|  |
| --- |
|  |
| SticKart |
| Journal |
|  |
| **Keith Cully** |

# Table of Contents:

[Table of Contents: 1](#_Toc358640356)

[Learning Outcomes 2](#_Toc358640357)

[Summary of Learning Outcomes: 2](#_Toc358640358)

[Sprint 4 Learning Outcomes (LO): 2](#_Toc358640359)

[LO: Connect a Windows Communication Foundation (WCF) data service to a SQL database, on an Azure hosted SQL server 2](#_Toc358640360)

[LO: Use LINQ to compose queries for a WCF data service 5](#_Toc358640361)

[LO: Set up SSL encryption on a WCF data service, hosted on an Azure cloud service 6](#_Toc358640362)

[LO: Force a .NET client application to accept self-signed certificates, for SSL connections 9](#_Toc358640363)

[LO: Create and configure service operations on a WCF data service 10](#_Toc358640364)

[Sprint 3 Learning Outcomes (LO): 12](#_Toc358640365)

[LO: Implement a thread safe notification system, using the singleton design pattern, in C# 12](#_Toc358640366)

[LO: How to implement the IComparer and IComparable classes in C#, to allow for easy sorting of custom objects 14](#_Toc358640367)

[Sprint 2 Learning Outcomes (LO): 16](#_Toc358640368)

[LO: Adjust the Kinect sensor's angle to ensure the player is being tracked correctly 16](#_Toc358640369)

[LO: Track a user's hand position smoothly, consistently and accurately 18](#_Toc358640370)

[LO: Create a motor driven entity using multiple bodies and joints using the Farseer physics engine 19](#_Toc358640371)

[LO: Implement procedural population of verbal commands for use in speech detection 21](#_Toc358640372)

[LO: Detect run and jump actions from Kinect input using gestures 24](#_Toc358640373)

[Sprint 1 Learning Outcomes (LO): 26](#_Toc358640374)

[LO: Use the Kinect sensor as an input device 26](#_Toc358640375)

[LO: Implement gesture tracking using the Kinect sensor 27](#_Toc358640376)

[LO: Create an extendable NUI-friendly menu system 28](#_Toc358640377)

[LO: Configure Visual Studio to create an installer package which will download and install custom dependencies 31](#_Toc358640378)

[LO: Implement voice commands based on the Kinect's audio stream 32](#_Toc358640379)

# Learning Outcomes

## Summary of Learning Outcomes:

<https://docs.google.com/spreadsheet/ccc?key=0AvF8UK7GqSIjdGNhZTJKb2NVZldzN0JOTThqZGhJd2c#gid=0>

## Sprint 4 Learning Outcomes (LO):

### LO: Connect a Windows Communication Foundation (WCF) data service to a SQL database, on an Azure hosted SQL server

**Goal:**

Connect a WCF data service to an Azure hosted SQL instance, so that the data service can access the high score data in the database.

**Problems:**

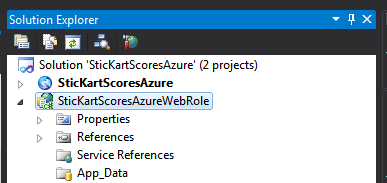
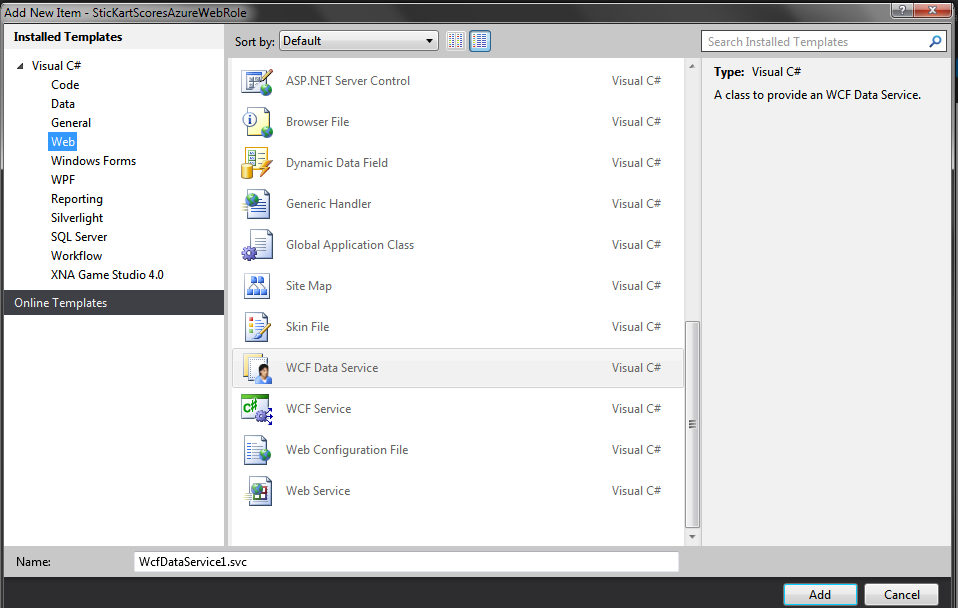
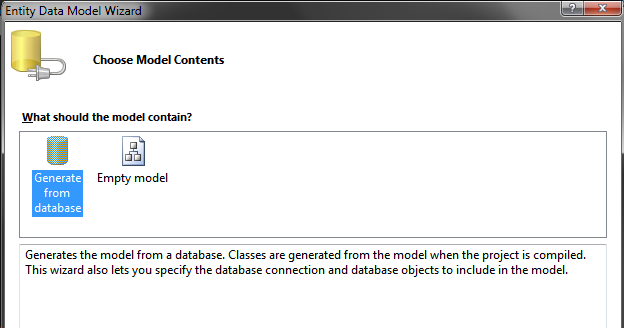
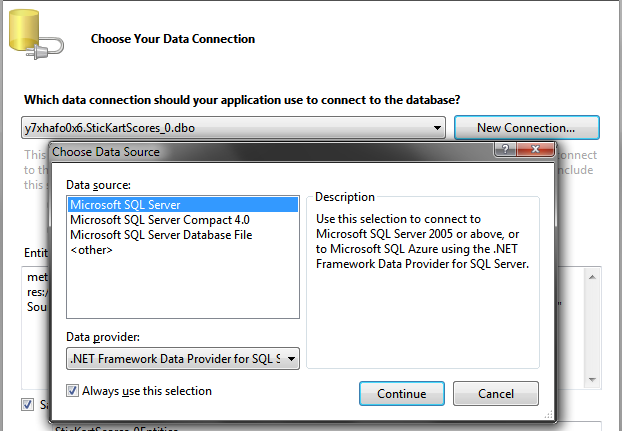
1. I needed to research how to configure the WCF service to connect to the SQL instance.
2. The login details for the SQL server would need to be encrypted.
3. The WCF service would need adequate permissions on the server hosting it.

**Solutions:**

After researching connecting a WCF data service, hosted on a cloud service, to an Azure SQL instance; I came across the following useful websites: <http://msdn.microsoft.com/en-us/library/ee621789.aspx> and <http://msftdbprodsamples.codeplex.com/releases/view/101644>.

For this explanation, I am assuming you already have a SQL instance running on Azure, have the Windows Azure SDK installed and have a base cloud service project created in Visual Studio 2010.

