REFERENCES

[1] Ghobadi, S. et al. "Hand Segmentation using 2D/3D Image ", Proceedings of Image and Vision Computing New Zealand , Hamilton, New Zealand, 2007.

[2] Tiara V. Malloy, “Sign Language Use for Deaf, Hard of Hearing, And Hearing Babies”, The Evidence Supports It, American Society for Deaf Children, july 2003.

[3] World Health Organization website, www.who.int, the date of retrieval (April,2017)

[4] Ayman Hamed , Nahla A. Belal , Khaled M. Mahar, “Arabic sign language alphabet recognition based on HOG-PCA using Microsoft Kinect in complex Backgrounds”, College of Computing and Information Technology Arab Academy for Science, Technology, and Maritime Transport ,Alexandria, Egypt,2016.

[5] S. Obdrzalek, G. Kurillo, F. Ofli, R. Bajcsy, E. Seto, H. Jimison, and M. Pavel, “Accuracy and robustness of Kinect pose estimation in the context of coaching of elderly population,” in Proc. IEEE EMBC,2012, pp. 1188–1193.

[6] Microsoft Corporation. Human Interface Guidelines, 1.8 edition, 2013.

[7] Alex Smola and S.V.N. Vishwanathan, Introduction to Machine Learning, Departments of Statistics and Computer Science Purdue University –and– College of Engineering and Computer Science Australian National University.

[8] Gary Bradski and Adrian Kaehler , “ Learning OpenCV”, O'Reilly Media; 1st edition (October 4, 2008).

[9] Priyanka Lokhande, Riya Prajapati and Sandeep Pansare, “Data Gloves for Sign Language Recognition System AVCOE”, Sangamner, S.P.Pune University, 2015.

[10] M. AL-Rousan a, K. Assaleh b, A. Tala’a c, “Video-based signer-independent Arabic sign language recognition using hidden Markov models”,alsevier journal,united kingdom,2009

[11] M. F. Tolba, Ahmed Samir, Magdy Aboul-Ela, “Arabic sign language continuous sentences recognition using PCNN and graph matching”, Verlag London Limited,2012.

[12] K. Assaleh1, T. Shanableh, M. Fanaswala, F. Amin1, H. Bajaj, “Continuous Arabic Sign Language Recognition in User Dependent Mode”,2010.

[13] A.S.Elons, Menna Ahmed and Hwaidaa Shedid, “Facial Expressions Recognition for Arabic Sign Language Translation”, Faculty of Computer and information Sciences, Ain Shams University,2016.

[14] Noha A. Sarhan, Yasser EI-Sonbaty, Sherine M. Youssef , “HMM-Based Arabic Sign Language Recognition Using Kinect” , College of Engineering and Technology Arab Academy for Science and Technology, 2015.

[15] Microsoft Corporation. Kinect for Windows SDK, programming guide, 1.8 edition, 2013.

[16] Leo Breiman, “RANDOM FORESTS”, Statistics Department, University of California, Berkeley, CA 94720, January 2001.

[17] Jamie Shotton et al, “Real-Time Human Pose Recognition in Parts from Single Depth Images”, Microsoft Research Cambridge & Xbox Incubation,2011.

[18] Fukunaga, Keinosuke; Larry D. Hostetler, "The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition". IEEE Transactions on Information Theory. IEEE. Retrieved 2008.

[19] Eamonn Keogh & Chotirat Ann Ratanamahatana, “Knowledge and informations systems”, 2004.

APPENDIX

Appendix A: ksl.Demo

- **AssemblyInfo.cs** contains information about your assembly, like name, description, version, etc. You can find more details about its content reading the comments that are included in it.
- Properties\Resources.resx**Resources.Designer.cs** file, the auto-generated code from adding resources to the Project + Resources tab.

- **App.xaml** use to share a Resource Dictionary or a design-time data set with your entire application.

```
<Application x:Class="KinectSignLanguعبر.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  StartupUri="MainWindow.xaml">
<Application.Resources>

 </Application.Resources>
 </Application>
```

- **Application configuration (config.App) files** contain settings specific to an application. This file contains configuration settings that the common language runtime reads (such as assembly binding policy, remoting objects, and so on), and settings that the application can read.

```
        namespace KSL.Demo
               {
                using Microsoft.Kinect;

               public class KSLConfig
                {

public ColorImageFormat ColorImageFormat =
ColorImageFormat.RgbResolution640x480Fps30;

 public TransformSmoothParameters transformSmoothParameters = new
TransformSmoothParameters
        {
            Smoothing = 0.5f,
            Correction = 0.5f,
            Prediction = 0.5f,
            JitterRadius = 0.05f,
            MaxDeviationRadius = 0.04f
```

```
        };
    }
       }
```

- **MainWindow.xaml** describes things that relate only to the Main Window

```
<Window x: Class="KSL.Demo.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:kv="clr-
namespace:Microsoft.Samples.Kinect.WpfViewers;assembly=Microsoft.Samples.Kinect.WpfViewe"
        Title="Kinect Sign Langu عمر:: beta"
```

