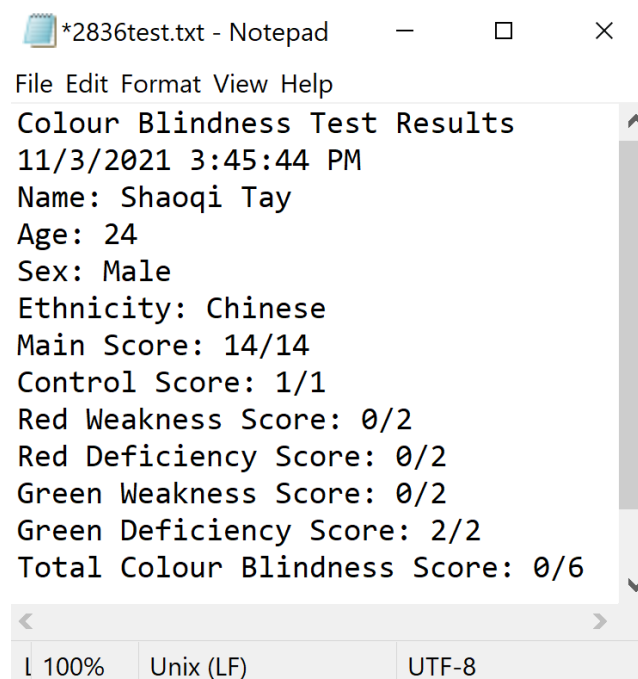


Figure 17: Example of Local Saving Method

MongoDB Database

The main reason which prompted the creation of the MongoDB database was that the local saving method would have been a hassle for nurses to query specific information in a patient's file. To reduce the workload of the clinical staff, the results needed to be saved in a more user-friendly interface.

The code snippet in Figure 18 enables the patient's personal details such as name, sex, ethnicity, and test results to be stored in MongoDB. This is accomplished by using UnityWebRequest to upload the "patientData" and "ishiharaData", which have been converted to the JSON format.

In the MongoDB database, the clinical staff would be able to view the information of multiple patients at once. Apart from that, the patient ID generated upon UploadPatient() is used to tag a patient's test results to that specific person. This prevents test results from being saved to the wrong patient.

```
IEnumerator Post(string url, string bodyJsonString)
{
    var request = new UnityWebRequest(url, "POST");
    byte[] bodyRaw = Encoding.UTF8.GetBytes(bodyJsonString);
    request.uploadHandler = (UploadHandler)new UploadHandlerRaw(bodyRaw);
    request.downloadHandler = (DownloadHandler)new DownloadHandlerBuffer();
    request.SetRequestHeader("Content-Type", "application/json");
    yield return request.SendWebRequest();

    Debug.Log("Status Code: " + request.responseCode);
}

1 reference
public void UploadPatient()
{
    StartCoroutine(Post("http://3.0.17.212:3000/create/patient", patientData));
}

1 reference
public void UploadIshiharaTest()
{
    StartCoroutine(Post("http://3.0.17.212:3000/add/test", ishiharaData));
}
```

Figure 18: MongoDB Database Code Snippet

Randomization of Ishihara Test Plates

The purpose of randomizing the order of the Ishihara test plates is to minimise the number of false negative errors, which could occur if patients memorised the standard order of the Ishihara number plates. Randomization of the order reduces the effectiveness of memorization and

ensures that a fair assessment of a patient's colour vision is conducted. The Shuffle() function accomplishes this by randomizing the order of the second to seventeenth plate and then saving the new randomised array into the original "questionPool" (Figure 19). Resultantly, the first demo plate is always displayed first, but the subsequent plates are randomised. The Shuffle() function is activated upon entering the Game scene. Since every test is completed by reverting to the Menu Screen scene, the order of the test plates is randomised for every test. In this regard, the iVESA's Ishihara test is superior to the traditional Ishihara booklet.

```
public void Shuffle()
{
    for (int i = 1; i < questionPool.Length; i++)
    {
        QuestionData tempPool = questionPool[i];
        int rnd = UnityEngine.Random.Range(1,i);
        questionPool[i] = questionPool[rnd];
        questionPool[rnd] = tempPool;
    }
}
```

Figure 19: Randomization of Ishihara Test Plates Code Snippet

Windows Speech Recognition

Windows Speech Recognition was implemented to provide an alternative method for participants to select the answer buttons. In the latest iteration, custom vocabulary was used to enable the answers to be selected by speaking either the number 1 to 6, or A to F. In the Unity program, the vocabulary for each of the six answer buttons can be edited in the Unity inspector for the GameController gameobject as shown in Figure 20.

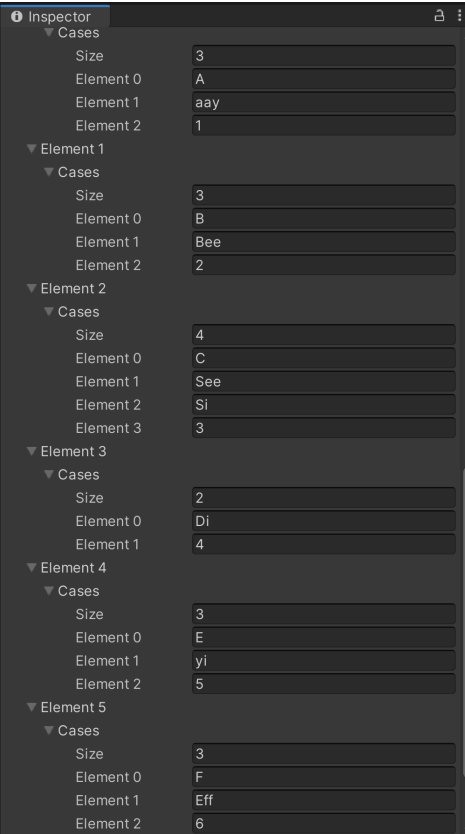


Figure 20: Unity Inspector for GameController gameobject