Connecting to the SQL Server:

1. In your cloud service project delete the default IService1.cs and Service1.svc files.
2. From solution explorer, right-click the web role in the base project and select Add->New Item.  
      
   
3. From the dialogue; select Web->WCF Data Service, give it a name and click Add.  
     
   
4. Right-click the web role again and select Add->New Item.
5. This time select Data->ADO.NET Entity Data Model, give it a name and click add.
6. From the dialogue that pops up select Generate from database then click next.  
     
   
7. Now select New Connection->Microsoft SQL Server then click Continue.  
     
   
8. Enter your SQL server connection details to establish a connection. These should take the form: (You can retrieve your server name from the Azure management portal at manage.windowsazure.com)
   1. **Server Name**: {servername}.database.windows.net
   2. **Use SQL Server Authentication**: Yes
   3. **User Name**: {Azure SQL Database UserName}
   4. **Password**: {Azure SQL Database Password}
   5. **Connect to a database** > **Select or enter a database name**: {YourDatabaseName}
9. Select yes, include the sensitive data in the connection string. This will be encrypted later.
10. Change the Save entity connection settings in Web.Config name to “<ARelevantName>\_Entities” and click next.
11. Select whatever tables you want the data service to have access to then click finish.
12. Open the file <YourDataServiceName>.svc.cs and change the section inside the brackets saying < /\* TODO: put your data source class name here \*/ > to the entities name you set in step 10.
13. If you encounter a build error when trying to build the project now, remove the reference to System.Data.Services.Client.dll.

Encrypting the connection string for testing on localhost:

1. Make a backup of your project’s web.config file.
2. Launch a Developer Command Prompt for VS2010 with elevated permissions.
3. Navigate to the folder containing your web role’s web.config file.
4. Run the following command: aspnet\_regiis -pef "connectionStrings" ".".

Encrypting the connection string on your live cloud service:

1. Remote desktop into your server instance.
2. Navigate to the WCF Data Service application folder containing web.config. Typically, \sitesroot\0 on drive E:\ or F:\.
3. Replace the existing **connectionStrings** element with the original connection string (From the backup you created when encrypting locally).
4. Save **web.config**.
5. Open a **Command Prompt**. From the folder path, enter **cmd** to open a **Command Prompt**.
6. Enter D:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet\_regiis -pef "connectionStrings" ".".

Setting permissions for your service:

1. Remote desktop into your server instance.
2. Open IIS manager and navigate to the virtual directory containing your data service.
3. Right-click on the virtual directory and select Manage Application->Advanced Settings.
4. Take down the Application Pool name.
5. Right-click on the virtual directory again and select Edit Permissions->Security.
6. If the Application Pool is missing from groups and users, add it using the name format “IIS APPPOOL\ApplicationPoolName”.
7. Make sure to give ApplicationPoolName Write permissions, if it needs write access.

Sources: <http://www.bluevalleytech.com/techtalk/blog/assigning-ntfs-folder-permission-to-iis7-application-pools/> and <http://stackoverflow.com/questions/4072675/wcf-data-service-update-returns-401-unauthorized-access-is-denied-due-to-inval>

### LO: Use LINQ to compose queries for a WCF data service

**Goal:**

Create client side queries to interact with the high score data service.

**Problems:**

1. I had to research methods of interacting with WCF data services in .NET.
2. My database needed to be redesigned as the original layout din not work well with my client side logic.
3. I had never used LINQ before.

**Solutions:**

I had initially considered writing some methods to output correctly formatted Uri queries but after some research I came upon this article: <http://msdn.microsoft.com/en-us/library/dd673933.aspx>. Basically, because of the implementation of the DataServiceQuery and the implementation of LINQ in .NET 3.5 and above, I could utilize LINQ to create properly structured queries with no extra parsing code.

LINQ stands for Language Integrated Query. This defines query expressions which the programmer can use to interact with query able containers or interfaces. For instance from, where, select, orderby, etc. You can see examples in the code samples below.

I had originally structured my database so that each level had its own table but due to the way the data is retrieved from the service this would have been very hard to work with and would not be very scalable. The problem was that when you execute a LINQ query on the database you must specify the entity type to operate on to view the objects’ data in its correct form. Due to the way the data model handles data base tables, entities from each table had different types, meaning that as the number of levels grew the complexity of the code would grow with it. The solution to this was to restructure the high scores into one table, with a level number identifier. This way the client code would be the same regardless of the number of levels, as all high score entities would be handled as the same object type.

Implementation:

The first step is to connect the client to the data service. I’m assuming you have already connected the application to the service; here I am strictly talking about the code. This is done by creating an instance of your database entities type:

private SticKartScores\_0Entities context;

this.context = new SticKartScores\_0Entities(new Uri(ScoreServiceConstants.ServiceUriString));

Once you have a database connection you can start querying it. The permissions on the data service will limit what permissions the client has. The following method checks if the score input has made it into the top ten high scores for a level, inserts it if it has and returns true if an insert or update has been performed. You can see the LINQ query used on the second line of the method.

private bool AddScoreToTable(ScoreNamePair scoreNamePair, int levelNumber)

{

bool highScore = false;

IQueryable<HighScore> scores = from score in this.context.HighScores where score.Level == levelNumber orderby score.Score ascending select score;

if (scores.Count() < 10)

{

this.context.AddToHighScores(HighScore.CreateHighScore(0, levelNumber, scoreNamePair.Name, scoreNamePair.Score));

this.context.SaveChanges();

highScore = true;

}

else if (scoreNamePair.CompareTo(new ScoreNamePair(scores.First().Score, scores.First().Name)) < 0)

{

HighScore lowest = scores.First();

lowest.Score = scoreNamePair.Score;

lowest.Name = scoreNamePair.Name;

this.context.UpdateObject(lowest);

this.context.SaveChanges();

highScore = true;

}

return highScore;

}

### LO: Set up SSL encryption on a WCF data service, hosted on an Azure cloud service

**Goal:**

Enable Secure Sockets Layer (SSL) encryption on a WCF data service, hosted on an Azure cloud service.

**Problems:**

1. I would need to obtain a certificate to use for signing.
2. I would need to deploy the certificate to the cloud service through Azure.
3. I would need to configure the data service to use the certificate.

**Solutions:**

Obtaining the certificate:

After reading up on setting up SSL with a service hosted on Azure I found out that the minimum required SSL certificate cryptographic key size was 2048-bits (<http://msdn.microsoft.com/en-us/library/windowsazure/ff795779.aspx>). As it would be quite expensive to purchase a certificate of this type I decided to use a self-signed certificate.