- **MainWindow.xaml.cs**

```
namespace KSL.Demo
{
    using KSL.Gestures.Classifier;
    using KSL.Gestures.Core;
    using KSL.Gestures.Logger;
    using KSL.Gestures.Segments;
    using Microsoft.Kinect;
    using Microsoft.Kinect.Toolkit;
    using Microsoft.Samples.Kinect.WpfViewers;
    using System;
    using System.Collections.Generic;
    using System.ComponentModel;
    using System.Speech.Synthesis;
    using System.Text;
    using System.Timers;
    using System.Windows;
    using System.Windows.Data;

    public partial class MainWindow : Window, INotifyPropertyChanged
    {
        SpeechSynthesizer voice = new SpeechSynthesizer();
        #region "Declaration"

        private KSLConfig config = new KSLConfig();
        private readonly KinectSensorChooser sensorChooser = new
KinectSensorChooser();
        private Skeleton[] skeletons = new Skeleton[0];
        private GesturesController gestureController;

        public event PropertyChangedEventHandler PropertyChanged;

        Classifier classifier = Classifier.getInstance;
        private Logger logger = Logger.getInstance;
```

```
        private string gestureSentence;
        private string gestureBuilder;
        Timer startStopTimer;
        private bool isNewSentence = false;

        #endregion

        #region "Constructor"

        public MainWindow()
        {
            DataContext = this;
            InitializeComponent();

            startStopTimer = new Timer(2000);
            startStopTimer.Elapsed += new
ElapsedEventHandler(startStopTimer_Elapsed);

            classifier.init();
            InitializeKinect();
        }

        #endregion

        #region "Initialize Kinect"

        private void InitializeKinect()
        {
            // initialize the Kinect sensor manعمر
            KinectSensorManager = new KinectSensorManager();
            KinectSensorManager.KinectSensorChanged += this.KinectSensorChanged;

            // locate an available sensor
            sensorChooser.Start();

            // bind chooser's sensor value to the local sensor manعمر
            var kinectSensorBinding = new Binding("Kinect") { Source =
this.sensorChooser };
            BindingOperations.SetBinding(this.KinectSensorManager,
KinectSensorManager.KinectSensorProperty, kinectSensorBinding);
        }

        private void KinectSensorChanged(object sender,
KinectSensorManagerEventArgs<KinectSensor> args)
        {
            if (null != args.OldValue)
                UninitializeKinectServices(args.OldValue);

            if (null != args.NewValue)
                InitializeKinectServices(KinectSensorManager, args.NewValue);
        }

        private void InitializeKinectServices(KinectSensorManager
KinectSensorManager, KinectSensor sensor)
        {
            KinectSensorManager.ColorFormat = config.ColorImageFormat;
            KinectSensorManager.ColorStreamEnabled = true;
```

```
            KinectSensorManager.DepthStreamEnabled = true;

            KinectSensorManager.TransformSmoothParameters =
config.transformSmoothParameters;

            sensor.SkeletonFrameReady += OnSkeletonFrameReady;
            KinectSensorManager.SkeletonStreamEnabled = true;

            KinectSensorManager.KinectSensorEnabled = true;

            if (!KinectSensorManager.KinectSensorAppConflict)
            {
                gestureController = new GesturesController();
                gestureController.GestureRecognized += OnGestureRecognized;

                RegisterGestures();
            }
        }

        private void UninitializeKinectServices(KinectSensor sensor)
        {
            sensor.SkeletonFrameReady -= OnSkeletonFrameReady;
            gestureController.GestureRecognized -= OnGestureRecognized;
        }

        private void OnSkeletonFrameReady(object sender,
SkeletonFrameReadyEventArgs e)
        {
            using (SkeletonFrame frame = e.OpenSkeletonFrame())
            {
                if (frame == null)
                    return;

                // resize the skeletons array if needed
                if (skeletons.Length != frame.SkeletonArrayLength)
                    skeletons = new Skeleton[frame.SkeletonArrayLength];

                // get the skeleton data
                frame.CopySkeletonDataTo(skeletons);

                foreach (var skeleton in skeletons)
                {
                    // skip the skeleton if it is not being tracked
                    if (skeleton.TrackingState != SkeletonTrackingState.Tracked)
                        continue;

                    // update the gesture controller
                    gestureController.UpdateAllGestures(skeleton);
                }
            }
        }

        public static readonly DependencyProperty KinectSensorManagerProperty =
DependencyProperty.Register
        (
            "KinectSensorManager",
            typeof(KinectSensorManager),
            typeof(MainWindow),
```

```
            new PropertyMetadata(null)
        );

        public KinectSensorManager KinectSensorManager
        {
            get { return GetValue(KinectSensorManagerProperty) as
KinectSensorManager; }
            set { SetValue(KinectSensorManagerProperty, value); }
        }

        #endregion

        #region "Gestures - Register, Recognize"

        private void RegisterGestures()
        {
            // Word: Hello
            IGesturesSegment[] helloSegments = new IGesturesSegment[2];
            HelloSegment1 helloSegment1 = new HelloSegment1();
            HelloSegment2 helloSegment2 = new HelloSegment2();
