Figure 9 (a) Application module (b) Code snippet of update() (continued) (see online version for colours)

```
void Update() {
        time = Time.timeSinceLevelLoad;
        text.text = time + " seconds";
        bool trigger_pinch = false;
        HandModel hand_model = GetComponent<HandModel> ();
        Hand leap_hand = hand_model.GetLeapHand ();
        if (leap_hand == null)
            return;
        else if (leap_hand.PinchStrength == 1.0f) {
            // Scale trigger distance by thumb proximal bone length.
            Vector leap_thumb_tip = leap_hand.Fingers [0].TipPosition;
            float proximal_length = leap_hand.Fingers [0].Bone (Bone.BoneType.TYPE_PROXIMAL).Length;
            float trigger_distance = proximal_length * TRIGGER_DISTANCE_RATIO;
            // Check thumb tip distance to joints on all other fingers.
            // If it's close enough, start pinching.
            for (int i = 1; i < HandModel.NUM_FINGERS && !trigger_pinch; ++i) {
                Finger finger = leap_hand.Fingers [i];
                fin = i;
                for (int j = 0; j < FingerModel.NUM_BONES && !trigger_pinch; ++j) {
                    Vector leap_joint_position = finger.Bone ((Bone.BoneType)j).NextJoint;
                    if (leap_joint_position.DistanceTo (leap_thumb_tip) < trigger_distance)
                        trigger_pinch = true;
                }
            }
            Vector3 pinch_position = hand_model.fingers [0].GetTipPosition ();

            // Only change state if it's different.
            if (trigger_pinch && !pinching_1)
                OnPinch (fin);

            else if (!trigger_pinch && pinching_1)
                OnRelease ();
```

(b)

Note: This is a modification of the built-in script, MagneticPinch.

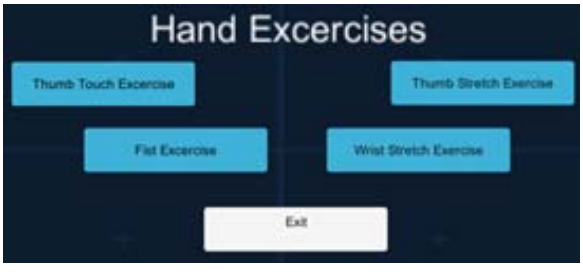
7 Technical description and results section

Games are designed in such a way that doctors can relate the duration of any session and the number of gestures made in that duration (that is, gestures in unit of time)

7.1 Thumb touch exercise

Figure 10(a) is the main menu screen and the start screen of the application. Hence, it is made as the first scene in the build settings. It is designed in the unity scene view. Linking to scenes (games) are done through buttons. Here is a screen shot [Figure 10(b)] of button script used to link main menu scene to the first exercise (thumb touch exercise).

Figure 10 (a) Start screen (b) Button script (see online version for colours)



(a)

Figure 10 (a) Start screen (b) Button script (continued) (see online version for colours)

```
using UnityEngine;
using System.Collections;

public class Firstb : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
    public void onClick(){
        Application.LoadLevel ("Pinch");
}
}

```

(b)

In Figure 10(b), Pinch is the name of the scene which contains the game for thumb touch exercise.

Figure 11 Selection of thumb touch exercise (see online version for colours)

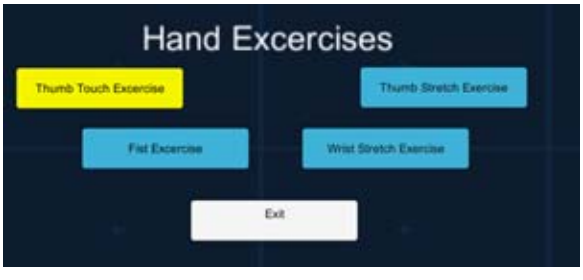
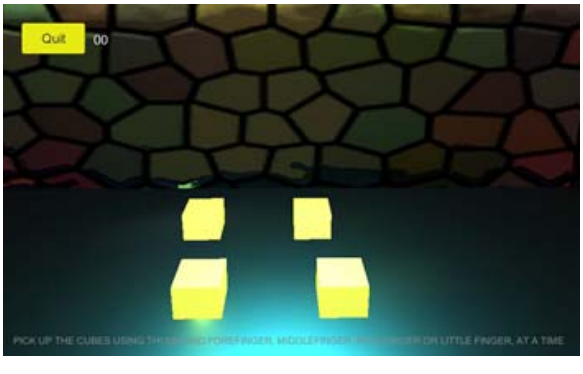


Figure 11 shows the result of hovering the cursor over thumb touch exercise button.

Figure 12 (a) Game begins (b) Script which makes the cubes revolve (see online version for colours)



(a)

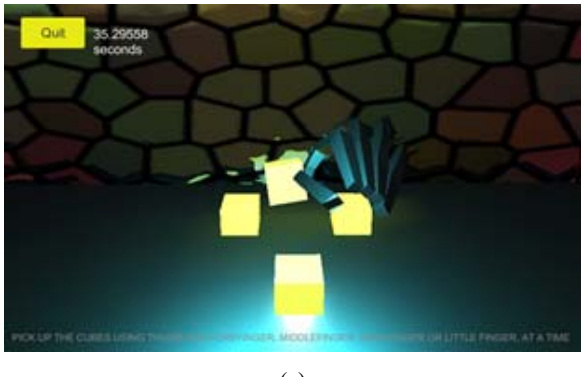
Figure 12 (a) Game begins (b) Script which makes the cubes revolve (continued) (see online version for colours)

```
using UnityEngine;
using System.Collections;

public class CubeMovement : MonoBehaviour {
    public Transform[] points;
    public float moveSpeed;
    protected int currentPoint;
    // Use this for initialization
    void Start () {
        transform.position = points [0].position;
        currentPoint = 0;
    }

    // Update is called once per frame
    void Update () {
        if (transform.position == points [currentPoint].position)
            currentPoint++;
        if (currentPoint >= points.Length)
            currentPoint = 0;

        transform.position = Vector3.MoveTowards (transform.position,
                                                  points [currentPoint].position,moveSpeed*Time.deltaTime);
        if (Input.GetKeyDown ("escape"))
            Application.LoadLevel ("ThumbOver");
    }
}