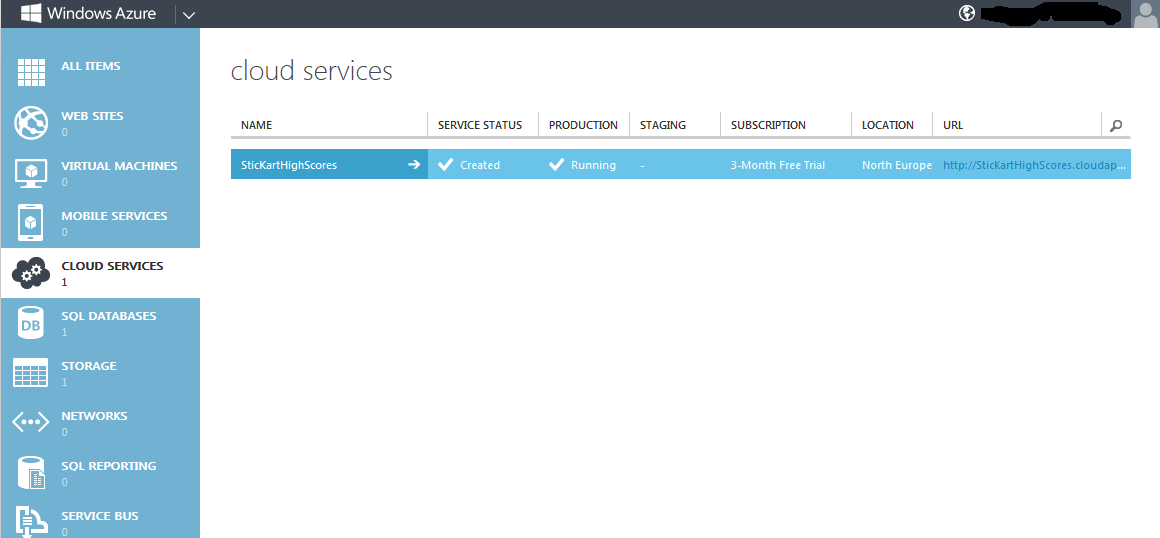
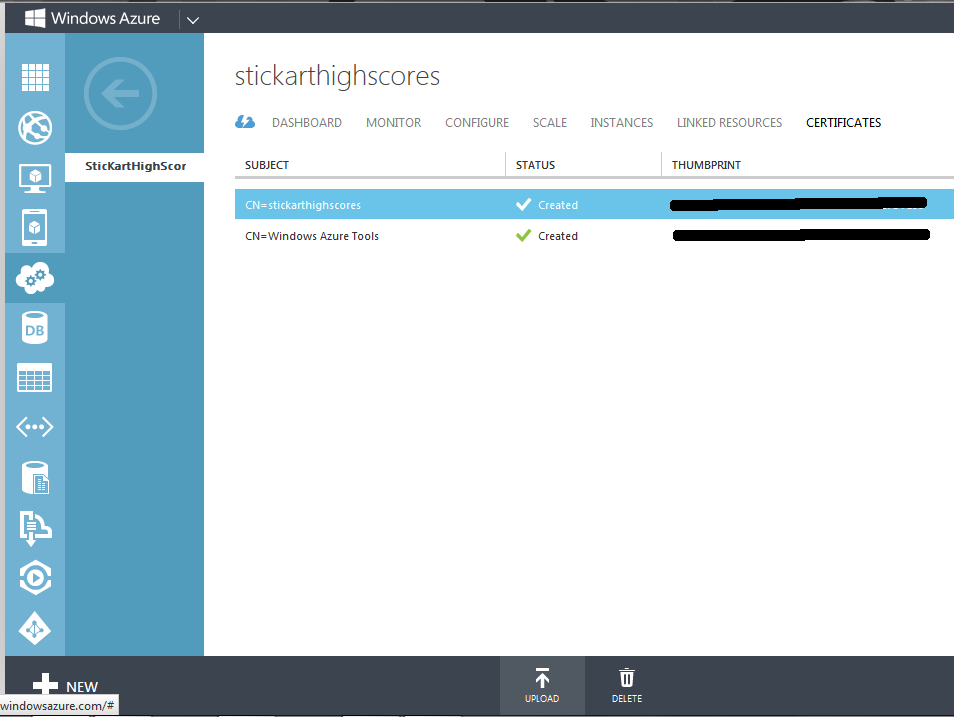
I used this site; <http://msdn.microsoft.com/en-us/library/windowsazure/gg551722.aspx>, as a guide on how to generate a self-signed certificate, using makecert. The steps are:

1. Open Visual Studio command prompt.
2. Execute the following command:  
   makecert -sky exchange -r -n "CN=<CertificateName>" -pe -a sha1 -len 2048 -ss My "<CertificateName>.cer"

The command basically says: make an exportable, self-signed, 2048-bit, X.509 certificate with type exchange, name "CN=<CertificateName>", using the sha1 algorithm to sign, put it in the personal certificate store and call the file "CN=<CertificateName>.cer". For more information on makecert see <http://msdn.microsoft.com/en-us/library/bfsktky3%28v=vs.80%29.aspx>.

Deploying the certificate to the Azure cloud service:

For this, I am assuming you already have your data service deployed on an Azure cloud service. The process to deploy an SSL cert with a new service deployment is slightly different. I used <http://msdn.microsoft.com/en-us/library/windowsazure/gg432987.aspx> and <http://www.windowsazure.com/en-us/develop/net/common-tasks/enable-ssl/> as guides. The steps in this process are:

1. Export the Personal Exchange Certificate (.pfx)
   1. Open the certificate manager by typing certmgr.msc in the Start menu textbox.
   2. If you followed the makecert method above, your certificate should be installed under personal certificates. If not, import your certificate here.
   3. Right-click on the certificate and select All tasks->Export.
   4. On the export private key page select yes, export the private key.
   5. Finish the wizard.
2. Get the certificate thumbprint
   1. Open the certificate manager again.
   2. Navigate to your certificate.
   3. Right-click on the certificate and select Open.
   4. Select Details->Thumbprint.
   5. Copy the thumbprint somewhere, capitalise it and remove the white spaces as this is what Azure expects.
3. Deploy the certificate
   1. Log into the Azure management portal <https://manage.windowsazure.com> and login.
   2. Navigate to cloud services and select your service.  
      
   3. Navigate to Certificates then select Upload.  
      
   4. From the dialogue, browse to the .pfx file you generated earlier, select it then enter the password which you used to create it in the password box.

Configuring the data service:

This is a very simple process and is explained very clearly here: <http://www.windowsazure.com/en-us/develop/net/common-tasks/enable-ssl/>. The steps are:

1. Update your service definition file (.csdef)
   1. Within the WebRole section, after the closing Imports tag, add a certificates section.  
      <WebRole name="CertificateTesting" vmsize="Small">  
      ...  
       <Certificates>  
      <Certificate name="CertificateName" storeLocation="LocalMachine" storeName="CA" />  
      </Certificates>  
      ...  
      </WebRole>  
      If you require more information on these values see <http://msdn.microsoft.com/en-us/library/windowsazure/gg465718.aspx>.
   2. In the Endpoints section, add an Input endpoint using the certificate you just added.  
      <Endpoints>  
      <InputEndpoint name="Endpoint2" protocol="https" port="443" certificate="CertificateName" />  
      </Endpoints>
   3. In the Bindings section, add a binding to the endpoint you just created.  
      <Bindings>  
      <Binding name="Endpoint2" endpointName="Endpoint2" />  
      </Bindings>
2. Update your cloud service configuration (ServiceConfiguration.Cloud.cscfg) file to identify the certificate to use.
   1. If your file does not contain a Certificates section add one between the Role tags, otherwise you just need to add the certificate to the certificates section.  
      <Role name="Deployment">  
      …  
      <Certificates>  
      <Certificate name=" CertificateName " thumbprint="9427BEfA18EC6865A9EBDC79D4C38DE50E6316FF" thumbprintAlgorithm="sha1" />  
      </Certificates>  
      ...  
      </Role>
3. Redeploy your data service.

### LO: Force a .NET client application to accept self-signed certificates, for SSL connections

**Goal:**

Enable my client application (game) to communicate with a WCF service through SSL using a self-signed certificate.