            helloSegments[0] = helloSegment1;
            helloSegments[1] = helloSegment2;
            gestureController.AddGesture("Hello", helloSegments);

            // Word: انت / انتr / انت're
            IGesturesSegment[] انتSegments = new IGesturesSegment[2];
            انتSegment1 انتSegment1 = new انتSegment1();
            انتSegment2 انتSegment2 = new انتSegment2();
            انتSegments[0] = انتSegment1;
            انتSegments[1] = انتSegment2;
            gestureController.AddGesture("انت", انتSegments);

            // Word: Name
            IGesturesSegment[] nameSegments = new IGesturesSegment[4];
            NameSegment1 nameSegment1 = new NameSegment1();
            NameSegment2 nameSegment2 = new NameSegment2();
            nameSegments[0] = nameSegment1;
            nameSegments[1] = nameSegment2;
            nameSegments[2] = nameSegment1;
            nameSegments[3] = nameSegment2;
            gestureController.AddGesture("Name", nameSegments);

            gestureController.AddGesture("Drive", driveSegments);


            // Word: ماذا
            IGesturesSegment[] ماذاSegments = new IGesturesSegment[10];
            ماذاSegment1 ماذاSegment1 = new ماذاSegment1();
            ماذاSegments[0] = ماذاSegment1;
            ماذاSegments[1] = ماذاSegment1;
            ماذاSegments[2] = ماذاSegment1;
            ماذاSegments[3] = ماذاSegment1;
            ماذاSegments[4] = ماذاSegment1;
            ماذاSegments[5] = ماذاSegment1;
            ماذاSegments[6] = ماذاSegment1;
            ماذاSegments[7] = ماذاSegment1;
            ماذاSegments[8] = ماذاSegment1;
            ماذاSegments[9] = ماذاSegment1;
```

```
        gestureController.AddGesture("ماذا", ماذاSegments);

        // Word: Hungry
        IGesturesSegment[] hungrySegments = new IGesturesSegment[3];
        HungrySegment1 hungrySegment1 = new HungrySegment1();
        HungrySegment2 hungrySegment2 = new HungrySegment2();
        HungrySegment3 hungrySegment3 = new HungrySegment3();
        hungrySegments[0] = hungrySegment1;
        hungrySegments[1] = hungrySegment2;
        hungrySegments[2] = hungrySegment3;
        gestureController.AddGesture("Hungry", hungrySegments);

        // Word: عمر / Old
        IGesturesSegment[] عمرSegments = new IGesturesSegment[2];
        عمرSegment1 عمرSegment1 = new عمرSegment1();
        عمرSegment2 عمرSegment2 = new عمرSegment2();
        عمرSegments[0] = عمرSegment1;
        عمرSegments[1] = عمرSegment2;
        gestureController.AddGesture("عمر", عمرSegments);


    }

    private void OnGestureRecognized(object sender, GesturesEventArgs e)
    {
        switch (e.GestureName)
        {
            case "Hello":
                display(WordsEnum.مرحبا);
                break;
            case "انت":
                display(WordsEnum.انت);
                break;
            case "Name":
                display(WordsEnum.اسم);
                break;

            case "عمر":
                display(WordsEnum.عمر);
                break;
            case "Hungry":
                display(WordsEnum.جائع);
                break;
            case "ماذا":
                display(WordsEnum.ماذا);
                break;
        }
    }

    #endregion

    #region "Display text"

    private void reset()
    {
        classifier.reset();
    }
```

```
        private void display(WordsEnum word)
        {
            classifier.addCode((int) word);
            string sentence = classifier.findSentence();
            List<int> sentenceBuilder = classifier.getSentenceBuilder();

            if (!isNewSentence)
            {
                if (sentenceBuilder.Count > 1 && !String.IsNullOrEmpty(sentence))
                {
                    isNewSentence = true;
                    startStopTimer.Start();
                    StringBuilder sb = new StringBuilder();
                    GestureBuilder = String.Empty;
                    foreach (int wordCode in sentenceBuilder)
                    {
                        sb.Append(Enum.GetName(typeof(WordsEnum), wordCode));
                        sb.Append(" • ");
                    }
                    sb.Length = sb.Length - 3;
                    GestureBuilder = sb.ToString();
                    GestureSentence = sentence;
                }
                else
                {
                    GestureSentence = String.Empty;
                    GestureBuilder = Enum.GetName(typeof(WordsEnum), word);
                   //voice.Speak(GestureBuilder);
                }
            }
        }

        public String GestureSentence
        {
            get { return gestureSentence; }

            private set
            {
                if (gestureSentence == value)
                    return;

                gestureSentence = value;

                if (this.PropertyChanged != null)
                    PropertyChanged(this, new
PropertyChangedEventArgs("GestureSentence"));
            }
        }

        public String GestureBuilder
        {
            get { return gestureBuilder; }

            private set
            {
                if (gestureBuilder == value)
                    return;
```

```
                gestureBuilder = value;

                if(this.PropertyChanged != null)
                    PropertyChanged(this, new
PropertyChangedEventArgs("GestureBuilder"));
            }
        }

        private void startStopTimer_Elapsed(object sender, ElapsedEventArgs e)
        {
            startStopTimer.Stop();
            this.isNewSentence = false;
        }

        #endregion
    }
}
```