```

(b)

Figure 12(a) shows the start screen of the game for thumb touch exercise, after the patient clicks the first button (the button mentioned above). A timer runs whenever the patient does the exercise and pauses when leap does not detect hands. This is achieved by placing the code for running the timer, in the update() [refer the first two lines of the function shown in Figure 9(b)], which is called in per frame basis. The four cubes as seen in Figure 12(a) keep revolving on the perimeter of a square in the game. This is achieved through a script shown in Figure 12(b).

Figure 13 (a) Game ongoing (b) AddForce() to make cubes jump to hand (see online version for colours)



(a)

Figure 13 (a) Game ongoing (b) AddForce() to make cubes jump to hand (continued) (see online version for colours)

```
// Accelerate what we are grabbing toward the pinch.
if (grabbed_1 != null && fin == 1) {
    Vector3 distance = pinch_position - grabbed_1.transform.position;
    grabbed_1.GetComponent<Rigidbody> ().AddForce (forceSpringConstant * distance);
    c1++;
    if (c1 > 1000)
        Destroy (grabbed_1, 1.0f);
    Debug.Log ("1 Done! " + c1 + " ");
}
else if (grabbed_2 != null && fin == 2) {
    Vector3 distance = pinch_position - grabbed_2.transform.position;
    grabbed_2.GetComponent<Rigidbody> ().AddForce (forceSpringConstant * distance);
    c2++;
    if (c2 > 1000)
        Destroy (grabbed_2, 1.0f);
    Debug.Log ("2 Done!" + c2 + " " + Time.timeSinceLevelLoad + "");
    //GetComponent<TextMesh>().text = Time.timeSinceLevelLoad+"";
} else if (grabbed_3 != null && fin == 3) {
    Vector3 distance = pinch_position - grabbed_3.transform.position;
    grabbed_3.GetComponent<Rigidbody> ().AddForce (forceSpringConstant * distance);
    c3++;
    if (c3 > 1000)
        Destroy (grabbed_3, 1.0f);
    Debug.Log ("3 Done!" + c3 + " " + Time.timeSinceLevelLoad + "");
    //GetComponent<TextMesh>().text = Time.timeSinceLevelLoad+"";
} else if (grabbed_4 != null && fin == 4) {
    Vector3 distance = pinch_position - grabbed_4.transform.position;
    grabbed_4.GetComponent<Rigidbody> ().AddForce (forceSpringConstant * distance);
    c4++;
    if (c4 > 1000)
        Destroy (grabbed_4, 1.0f);
```

(b)

Figure 14 Quit button (see online version for colours)

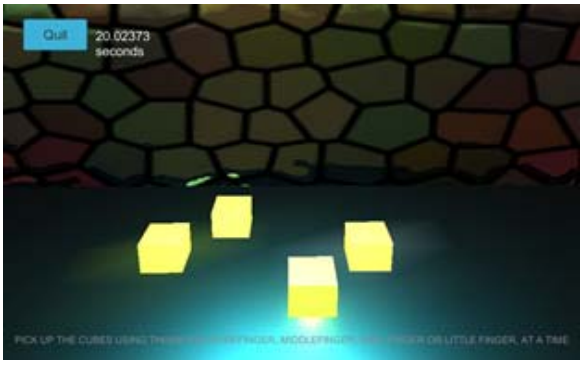


Figure 13(a) shows the game ongoing. As described in the figures, a timer runs when the sensor detects hands and the gestures are counted. When a gesture is detected, in this case, a pinch gesture for thumb touch exercise, the corresponding cube assigned for the particular finger, jumps to the point of contact between the touching fingers as shown in Figure 13(a). Figure 13(b) shows the code snippet which makes it possible. As shown in

Figure 13(b), AddForce() makes it possible for the corresponding cubes to jump to hand. After a total of 4000 frames which have detected pinch, the game automatically closes. This gives a break for over enthusiastic patients. A patient can also quit from the game anytime. Variables grabbed_1, grabbed_2, grabbed_3 and grabbed_4 are of type GameObject class. The cube objects are dragged and dropped (assigned to) onto these variables, in the inspector.

Figure 14 shows the screen when the patient hovers the cursor over quit button to end the game. This button can be clicked anytime during the game. On clicking the button, the screen shown in Figure 15 appears.

Figure 15 (a) Output screen (b) Modify values on screen (see online version for colours)

(a)

```
1 using UnityEngine;
2 using System.Collections;
3 using UnityEngine.UI;
4
5 public class ModifyText : MonoBehaviour {
6     public Text text,num;
7     // Use this for initialization
8     void Start () {
9         text.text = MagneticPinch.time.ToString()+" seconds";
10        num.text = MagneticPinch.num.ToString ();
11    }
12
13    // Update is called once per frame
14    void Update () {
15
16    }
17 }
```

(b)

Figure 15(a) shows the output screen which appears when the patient quits from the game. It contains the number of gestures performed and the duration of the game. On clicking the quit button, another function is also accomplished. Details about the completed session are appended to a text file named report. Figure 17(b) shows the script for file operations and Figure 15(b) shows the script which modifies values on output screen.

Figure 15(b) shows the script which modifies the score and duration panel on the output screen, of the completed game. These values are received from the public variables of the script which recognises gesture (MagneticPinch).