**Problems:**

1. By default a .NET client trying to access a service using a self-signed SSL certificate will throw an error and will not connect as the certificate is not trusted.
2. I had to research how to override this behaviour.

**Solutions:**

After doing some research I found this article <http://stackoverflow.com/questions/4977218/how-to-accept-a-self-signed-ssl-certificate-in-a-wcf-client>. It describes how to add an event handler to the .Net framework’s System.Net ServicePointManager to allow for custom certificate validation. I used the answer and modified it to only validate against a certificate with the same name and issuer as my self-signed certificate. To do this you must add the following references:

using System.Net;

using System.Security.Cryptography.X509Certificates;

Then add an event handler to your class as follows (CertName is a string constant):

private bool CertificateCheck(object sender, X509Certificate certificate, X509Chain chain, System.Net.Security.SslPolicyErrors error)

{

return certificate.Subject == ScoreServiceConstants.CertName && certificate.Issuer == ScoreServiceConstants.CertName;

}

Then add this event handler to the ServicePointManager.ServerCertificateValidationCallback before any calls to your service.

ServicePointManager.ServerCertificateValidationCallback += new System.Net.Security.RemoteCertificateValidationCallback(this.CertificateCheck);

Now update the data service Uri to use an https address and your client should now be able to communicate with your data service using SSL.

### LO: Create and configure service operations on a WCF data service

**Goal:**

Create service operations in a WCF data service to enforce specific access rules to database content.

**Problems:**

1. I had to research how to create service operations.
2. I had to research how to set permissions on a WCF data service.

**Solutions:**

After doing some research on setting access rules for data services I found the following page <http://msdn.microsoft.com/en-us/library/dd728275.aspx> which I used as a guide. Basically, to set access permissions to the data exposed by the service data entity model, on initialization of the data service we set entity access rules as below. This gives the data service write permissions to the high scores table and read permissions to the active players and statistics tables.

public class ScoresWcfDataService : DataService<SticKartScores\_0Entities>

{

public static void InitializeService(DataServiceConfiguration config)

{

config.SetEntitySetAccessRule("HighScores", EntitySetRights.All);

config.SetEntitySetAccessRule("ActivePlayers", EntitySetRights.AllRead);

config.SetEntitySetAccessRule("Statistics", EntitySetRights.AllRead);

}

}

These tables can then be queried from the client using properly formatted OData queries or LINQ. To add additional query operations, for instance adding updates or insertions, without giving the client write access we can add service operations. These can also be used to build a service API for the client to use. More information on these can be found at <http://msdn.microsoft.com/en-us/library/cc668788.aspx>. These can accept any number of arguments. The arguments must be serializable to strings. The return type must be void, IQueryable, IEnumerable, a type exposed by the entity model or a primitive class type. The service operation must use the WebGet or WebInvoke tag. We can use LINQ queries to access the data in the same way described for the client earlier. The only difference in working with the data is that we do not have to flag entities as updated before calling save on the database context. An example of this is registering a player below (Defensive code has been removed for simplicity). This checks if the player passed in has been registered with the service and registers them if not. It returns true if the player was registered and false otherwise.

[WebGet]

public bool RegisterPlayer(string name, string password)

{

SticKartScores\_0Entities context = this.CurrentDataSource;

var playersRegistered = (from player in context.Statistics where player.Name == name && player.Password == password select player).Take(1);

if (playersRegistered.Count() == 0)

{

Statistic playerData = new Statistic();

playerData.Name = name;

playerData.Password = password;

context.Statistics.AddObject(playerData);

context.SaveChanges();

return true;

}

else

{

return false;

}

}

The following Uri calls this query (You will not be authorized to actually call it): <https://stickarthighscores.cloudapp.net/ScoresWcfDataService.svc/RegisterPlayer?name=’name’&password=’1234’>

To set permissions for a service operation we must first set read permissions for the entity set(s) which it operates on. The operation can have higher permissions than this but the set must have at least read. We then follow the same format as setting entity set permissions but with a different method call. The line below is added after the entity set permissions described earlier and gives the register player service operation write permissions.

config.SetServiceOperationAccessRule("RegisterPlayer", ServiceOperationRights.All);

## Sprint 3 Learning Outcomes (LO):

### LO: Implement a thread safe notification system, using the singleton design pattern, in C#

**Goal:**

In C#, using the singleton design pattern, create a thread-safe notification manager for use in the game.

**Problems:**

1. I had to research how to lock entities correctly in C#.
2. I needed to ensure thread-safety during instantiation of the singleton object.
3. The notification manager would need to be initialized with a non-default constructor.
4. The notification manager would have to show notifications one at a time, in order of creation.

**Solutions:**

After doing some research on implementing the singleton design pattern in a thread-safe manner, I found the following sites and used them as a guide: <http://msdn.microsoft.com/en-us/library/ff650316.aspx> and <http://csharpindepth.com/Articles/General/Singleton.aspx>. These introduced me to thread-safety in C#.

I learned about the sealed and volatile keywords from here also. Sealed ensures that a class cannot be derived from, which in turn enforces the singleton pattern. If a class could derive from a singleton that would mean that more than one instance of the object could be created through the derived class. The sealed keyword can also speed up performance, due to optimizations in the just in time compiler. It is for these reasons that the notification manager is a sealed class.

public sealed class NotificationManager

The volatile keyword marks variables as modifiable by other threads, other processes and/or the operating system (<http://msdn.microsoft.com/en-us/library/x13ttww7%28v=vs.71%29.aspx>). The singleton instance and some other items in the notification manager class are marked as volatile for these reasons.

private static volatile NotificationManager managerSingleton = null;

I also learned that, when implementing a thread-safe singleton in c#, it is not a good idea to lock on this or on the static singleton instance as locking on either of these can cause dead lock. This is why I used a dummy object, with no other function, as my locking object. The object is declared as follows:

private static object mutex = new object();

It is then locked as follows:

lock (NotificationManager.mutex)

{

//Code here

}

The singleton instance is created by calling an initialization method. This initialization method implements double-check locking to ensure thread safety. This checks if the singleton instance is null. If it is the method locks on the dummy object, checks that the singleton instance is still null (in case another thread created it in the meantime), then creates the singleton instance. If it is not null at the first check then no time is wasted on locking. This is more efficient if the initialize method is being called repetitively.

public static NotificationManager Initialize(ContentManager contentManager, SpriteBatch spriteBatch, Vector2 displayDimensions)

{

if (NotificationManager.managerSingleton == null)

{

lock (NotificationManager.mutex)

{

if (NotificationManager.managerSingleton == null)

{

NotificationManager.managerSingleton = new NotificationManager(contentManager, spriteBatch, displayDimensions);

}

}

}

return NotificationManager.managerSingleton;

}

Once initialized the singleton instance can be accessed via a property. This does not need a lock as no element is being modified.

public static NotificationManager Instance

{

get

{

return NotificationManager.managerSingleton;

}

}

The actual notification system is implemented using a queue. This ensures notifications are processed in order of creation. The manager uses a notification factory to create notifications. These are then added to the queue of notifications. Note: there is also a notification settings manager involved, to ensure the player is not shown the same notification repeatedly and I removed some other defensive logic for simplicity.

public static void AddNotification(NotificationType type)

{

lock (NotificationManager.mutex)

{

NotificationManager.managerSingleton.notificationQueue.Enqueue(NotificationManager.managerSingleton.notificationFactory.Create(type));

}

}

In the update phase the notification manager updates the notification at the front of the queue and removes it from the queue, if it has been active for longer than its time to live. On draw the notification manager only draws the notification at the front of the queue. Each notification just manages its own time to live, updating of any animated sprites it contains and rendering of the visible entities it contains.