Appendix B: ksl.Getures

B.1: Classifier

Use to make classification

```
namespace KSL.Gestures.Classifier
{
    using KSL.Gestures.Logger;
    using System;
    using System.Collections.Generic;
    using System.Linq;

    public sealed class Classifier
    {
        #region "Declarations"

        private static readonly Classifier instance = new Classifier();

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

        private int currentWordIndex = 0;

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

        private string foundSentence = String.Empty;

        private List<SentenceStructure> sentencesDictionary = new
List<SentenceStructure>();

        private bool areThereMatches = false;

        #endregion

        #region "Constructors"

        static Classifier() { }

        private Classifier() { }

        #endregion

        public static Classifier getInstance
        {
            get { return instance; }
        }

        public void init()
        {
            SentenceStructure s = new SentenceStructure


            s = new SentenceStructure
            {
                ID = 2,
```

```
                Codes = new List<int> { (int)WordsEnum.انت, (int)WordsEnum.اسم },
                Text = "اسمك؟ ما"
            };
            this.sentencesDictionary.Add(s);

            s = new SentenceStructure
            {
                ID = 4,
                Codes = new List<int> { (int)WordsEnum.عمر, (int)WordsEnum.انت },
                Text = "عمرك؟ كم"
            };
            this.sentencesDictionary.Add(s);

            s = new SentenceStructure
            {
                ID = 7,
                Codes = new List<int> { (int)WordsEnum.انت, (int)WordsEnum.جائع },
                Text = "جائع؟ انت هل"
            };
            this.sentencesDictionary.Add(s);
        }

        public void addCode(int wordCode)
        {
            if (this.sentenceBuilder.Count > 0 &&
this.sentenceBuilder[this.sentenceBuilder.Count - 1] == wordCode)
            {
                return;
            }

            this.sentenceBuilder.Add(wordCode);
        }

        public string findSentence()
        {
            this.foundSentence = String.Empty;

            if (this.sentenceBuilder.Count > 1)
            {

                for (int i = currentWordIndex; i < this.sentenceBuilder.Count; i
+= 1)
                {
                    areThereMatches = false;

                    foreach (SentenceStructure s in this.sentencesDictionary)
                    {
                        if (this.notMatchList.Contains(s.ID) ||
this.sentenceBuilder.Count > s.Codes.Count)
                            continue;

                        if (this.sentenceBuilder[i] != s.Codes[i])
                        {
                            this.notMatchList.Add(s.ID);
                            if (!areThereMatches)
                                areThereMatches = false;

                            continue;
```

```
                        }

                        if (this.sentenceBuilder.Count == s.Codes.Count &&
this.sentenceBuilder.SequenceEqual(s.Codes))
                        {
                            this.foundSentence = s.Text;
                            areThereMatches = true;

                            return this.foundSentence;
                        }

                        areThereMatches = true;
                    }

                    if (!this.areThereMatches)
                    {
                        this.foundSentence = String.Empty;
                        this.currentWordIndex = 0;
                        i = currentWordIndex;
                        int temp = sentenceBuilder[sentenceBuilder.Count - 1];
                        this.sentenceBuilder.Clear();
                        this.sentenceBuilder.Add(temp);
                        this.notMatchList.Clear();

                        return findSentence();
                    }

                    if (this.sentenceBuilder.Count > 0)
                        this.currentWordIndex += 1;

                    return String.Empty;
                }
            }

            return String.Empty;
        }

        public bool getAreThereMatches()
        {
            return this.areThereMatches;
        }

        public string getSentence()
        {
            return this.foundSentence;
        }

        public List<int> getSentenceBuilder()
        {
            List<int> temp = new List<int>(this.sentenceBuilder);

            if (!String.IsNullOrEmpty(this.foundSentence))
                this.clear();

            return temp;
        }

        private void clear()
```

```
        {
            this.sentenceBuilder.Clear();
            this.notMatchList.Clear();
            this.foundSentence = String.Empty;
            this.currentWordIndex = 0;
        }

        public void reset()
        {
            this.clear();
        }
    }
}
```

- **SentenceStructure**

```
namespace KSL.Gestures.Classifier
{
    using System.Collections.Generic;

    public class SentenceStructure
    {
        public int ID { set; get; }

        public List<int> Codes { set; get; }

        public string Text { set; get; }
    }
}
```

- **WordsEnum** is used to declare an enumeration, a distinct type that consists of a set of named constants

```
namespace KSL.Gestures.Classifier
{
    public enum WordsEnum
    {
        عمر = 100, // Also used as: "Old"
        مرحبا = 107,
        جائع = 108,
        اسم = 110,
        ماذا = 111,
        انت = 112 // Also used as: "انتr", "انت are", "Are انت"
    }
}
```

B.2: Core

- **Gesture.cs** used to make up the gesture

```
namespace KSL.Gestures.Core
{
    using Microsoft.Kinect;
    using System;

    class Gesture
    {
        private IGesturesSegment[] gestureParts; // the parts that make up the
gesture.

        private int currentGesturePart = 0; // the current gesture part that we
are matching against

        private int pausedFrameCount = 10; // the number of frames to pause for
when a pause is initiated

        private int frameCount = 0; // the current frame that we are on

        private bool paused = false; // are we paused?

        private string name; // the name of the gesture

        public event EventHandler<GesturesEventArgs> GestureRecognized; // occurs
when gesture recognized

        public Gesture(string name, IGesturesSegment[] gestureParts)
        {
            this.name = name;
            this.gestureParts = gestureParts;
        }

        public void UpdateGesture(Skeleton data)
        {
            if (this.paused)
            {
                if (this.frameCount == this.pausedFrameCount)
                {
                    this.paused = false;
                }

                this.frameCount += 1;
            }

            GesturePartResult result =
this.gestureParts[this.currentGesturePart].CheckGesture(data);

            if (result == GesturePartResult.Succeed)
            {
                if (this.currentGesturePart + 1 < this.gestureParts.Length)
                {
                    this.currentGesturePart += 1;
                    this.frameCount = 0;
```

```
                    this.pausedFrameCount = 10;
                    this.paused = true;
                }
                else
                {
                    if (this.GestureRecognized != null)
                    {
                        this.GestureRecognized(this, new
GesturesEventArgs(this.name, data.TrackingId));
                        this.Reset();
                    }
                }
            }
            else if (result == GesturePartResult.Fail || this.frameCount == 50)
            {
                this.Reset();
            }
            else
            {
                this.frameCount += 1;
                this.pausedFrameCount = 5;
                this.paused = true;
            }
        }

        public void Reset()
        {
            this.currentGesturePart = 0;
            this.frameCount = 0;
            this.pausedFrameCount = 5;
            this.paused = true;
        }

    }
}
```