### LO: How to implement the IComparer and IComparable classes in C#, to allow for easy sorting of custom objects

**Goal:**

Create a high score table system which will store the top ten scores, with the players’ names, for each level.

**Problems:**

1. I would need to store a separate high score table for each level in the game.
2. Each high score table’s data would need to be sorted.
3. The data would need to be able to be serialized and deserialized easily.
4. I would need to research how to use the IComparer and IComparable classes.

**Solutions:**

I initially decided to structure my high score system like this: The game settings class (which is already being serialized and keeps track of total and unlocked level numbers) would store a collection of high score tables. The high score tables would in turn use a SortedList container to store key-value pairs of scores and names. The high score table would then just add to and remove from the SortedList.

Unfortunately, it is not possible to serialize the SortedList container using the XMLSerializer in C#. For this reason, I decided to create a class to store a score-name pair. I would then implement the IComparer and IComparable interfaces in this class. This would allow me to store the score-name pair objects in a List and use the built in List.Sort() method to sort them.

I found the following web sites very useful as guides for implementing the IComparer and IComparable interfaces: <http://support.microsoft.com/kb/320727> and <http://www.dotnetperls.com/icomparable>.

Basically, by implementing the IComparable interface you specify a default ordering for objects of the type, which allows you to use the .Sort() method of the built in collections. Implementing the IComparer interface allows you to specify extra comparison algorithms, which you can use to sort the objects in different orders.

To implement the IComparable interface in your class, you must inherit from the IComparable class which is in the System namespace.

public class ScoreNamePair : IComparable

{

//Code here

}

Then you must implement the interfaces CompareTo() method in your class. This method returns -1 for is less than, 0 for is equal to or 1 for is greater than the other object passed in. In this example Score is an integer and Name is a string. The method compares scores, so they will be sorted from high to low and if the scores are equal it compares the names, so they will be sorted alphabetically. Note the reverse ordering of the CompareTo() calls within the if statements. This is to ensure high to low numeric ordering and low to high alphabetic ordering.

public int CompareTo(object other)

{

ScoreNamePair otherPair = (ScoreNamePair)other;

if (this.Score == otherPair.Score)

{

return this.Name.CompareTo(otherPair.Name);

}

else

{

return otherPair.Score.CompareTo(this.Score);

}

}

To use the IComparer functionality in your class, you must create nested classes which inherit the IComparer interface and implement the IComparer.Compare method. The following comparison will result in objects being sorted by score, in descending order. You must use the IComparer interface relevant to the container you intend to use. To do this implement the IComparer interface from the same namespace as the container. In this case I would be storing and sorting the objects using a List which is in the System.Collections.Generic namespace. I included this namespace to get access to its IComparer.

private class SortScoreDescendingHelper : IComparer<ScoreNamePair>

{

int IComparer.Compare(object objectOne, object objectTwo)

{

if ((objectOne as ScoreNamePair).Score < (objectTwo as ScoreNamePair).Score)

{

return 1;

}

else if ((objectOne as ScoreNamePair).Score > (objectTwo as ScoreNamePair).Score)

{

return -1;

}

else

{

return 0;

}

}

}

You must then create a static method in your class, (the one which you want to compare), to retrieve an instance of your nested class as an IComparer.

public static IComparer<ScoreNamePair> SortScoreDescending()

{

return (IComparer<ScoreNamePair>)new SortScoreDescendingHelper();

}

To use the functionality of the IComparer implementation you created: When calling List.Sort(), pass a call to the static method which you created above to the .Sort(), method as an argument. In the example below this.Scores is a list of score-name pair objects.

this.Scores.Sort(ScoreNamePair.SortScoreDescending());

## Sprint 2 Learning Outcomes (LO):

### LO: Adjust the Kinect sensor's angle to ensure the player is being tracked correctly

**Goal:**

Implement procedural adjustment of the Kinect sensor’s angle of elevation so that the player is kept within the field of view of the sensor.

**Problems:**

1. I had to research how to adjust the sensor’s angle and the limitations of the adjustments.
2. I needed to ensure adjustment would only happen if the player was in the correct area, to ensure accuracy.
3. The adjustment would need to happen more than once if the sensor got moved or in other unusual circumstances.
4. Gesture detection would need to be disabled during sensor adjustment.

**Solutions:**

I found documentation for the Kinect sensor’s angle adjustment on msdn at <http://msdn.microsoft.com/en-us/library/microsoft.kinect.kinectsensor.elevationangle.aspx>. This informed me of the limits of the adjustments which could be made and also that there was a limit on the amount of consecutive adjustments which can be made to the angle.

To ensure the player was in the correct area for the adjustment to happen I employed a number of methods.

The first was to only allow the adjustment to happen while there is a skeleton in view and to perform the adjustment to suit the closest skeleton. The following is a simplified version of the code I used to track the closest skeleton.

if (this.ReadSkeletonFrame()) // This reads in skeleton data and returns true if successful.

{

Skeleton closestSkeleton = null;

foreach (Skeleton skeleton in this.skeletonData)

{

switch (skeleton.TrackingState)

{

case SkeletonTrackingState.Tracked: // Only check against fully tracked skeletons.

if (closestSkeleton == null || skeleton.Position.Z < closestSkeleton.Position.Z)

closestSkeleton = skeleton;

break;

default:

break;

}

}

// The code below goes here.

}

The second was to monitor the player’s depth value and to only allow the angle adjustment to occur when the player is at a set minimum distance from the sensor (2.3m). This ensures that the player is far enough away that their entire body would be within the sensor’s field of view. Gesture detection is also disabled while the player is too close. The video below shows the user being informed to step back with their position in floor space displayed above.



The third was to readjust the angle if the player’s head or legs were out of view and all the previous criteria were met. This allows for correction from an inaccurate adjustment at the start of the game. For example; the player starts the game at an elevated point, the sensor adjusts to suit the elevated position and then the player steps down. The sensor would then readjust.

A simplified version of the code to implement the two of these is as below. The closest skeleton is the one calculated above. This calls the method to adjust the sensor on the first run through then sets the angle-set flag to false, if either the lower half or the upper half of the player’s body is not being tracked. This flag triggers a readjustment of the sensor angle in the next update cycle. This code also prevents the gestures from updating while the sensor is adjusting.

if (closestSkeleton.Position.Z > this.minimumPlayerDistance)

{

if (this.kinectAngleSet)

{

//Update gesture data, etc.

this.kinectAngleSet = closestSkeleton.Joints[JointType.Head].TrackingState == JointTrackingState.Tracked && closestSkeleton.Joints[JointType.KneeLeft].TrackingState == JointTrackingState.Tracked;

}

else

{

this.AdjustSensorAngle(closestSkeleton);

}

}

The actual adjustment method is quite simple. If the player’s spine joint is being tracked I use basic trigonometry to calculate the angle between centre of the sensor’s field of view and the skeleton’s spine joint. The sensor is then adjusted by this angle. If the spine is not in view the head or a foot is used as a point of reference and the sensor is incrementally adjusted until the spine is in view.

private void AdjustSensorAngle(Skeleton skeleton)

{

if (skeleton.Joints[JointType.Spine].TrackingState == JointTrackingState.Tracked)

{

Vector2 jointMapping = Vector2.Normalize(new Vector2(skeleton.Joints[JointType.Spine].Position.Z, skeleton.Joints[JointType.Spine].Position.Y));

// Calculate the angle from the centre of the field of view to the skeleton’s spine

float angle = MathHelper.ToDegrees((float)Math.Asin(jointMapping.Y));

if (angle > this.thresholdAngleToBody || -angle > this.thresholdAngleToBody)

{

// The method TrySetElevetionAngle is just a wrapper for exception handling

// if the angle value is too high.

this.kinectSensor.TrySetElevationAngle(this.kinectSensor.ElevationAngle + (int)angle);

}

this.kinectAngleSet = true;

}

else if (skeleton.Joints[JointType.Head].TrackingState == JointTrackingState.Tracked)

{

this.kinectSensor.TrySetElevationAngle(this.kinectSensor.ElevationAngle - 4);

}

else if (skeleton.Joints[JointType.FootLeft].TrackingState == JointTrackingState.Tracked)

{

this.kinectSensor.TrySetElevationAngle(this.kinectSensor.ElevationAngle + 4);

}

}

### LO: Track a user's hand position smoothly, consistently and accurately

**Goal:**

Refine the tracking of the player’s hand position so the menu system is easier to navigate using Kinect gesture input.

**Problems:**

1. The on screen hand position had to be completely independent of the player’s body position.
2. I would need to research how to apply smoothing to the skeleton data coming from the Kinect sensor.
3. The changes would have to have a limited or no impact on the gestures which were already defined.
4. The hand tracking would need to allow people with varying arm lengths to reach all menu items.

**Solutions:**

I took a multi-pronged approach to this task. The first change I made was to increase the resolution of the depth-stream data which was being retrieving from the sensor. This allowed for greater precision when converting between 3D world space and 2D screen space and reduced jitter in the controls. This is done by setting the DepthImageFormat to a higher resolution value when enabling the depth stream.

this.kinectSensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);

I then altered the algorithm I was using to position the hand on screen. I set this up so that it is based on the offset from the shoulder to the hand. This works by converting the shoulder’s position and the hand’s position into 2D space then calculating the offset from the shoulder to the hand. This value is then scaled and added to the centre position of the screen to get the on-screen hand position. This makes the on screen representation independent of the player’s overall body position, as the shoulder is always translated to the centre of the screen. The following is the code to perform this task (with some defensive code removed). Tools.Convert is a method to convert a 3D skeleton point into 2D space. It returns a value between (-1, -1) and (1, 1).

public Vector2 HandPosition

{

get

{

Vector2 handPosition = Tools.Convert(this.kinectSensor, this.gestureManager.HandPosition, this.coordinateMapper);

Vector2 shoulderPosition = Tools.Convert(this.kinectSensor, this.gestureManager.ShoulderPosition, this.coordinateMapper);

Vector2 scaling = this.screenDimensions \* 1.8f;

Vector2 relativeHandPosition = handPosition - shoulderPosition;

relativeHandPosition.X \*= scaling.X;

relativeHandPosition.Y \*= scaling.Y;

return (this.screenDimensions / 2.0f) + relativeHandPosition;

}

}

I then added smoothing parameters to the skeleton stream to reduce jitter in the controls and to make menu navigation smoother. I researched how to implement this and found the blog at <http://cm-bloggers.blogspot.ie/2011/07/kinect-sdk-smoothing-skeleton-data.html> to be the most useful by far. It gives a detailed breakdown of what each smoothing parameter does. I had to test this with a number of different parameter values so that I could keep the gesture detection quick and accurate while still getting the advantages of the data smoothing. To enable smoothing of the skeleton data the smoothing parameters must be passed to the skeleton stream when you are enabling it, as so:

TransformSmoothParameters smoothing = new TransformSmoothParameters();

smoothing.Smoothing = 0.6f;

smoothing.Correction = 0.2f;

smoothing.JitterRadius = 0.125f;

smoothing.Prediction = 0.5f;

smoothing.MaxDeviationRadius = 0.04f;

this.kinectSensor.SkeletonStream.Enable(smoothing);

### LO: Create a motor driven entity using multiple bodies and joints using the Farseer physics engine

**Goal:**

Implement a moveable game character which suits movement along platforms and uneven terrain using the Farseer physics engine.

**Problems:**

1. The character would have to move in a human-like manner.
2. The character would have to move over flat and curved surfaces with ease.
3. The character would have to be easily controllable.
4. I would have to research how to create joints using the Farseer physics engine.
5. The character would have to have modifiable/deformable collision bounds to accommodate both standing and crouching body states.

**Solutions:**

I found some information on creating a game character with reasonably realistic movement using Farseer at <http://amazingretardo.simiansoftwerks.com/2010/02/17/platformer-character-control-farseer-physics-engine/> and about three quarters of the way down [www.madgamedev.com/post/2010/09/09/Article-XNA-Farseer-Platform-Physics-Tutorial.aspx](http://www.madgamedev.com/post/2010/09/09/Article-XNA-Farseer-Platform-Physics-Tutorial.aspx). This solution solves a lot of the problems associated with moving a character using a 2D physics engine. This is because the wheel can be used to simulate acceleration accurately and can also be used to reset some of the character’s abilities (explained further down).

I used this idea of a box connected to a motor driven circle as a base for my character. Basically a box is connected to a circle using a revolute (revolving) joint which turns it into a wheel. To prevent the box from just flopping around on the wheel, it is kept upright using a fixed angle joint. To move the character all you then have to do is apply a motor force to the circle, which causes it to spin, pushing the box forward.

My character needed to have a changeable height so I added a third body to this setup. The third body was a longer body which would span the full height of the character. The smaller box would only span the height of the crouching character. These two bodies would be joined using a weld joint. I could then use collision filtering to prevent collisions with the large body while the character is crouched and prevent collisions with the small body while the character is standing.

Original idea My Implementation

The bodies are created using the Farseer engine’s BodyFactory and the joints can be created using the JointFactory. The code to create the bodies and joints is below. (I have removed a lot of conversions and settings for simplicity)

// Full length body

this.fullBody = BodyFactory.CreateRectangle(physicsWorld, this.Width, this.Height \* 0.75f, density, this.fullBodyOffset);

this.fullBody.CollisionCategories = EntityConstants.StickManCategory;

// Half-length body

this.smallBody = BodyFactory.CreateRectangle(physicsWorld, this.Width, this.Height \* 0.25f, density, this.smallBodyOffset);

this.smallBody.IgnoreCollisionWith(this.fullBody);

this.smallBody.CollisionCategories = EntityConstants.StickManCategory;

// Wheel body

this.wheelBody = BodyFactory.CreateCircle(physicsWorld, this.Width / 2.0f, density, this.wheelBodyOffset);

this.wheelBody.IgnoreCollisionWith(this.smallBody);

this.wheelBody.IgnoreCollisionWith(this.fullBody);

// This is set high to prevent rolling when stationary and to give the greatest control over movement

this.wheelBody.Friction = float.MaxValue;.

this.wheelBody.CollisionCategories = EntityConstants.StickManCategory;

// Weld joint between two boxes.

this.bodyJoint = JointFactory.CreateWeldJoint(physicsWorld, this.fullBody, this.smallBody, this.smallBody.Position);

// Fixed angle joint to hold the boxes upright

this.angleUprightJoint = JointFactory.CreateFixedAngleJoint(physicsWorld, this.smallBody);

// Revolute joint to act as the entity’s motor

this.motorJoint = JointFactory.CreateRevoluteJoint(physicsWorld, this.smallBody, this.wheelBody, Vector2.Zero);

this.motorJoint.MotorEnabled = true; // Enables the use of the joint as a motor.