- **GesturesController.cs** contain to make anew list gesture and recognized gestures and update all gestures

```
namespace KSL.Gestures.Core
{
    using Microsoft.Kinect;
    using System;
    using System.Collections.Generic;

    public class GesturesController
    {
        private List<Gesture> gestures = new List<Gesture>();

        public event EventHandler<GesturesEventArgs> GestureRecognized;

        public GesturesController() { }

        public void UpdateAllGestures(Skeleton data)
        {
            foreach (Gesture gesture in this.gestures)
```

```
            {
                gesture.UpdateGesture(data);
            }
        }

        public void AddGesture(string name, IGesturesSegment[] gestureDef)
        {
            Gesture gesture = new Gesture(name, gestureDef);
            gesture.GestureRecognized += onGestureRecognized;
            this.gestures.Add(gesture);
        }

        private void onGestureRecognized(object sender, GesturesEventArgs e)
        {
            if (this.GestureRecognized != null)
            {
                this.GestureRecognized(this, e);
            }

            foreach (Gesture g in this.gestures)
            {
                g.Reset();
            }
        }
    }
}
```

- **GesturePartResult**

```
namespace KSL.Gestures.Core
{
    public enum GesturePartResult
    {
        Fail,
        Succeed,
        Pausing
    }
}
```

- **GesturesEventArgs**

```
namespace KSL.Gestures.Core
{
    using System;

    public class GesturesEventArgs : EventArgs
    {
        /// <summary>
        /// Gets or sets the type of the gesture.
        /// </summary>
        public string GestureName { get; set; }

        /// <summary>
        /// Gets or sets the tracking ID.
```

```
        /// </summary>
        public int TrackignId { get; set; }

        public GesturesEventArgs(string name, int trackingId)
        {
            this.TrackignId = trackingId;
            this.GestureName = name;
        }
    }
}
```

- **IGesturesSegment**

```
namespace KSL.Gestures.Core
{
    using Microsoft.Kinect;
    using System;

    public interface IGesturesSegment
    {
        GesturePartResult CheckGesture(Skeleton skeleton);
    }
}
```

B.3:segment

- **Agesegment.cs**

```
namespace KSL.Gestures.Segments
{
    using KSL.Gestures.Core;
    using Microsoft.Kinect;

    public class عمرSegment1 : IGesturesSegment
    {
        public GesturePartResult CheckGesture(Skeleton skeleton)
        {
            if (skeleton.Joints[JointType.HandRight].Position.X >
skeleton.Joints[JointType.ShoulderLeft].Position.X &&
                skeleton.Joints[JointType.HandRight].Position.X <
skeleton.Joints[JointType.ShoulderRight].Position.X)
            {
                if (skeleton.Joints[JointType.HandRight].Position.Y <
skeleton.Joints[JointType.Head].Position.Y &&
                    skeleton.Joints[JointType.HandRight].Position.Y >
skeleton.Joints[JointType.ShoulderCenter].Position.Y)
                {
                    return GesturePartResult.Succeed;
                }
```

```
                return GesturePartResult.Pausing;
            }

            return GesturePartResult.Fail;
        }
    }

    public class عمرSegment2 : IGesturesSegment
    {
        public GesturePartResult CheckGesture(Skeleton skeleton)
        {
            if (skeleton.Joints[JointType.HandRight].Position.X >
skeleton.Joints[JointType.ShoulderLeft].Position.X &&
                skeleton.Joints[JointType.HandRight].Position.X <
skeleton.Joints[JointType.ShoulderRight].Position.X)
            {
                if (skeleton.Joints[JointType.HandRight].Position.Y <
skeleton.Joints[JointType.ShoulderCenter].Position.Y)
                {
                    return GesturePartResult.Succeed;
                }

                return GesturePartResult.Pausing;
            }

            return GesturePartResult.Fail;
        }
    }
}
```

- **HelloSegment.cs**

```
namespace KSL.Gestures.Segments
{
    using KSL.Gestures.Core;
    using Microsoft.Kinect;

    public class مرحباSegment1 : IGesturesSegment
    {
        public GesturePartResult CheckGesture(Skeleton skeleton)
        {
            // Right hand in form of head.
            if (skeleton.Joints[JointType.HandRight].Position.Z <
skeleton.Joints[JointType.Head].Position.Z)
            {
                // Right hand above shoulder center.
                if (skeleton.Joints[JointType.HandRight].Position.Y >
skeleton.Joints[JointType.ShoulderCenter].Position.Y)
                {
                    // Right hand left of right shoulder.
                    if (skeleton.Joints[JointType.HandRight].Position.X <
skeleton.Joints[JointType.ShoulderRight].Position.X)
                    {
                        return GesturePartResult.Succeed;
                    }
```

```
                    return GesturePartResult.Pausing;
                }

                return GesturePartResult.Fail;
            }

            return GesturePartResult.Fail;
        }
    }

    public class HelloSegment2 : IGesturesSegment
    {
        public GesturePartResult CheckGesture(Skeleton skeleton)
        {
            // Right hand in form of head.
            if (skeleton.Joints[JointType.HandRight].Position.Z <
skeleton.Joints[JointType.Head].Position.Z)
            {
                // Right hand above shoulder center.
                if (skeleton.Joints[JointType.HandRight].Position.Y >
skeleton.Joints[JointType.ShoulderCenter].Position.Y)
                {
                    // Right hand right of right shoulder.
                    if (skeleton.Joints[JointType.HandRight].Position.X >
skeleton.Joints[JointType.ShoulderRight].Position.X)
                    {
                        return GesturePartResult.Succeed;
                    }

                    return GesturePartResult.Pausing;
                }

                return GesturePartResult.Fail;
            }

            return GesturePartResult.Fail;
        }
    }
}
```