To achieve the collision size switching I used collision filtering. On a collision with the tall body, the collision is only processed if the character is not crouching. Inversely the collisions with the small body are disregarded unless the character is crouching.

The wheel can then be used to control the character’s ability to jump and collide with platforms. I refined the collision detection with the wheel to only reset the jump ability by checking if the point of the collision was below the wheel’s centre position. I used this page as a guide: <http://farseerphysics.codeplex.com/discussions/250916>. The code to do this is as follows:

// Adds the event handler for collisions to the wheel physics body

this.wheelBody.OnCollision += this.CollisionHandlerWheel;

// The return value of this event handler tells the physics engine

// whether to calculate the collision response or not.

// Multiple lines have been left out for simplicity.

private bool CollisionHandlerWheel(Fixture fixtureOne, Fixture fixtureTwo, Contact contact)

{

switch (fixtureTwo.CollisionCategories)

{

case EntityConstants.PlatformCategory:

collided = false;

// This checks if the platform is below the wheel if collisions

if (fixtureOne.Body.Position.Y < fixtureTwo.Body.Position.Y)

{

this.Land(); // Land resets the character’s state to running/standing.

collided = true;

}

break;

//…

}

}

### LO: Implement procedural population of verbal commands for use in speech detection

**Goal:**

Implement procedural population of the speech engine, with all selectable item names used in menus, so that I do not have to manually add commands as menus are added.

**Problems:**

1. The input manager would need to be fed with all possible commands.
2. Selectable items in the level select menu would have to retain their numerical value, for use in level selection, while displaying and responding to the verbally spoken version of the number.
3. The menu manager would have to be able to retrieve selectable item names from each menu.

**Solutions:**

In each menu, upon adding an item to the menu, it checks if the item is selectable and adds its name to a collection of selectable item names. This takes advantage of the polymorphic structure of my menu item hierarchy. The menu manager can then be called to amalgamate all of these collections into one single collection of selectable item names. The code for populating a menu’s selectable item collection is below.

public void AddItem(MenuItem menuItem)

{

if (typeof(MenuSelectableItem).IsAssignableFrom(menuItem.Type))

{

this.selectableItemNames.Add((menuItem as MenuSelectableItem).Name.ToUpperInvariant());

}

this.menuItems.Add(menuItem);

}

However this system breaks down somewhat when it comes to the level select menu. The problem is that the numerical values of the buttons are used in level selection but the numerical values cannot be used to build the grammar of the speech engine. For this I had to build a converter from a numerical value to its verbal counterpart (i.e. “1” goes to “one”). The converter only operates on numbers from 1 to 999 and works as follows;

I create lookup arrays of digits, teens and tens:

string hundred = "hundred";

string[] otherTens = { string.Empty, string.Empty, "twenty", "thirty", "fourty", "fiftey", "sixtey", "seventy", "eighty", "ninety" };

string[] teens = { "ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen", "eighteen", "nineteen" };

string[] singles = { string.Empty, "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" };

I then use division and modulus operations to count the number of hundreds, tens & digits and assign their verbal counterparts to the string numberAsWords:

// number is the numeric value to convert

string numberAsWords = string.Empty;

int leftover = number;

int hundredCount = leftover / 100;

leftover = leftover % 100;

if (hundredCount > 0)

{

numberAsWords += singles[hundredCount] + " " + hundred;

if (leftover > 0)

numberAsWords += " and ";

}

int tenCount = leftover / 10;

leftover = leftover % 10;

if (tenCount == 1)

{

numberAsWords += teens[leftover];

}

else if (tenCount != 0)

{

numberAsWords += otherTens[tenCount];

if (leftover > 0)

numberAsWords += " ";

}

if (leftover > 0 && tenCount != 1)

{

numberAsWords += singles[leftover];

}

return numberAsWords;

The level select menu then has to be treated slightly differently by the menu manager, when amalgamating the selectable item names, to account for this. This is done as follows: (I have removed some error checks and other defensive code for simplicity).

public List<string> GetAllSelectableNames()

{

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

foreach (MenuType menuType in this.menus.Keys)

{

switch (menuType)

{

case MenuType.LevelSelect:

foreach (string name in this.menus[menuType].SelectableItemNames)

{

// If name is “1”; it is parsed to the integer 1, then converted to “one”.

selectableNames.Add(ConvertToWords.ConvertIntToWords(int.Parse(name)));

}

break;

default:

if (this.menus[menuType] != null)

{

selectableNames.AddRange(this.menus[menuType].SelectableItemNames);

}

break;

}

}

return selectableNames;

}

This collection is then passed to the input manager when trying to initialize the speech engine. The collection is then iterated through. Each element is used to create a grammar choice which the speech engine can interpret as a command. The original, manual version of this code can be seen in the learning outcome; “Implement voice commands based on the Kinect's audio stream”.

// selectableNames is the list of selectable item names passed into this method.

this.speechEngine = new SpeechRecognitionEngine(recognizerInfo.Id);

Choices grammarChoices = new Choices();

foreach (string name in selectableNames)

{

grammarChoices.Add(new SemanticResultValue(name.ToLower(), name));

}

GrammarBuilder grammarBuilder = new GrammarBuilder();

grammarBuilder.Culture = recognizerInfo.Culture;

grammarBuilder.Append(grammarChoices);

Grammar grammar = new Grammar(grammarBuilder);

this.speechEngine.LoadGrammar(grammar);

### LO: Detect run and jump actions from Kinect input using gestures

**Goal:**

Implement action based input commands, using input from multiple gesture detectors. In plain English; using the gesture manager which I implemented earlier, monitor two gesture detectors tracking each of the player’s feet and interpret run and jump actions where appropriate.

**Problems:**

1. I would need to create a vertical swipe gesture detector, which would need to track feet and allow for switching off down gesture detection.
2. The DateTime which was used to track gesture times would not be accurate enough for differentiating between run and jump actions.
3. Converting the detected gestures from these detectors would need to be handled differently to the other gesture detectors.
4. The run action would need to be easy enough to do without heavy physical exertion.

**Solutions**

I created a vertical swipe gesture detector which I could toggle up and/or down gesture detection on and off. I created two instances of this gesture detector type in the gesture manager and set each of them to track one of the player’s ankles. I set them both to ignore downward swipe gestures.

On updating of the gesture manager; if a gesture detector which was tracking an ankle had detected a gesture, a leg gesture processing method was called. This also updates a leg lift timer which is used by the process leg gesture method. This timer gives greater accuracy than DateTime.Now() or DateTime.UtcNow(), as these are only accurate to within about 50ms. I had issues with run actions being detected as jump actions due to this inaccuracy initially.