- **HungrySegment.cs**

```
namespace KSL.Gestures.Segments
{
    using KSL.Gestures.Core;
    using Microsoft.Kinect;
    using System;

    public class جائعSegment1 : IGesturesSegment
    {
        public GesturePartResult CheckGesture(Skeleton skeleton)
        {
```

```
            if (skeleton.Joints[JointType.HandRight].Position.X <
skeleton.Joints[JointType.ShoulderRight].Position.X &&
                skeleton.Joints[JointType.HandRight].Position.X >
skeleton.Joints[JointType.ShoulderLeft].Position.X &&
                skeleton.Joints[JointType.HandLeft].Position.X <
skeleton.Joints[JointType.HipLeft].Position.X)
            {
                if (skeleton.Joints[JointType.HandLeft].Position.Y <
skeleton.Joints[JointType.HipLeft].Position.Y)
                {
                    if (skeleton.Joints[JointType.HandRight].Position.Y >
skeleton.Joints[JointType.Spine].Position.Y)
                    {
                        return GesturePartResult.Succeed;
                    }

                    return GesturePartResult.Pausing;
                }

                return GesturePartResult.Fail;
            }

            return GesturePartResult.Fail;
        }
    }

    public class HungrySegment2 : IGesturesSegment
    {
        public GesturePartResult CheckGesture(Skeleton skeleton)
        {
            if (skeleton.Joints[JointType.HandRight].Position.X <
skeleton.Joints[JointType.ShoulderRight].Position.X &&
                skeleton.Joints[JointType.HandRight].Position.X >
skeleton.Joints[JointType.ShoulderLeft].Position.X &&
                skeleton.Joints[JointType.HandLeft].Position.X <
skeleton.Joints[JointType.ElbowLeft].Position.X)
            {
                if (skeleton.Joints[JointType.HandRight].Position.Y <
skeleton.Joints[JointType.Spine].Position.Y &&
                    skeleton.Joints[JointType.HandLeft].Position.Y <
skeleton.Joints[JointType.HipLeft].Position.Y)
                {
                    if (skeleton.Joints[JointType.HandRight].Position.X <
skeleton.Joints[JointType.Spine].Position.X)
                    {
                        return GesturePartResult.Succeed;
                    }

                    return GesturePartResult.Pausing;
                }

                return GesturePartResult.Fail;
            }

            return GesturePartResult.Fail;
        }
    }
```

```
    public class HungrySegment3 : IGesturesSegment
    {
        public GesturePartResult CheckGesture(Skeleton skeleton)
        {
            if (skeleton.Joints[JointType.HandRight].Position.X <
skeleton.Joints[JointType.ShoulderRight].Position.X &&
                skeleton.Joints[JointType.HandRight].Position.X >
skeleton.Joints[JointType.ShoulderLeft].Position.X &&
                skeleton.Joints[JointType.HandLeft].Position.X <
skeleton.Joints[JointType.ElbowLeft].Position.X)
            {
                if (skeleton.Joints[JointType.HandRight].Position.Y <
skeleton.Joints[JointType.Spine].Position.Y &&
                    skeleton.Joints[JointType.HandLeft].Position.Y <
skeleton.Joints[JointType.HipLeft].Position.Y)
                {
                    if (skeleton.Joints[JointType.HandRight].Position.X >
skeleton.Joints[JointType.Spine].Position.X)
                    {
                        return GesturePartResult.Succeed;
                    }

                    return GesturePartResult.Pausing;
                }

                return GesturePartResult.Fail;
            }

            return GesturePartResult.Fail;
        }
    }
}
```

- **NameSegment.cs**

```
namespace KSL.Gestures.Segments
{
    using KSL.Gestures.Core;
    using Microsoft.Kinect;

    public class اسمSegment1 : IGesturesSegment
    {
        public GesturePartResult CheckGesture(Skeleton skeleton)
        {
            if (skeleton.Joints[JointType.HandRight].Position.X >
skeleton.Joints[JointType.ElbowLeft].Position.X &&
                skeleton.Joints[JointType.HandRight].Position.X <
skeleton.Joints[JointType.ElbowRight].Position.X &&
                skeleton.Joints[JointType.HandLeft].Position.X >
skeleton.Joints[JointType.ElbowLeft].Position.X &&
                skeleton.Joints[JointType.HandLeft].Position.X <
skeleton.Joints[JointType.ElbowRight].Position.X &&
                skeleton.Joints[JointType.HandRight].Position.Y <
skeleton.Joints[JointType.ShoulderCenter].Position.Y)
            {
```

```
                if (skeleton.Joints[JointType.HandLeft].Position.Y <
skeleton.Joints[JointType.Spine].Position.Y)
                {
                    if (skeleton.Joints[JointType.HandRight].Position.Y <
skeleton.Joints[JointType.Spine].Position.Y)
                    {
                        return GesturePartResult.Succeed;
                    }

                    return GesturePartResult.Pausing;
                }

                return GesturePartResult.Fail;
            }

            return GesturePartResult.Fail;
        }
    }

    public class NameSegment2 : IGesturesSegment
    {
        public GesturePartResult CheckGesture(Skeleton skeleton)
        {
            if (skeleton.Joints[JointType.HandRight].Position.X >
skeleton.Joints[JointType.ElbowLeft].Position.X &&
                skeleton.Joints[JointType.HandRight].Position.X <
skeleton.Joints[JointType.ElbowRight].Position.X &&
                skeleton.Joints[JointType.HandLeft].Position.X >
skeleton.Joints[JointType.ElbowLeft].Position.X &&
                skeleton.Joints[JointType.HandLeft].Position.X <
skeleton.Joints[JointType.ElbowRight].Position.X &&
                skeleton.Joints[JointType.HandRight].Position.Y <
skeleton.Joints[JointType.ShoulderCenter].Position.Y)
            {
                if (skeleton.Joints[JointType.HandLeft].Position.Y <
skeleton.Joints[JointType.Spine].Position.Y)
                {
                    if (skeleton.Joints[JointType.HandRight].Position.Y >
skeleton.Joints[JointType.Spine].Position.Y)
                    {
                        return GesturePartResult.Succeed;
                    }

                    return GesturePartResult.Pausing;
                }

                return GesturePartResult.Fail;
            }

            return GesturePartResult.Fail;
        }
    }
       }
```

- **whatSegment.cs**

```
namespace KSL.Gestures.Segments
{
    using KSL.Gestures.Core;
    using Microsoft.Kinect;
    using System;

    public class ماذاSegment1 : IGesturesSegment
    {
        public GesturePartResult CheckGesture(Skeleton skeleton)
        {
            if (skeleton.Joints[JointType.HandLeft].Position.X <
skeleton.Joints[JointType.ShoulderLeft].Position.X &&
                skeleton.Joints[JointType.HandRight].Position.X >
skeleton.Joints[JointType.ShoulderRight].Position.X)
            {
                if (skeleton.Joints[JointType.HandLeft].Position.Y >
skeleton.Joints[JointType.ShoulderLeft].Position.Y &&
                    skeleton.Joints[JointType.HandRight].Position.Y >
skeleton.Joints[JointType.ShoulderRight].Position.Y)
                {
                    if (skeleton.Joints[JointType.HandLeft].Position.Y >
skeleton.Joints[JointType.ElbowLeft].Position.Y &&
                        skeleton.Joints[JointType.HandRight].Position.Y >
skeleton.Joints[JointType.ElbowRight].Position.Y)
                    {
                        return GesturePartResult.Succeed;
                    }

                    return GesturePartResult.Pausing;
                }

                return GesturePartResult.Fail;
            }

            return GesturePartResult.Fail;
        }
    }
}
```

- **youSegment.cs**

```
namespace KSL.Gestures.Segments
{
    using KSL.Gestures.Core;
    using Microsoft.Kinect;

    public class انتSegment1 : IGesturesSegment
    {
        public GesturePartResult CheckGesture(Skeleton skeleton)
        {
```

```
            if (skeleton.Joints[JointType.HandRight].Position.X >
skeleton.Joints[JointType.ElbowLeft].Position.X &&
                skeleton.Joints[JointType.HandRight].Position.X <
skeleton.Joints[JointType.ElbowRight].Position.X)
            {
                if (skeleton.Joints[JointType.HandRight].Position.Y <
skeleton.Joints[JointType.ShoulderCenter].Position.Y &&
                    skeleton.Joints[JointType.HandRight].Position.Y >
skeleton.Joints[JointType.HipCenter].Position.Y)
                {

                    if (skeleton.Joints[JointType.HandRight].Position.Z >
skeleton.Joints[JointType.ElbowRight].Position.Z)
                    {
                        return GesturePartResult.Succeed;
                    }
```

- **Logger.cs**

```
namespace KSL.Gestures.Logger
{
    using System;
    using System.IO;

    public sealed class Logger
    {
        private static string filePath = @"D:\log.txt";

        private static FileMode fileMode = FileMode.Append;

        private static FileAccess fileAccess = FileAccess.Write;

        private static readonly Logger instance = new Logger();

        static Logger() { }

        private Logger() { }

        public static Logger getInstance
        {
            get { return instance; }
        }

        public void logMessعمر(string messعمر, errorFlag flag)
        {
            using (FileStream fs = new FileStream(filePath, fileMode,
fileAccess))
            using (StreamWriter sw = new StreamWriter(fs))
            {
                string text = String.Format("[{0}] {1} {2}", DateTime.Now,
getErrorLogFlag(flag), messعمر);
                sw.WriteLine(text);
            }
        }
```

```
    private static string getErrorLogFlag(errorFlag flag)
    {
        string errorDesc = String.Empty;

        switch (flag)
        {
            case errorFlag.SentenceBuilderState:
                errorDesc = "Sentence builder state:";
                break;
            case errorFlag.SentenceDetected:
                errorDesc = "Detected:";
                break;
            case errorFlag.WordDetected:
                errorDesc = "Detected:";
                break;
            case errorFlag.WordRemove:
                errorDesc = "Removed:";
                break;
            case errorFlag.WordAdded:
                errorDesc = "Added:";
                break;
            default:
                errorDesc = "Uknown:";
                break;
        }

        return errorDesc;
    }
}

public enum errorFlag
{
    WordDetected,
    SentenceDetected,
    SentenceBuilderState,
    WordRemove,
    WordAdded
}
   }
```

Appendix C: ksl.Tests

- **ClassifierTests.cs**

```
namespace KSL.Tests
{
    using KSL.Gestures.Classifier;
    using Microsoft.VisualStudio.TestTools.UnitTesting;
    using System;
    using System.Collections.Generic;

    [TestClass]
    public class ClassifierTests
    {
        Classifier classifier = Classifier.getInstance;
        private List<int> testList = new List<int>();
        private string sentence = String.Empty;

        [TestInitialize]
        public void init()
        {
            classifier.init();
        }

        [TestMethod]
        public void ItShouldBeEmptySentence()
        {
            classifier.addCode((int) WordsEnum.عمر);
            sentence = classifier.findSentence();
            testList = classifier.getSentenceBuilder();

            Assert.AreEqual(String.Empty, sentence);
        }

        [TestMethod]
        public void InputAndOutputListsSouldBeEqual()
        {
            classifier.addCode((int)WordsEnum.انت);
            sentence = classifier.findSentence();
            testList = classifier.getSentenceBuilder();

            classifier.addCode((int)WordsEnum.ماذا);
            sentence = classifier.findSentence();
            testList = classifier.getSentenceBuilder();

            classifier.addCode((int)WordsEnum.مرحبا);
            sentence = classifier.findSentence();
            testList = classifier.getSentenceBuilder();

            classifier.addCode((int)WordsEnum.عمر);
            sentence = classifier.findSentence();
            testList = classifier.getSentenceBuilder();

            classifier.addCode((int)WordsEnum.انت);
            sentence = classifier.findSentence();
            testList = classifier.getSentenceBuilder();
```

```
    classifier.addCode((int)WordsEnum.انت);
    sentence = classifier.findSentence();
    testList = classifier.getSentenceBuilder();

    classifier.addCode((int)WordsEnum.اسم);
    sentence = classifier.findSentence();
    testList = classifier.getSentenceBuilder();

    List<int> expectedList = new List<int>
    {
        (int) WordsEnum.انت, (int) WordsEnum.اسم
    };

    CollectionAssert.AreEqual(expectedList, testList);
}

[TestMethod]
public void ItShouldReturnHowOldAreانت()
{
    classifier.addCode((int)WordsEnum.انت);
    sentence = classifier.findSentence();
    testList = classifier.getSentenceBuilder();

    classifier.addCode((int)WordsEnum.ماذا);
    sentence = classifier.findSentence();
    testList = classifier.getSentenceBuilder();

    classifier.addCode((int)WordsEnum.مرحبا);
    sentence = classifier.findSentence();
    testList = classifier.getSentenceBuilder();

    classifier.addCode((int) WordsEnum.عمر);
    sentence = classifier.findSentence();
    testList = classifier.getSentenceBuilder();

    classifier.addCode((int) WordsEnum.انت);
    sentence = classifier.findSentence();
    testList = classifier.getSentenceBuilder();

    StringAssert.Equals("How old are انت?", sentence);
}

[TestMethod]
public void ItShouldReturnHowOldAreانت2()
{
    classifier.addCode((int)WordsEnum.مرحبا);

    testList = classifier.getSentenceBuilder();


    classifier.addCode((int)WordsEnum.انت);
    sentence = classifier.findSentence();

    testList = classifier.getSentenceBuilder();
```

```
        classifier.addCode((int)WordsEnum.عمر);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.انت);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.مرحبا);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.عمر);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.عمر);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.انت);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        StringAssert.Equals("How old are انت?", sentence);
    }

    [TestMethod]
    public void ItShouldReturnAreانتHungry()
    {
        classifier.addCode((int)WordsEnum.مرحبا);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();


        classifier.addCode((int)WordsEnum.انت);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();


        classifier.addCode((int)WordsEnum.عمر);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.انت);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();
```

```
        classifier.addCode((int)WordsEnum.مرحبا);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.عمر);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.جائع);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.جائع);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.انت);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.جائع);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        StringAssert.Equals("Are انت hungry?", sentence);
    }

    [TestMethod]
    public void ItShouldReturnماذاDoانتWantToEat()
    {
        classifier.addCode((int)WordsEnum.مرحبا);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();



        classifier.addCode((int)WordsEnum.انت);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.عمر);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();

        classifier.addCode((int)WordsEnum.انت);
        sentence = classifier.findSentence();

        testList = classifier.getSentenceBuilder();
```

```
                classifier.addCode((int)WordsEnum.مرحبا);
                sentence = classifier.findSentence();

                testList = classifier.getSentenceBuilder();

                classifier.addCode((int)WordsEnum.عمر);
                sentence = classifier.findSentence();

                testList = classifier.getSentenceBuilder();

                classifier.addCode((int)WordsEnum.جائع);
                sentence = classifier.findSentence();

                testList = classifier.getSentenceBuilder();

                classifier.addCode((int)WordsEnum.جائع);
                sentence = classifier.findSentence();

                testList = classifier.getSentenceBuilder();

                classifier.addCode((int)WordsEnum.انت);
                sentence = classifier.findSentence();

                testList = classifier.getSentenceBuilder();

                classifier.addCode((int)WordsEnum.جائع);
                sentence = classifier.findSentence();

                testList = classifier.getSentenceBuilder();

                classifier.addCode((int)WordsEnum.ماذا);
                sentence = classifier.findSentence();

                testList = classifier.getSentenceBuilder();

                classifier.addCode((int)WordsEnum.انت);
                sentence = classifier.findSentence();

                testList = classifier.getSentenceBuilder();


            }
        }
    }
```