DateTime.Now() is also horribly inefficient. See <http://www.eggheadcafe.com/tutorials/csharp/71b57428-6b59-4466-9762-ecb437ffac98/is-twitter-good-for-developers--and-datetimenow.aspx> or <http://jason-mitchell.com/xna/xna-and-c-calling-datetime-now-is-expensive/> if you are interested.

if (this.lastLegLiftCounter < this.runTimeLimit)

{

this.lastLegLiftCounter += gameTime.ElapsedGameTime.TotalSeconds;

}

this.skeletonJoints = skeleton.Joints;

foreach (GestureDetector gestureDetector in this.gestureDetectors)

{

gestureDetector.Add(this.skeletonJoints[gestureDetector.JointToTrack].Position);

if (gestureDetector.GestureDetected != GestureType.None)

{

if (gestureDetector.JointToTrack == JointType.AnkleLeft || gestureDetector.JointToTrack == JointType.AnkleRight)

{

this.ProcessLegGesture(gestureDetector.JointToTrack);

}

}

}

This method kept track of which ankle was last lifted last and how long since the lift happened. If the current ankle being lifted was not the last ankle lifted then the method would check for a run or jump action. If both legs were lifted within a very short timeframe of each other the action was taken to be a jump, otherwise if they were both lifted within a longer timeframe the action was taken to be a run action. This window of detection for a run gesture would allow for detection of moderate march up to a fast sprint.

private void ProcessLegGesture(JointType jointTracked)

{

if ((this.lastLegLifted == JointType.AnkleLeft && jointTracked == JointType.AnkleRight) || (this.lastLegLifted == JointType.AnkleRight && jointTracked == JointType.AnkleLeft))

{

if (this.lastLegLiftCounter < this.jumpTimeLimit)

{

this.detectedGestures.Enqueue(GestureType.Jump);

}

else if (this.lastLegLiftCounter < this.runTimeLimit)

{

this.detectedGestures.Enqueue(GestureType.Run);

}

this.lastLegLiftCounter = 0.0;

}

this.lastLegLifted = jointTracked;

}

## Sprint 1 Learning Outcomes (LO):

### LO: Use the Kinect sensor as an input device

**Goal:**

Set up the use of a Kinect sensor correctly in the project.

**Problems:**

1. I had to research how this was done.
2. I needed to research how to initialize and dispose of the sensor’s resources correctly.
3. I needed to tailor the implementation to suit the requirements of a game’s update cycle.

**Solution:**

I used the guide found at <http://tobint.com/blog/kinect-for-windows-sdk/> and read through the source contained in the “XNA Basics” and “Kinect Explorer” samples, which are contained in the Kinect for Windows Developer Toolkit (part of the SDK), to get up and running. Most of this is aimed at use with Windows Presentation Foundation applications and the XNA version wasn’t suited to my needs either but they both contributed heavily to my final solution.

The following tries to initialize the sensor and enables the depth and skeleton data streams. This is done by checking if there are any Kinect sensors available, checking if the sensor is connected and ensuring that the device is not already in use by another process.

bool successful = false;

if (KinectSensor.KinectSensors.Count > 0)

{

this.kinectSensor = KinectSensor.KinectSensors[0];

if (this.kinectSensor.Status == KinectStatus.Connected)

{

this.kinectSensor.DepthStream.Enable(DepthImageFormat.Resolution320x240Fps30);

this.kinectSensor.SkeletonStream.Enable();

try

{

this.kinectSensor.Start();

successful = true;

}

catch (IOException)

{

// The device is in use by another process.

}

}

}

The following code gets the skeleton data from the skeleton data stream using polling. The solution here comes from the “XNA Basics” sample project in the Kinect for Windows Developer Toolkit.

using (this.skeletonFrame = this.kinectSensor.SkeletonStream.OpenNextFrame(0))

{

// Sometimes we get a null frame back if no data is ready

if (null == this.skeletonFrame)

{

return false;

}

// Reallocate if necessary

if (null == this.skeletonData || this.skeletonData.Length != this.skeletonFrame.SkeletonArrayLength)

{

this.skeletonData = new Skeleton[this.skeletonFrame.SkeletonArrayLength];

}

this.skeletonFrame.CopySkeletonDataTo(this.skeletonData);

return true;

}

### LO: Implement gesture tracking using the Kinect sensor

**Goal:**

Create a gesture detection system by tracking bone or joint positions over time.

**Problems:**

1. Figuring out how to create a gesture interpreter which can accept a variety of gesture types.
2. Converting the gestures detected into actionable game commands.

**Solutions:**

I used the project at <http://kinecttoolbox.codeplex.com/> as a guide for handling gesture tracking. This project was built for the use with Windows Presentation Foundation so I had to rewrite the gesture detector to suit XNA better.

Basically a gesture detector contains a list of gesture entries, each of which stores a position and a time. This abstract gesture detector is defined by the variables in this constructor:

public GestureDetector(JointType jointToTrack = JointType.HandRight, int maxRecordedPositions = 20, int millisecondsBetweenGestures = 0)

{

this.GestureDetected = GestureType.None;

this.gestureEntries = new List<GestureEntry>();

this.JointToTrack = jointToTrack;

this.maxRecordedPositions = maxRecordedPositions;

this.millisecondsBetweenGestures = millisecondsBetweenGestures;

}

It implements methods to add GestureEntry objects, reset the detector and flag a gesture as found. It leaves the method to look for a gesture as abstract so any inherited class must implement their own specific code to detect their specific gesture type. For instance the “push” gesture detector implements this as below. ScanPositions is a method which performs a search on the list of gesture entries based on the functions passed in.

protected override void LookForGesture()

{

if (this.ScanPositions(

(p1, p2) => Math.Abs(p2.Y - p1.Y) < this.pushMaximumHeight,

(p1, p2) => Math.Abs(p2.X - p1.X) < this.pushMaximumWidth,

(p1, p2) => p2.Z - p1.Z < 0.01f,

(p1, p2) => Math.Abs(p2.Z - p1.Z) > this.pushMinimumLength,

this.pushMinimumDuration,

this.pushMaximumDuration))

{

this.GestureFound(GestureType.Push);

return;

}

}

I then implemented a gesture manger class to monitor all of the active gestures which contains a collection of gesture detectors and a queue of detected gestures. This manages the updating of gesture detectors and allows for the input manager to query for any available gestures. I have created it this way to allow for the processing of multiple individual gestures into one unified action (e.g. running). The following is the update method of the gesture manager class. Skeleton is a parameter of the method.

this.skeletonJoints = skeleton.Joints;

foreach (GestureDetector gestureDetector in this.gestureDetectors)

{

if (this.skeletonJoints[gestureDetector.JointToTrack].TrackingState != JointTrackingState.NotTracked)

{

gestureDetector.Add(this.skeletonJoints[gestureDetector.JointToTrack].Position);

if (gestureDetector.GestureDetected != GestureType.None)

{

this.detectedGestures.Enqueue(gestureDetector.GestureDetected);

gestureDetector.Reset();

}

}

}

The input manager can then simply query the gesture manager for the next gesture detected and apply the correct game action.

### LO: Create an extendable NUI-friendly menu system

**Goal:**

Create a tile based menu system which will work with Kinect gesture input and voice commands.

**Problems:**

1. The menu system must be flexible enough to cater for multiple menu types and layouts.
2. Any menu text must be large, self-centring and easy to read.
3. If a menu item is selectable it must be selectable by position or by name and must share its name with the voice command system.
4. Any selectable item must be selectable using a hand gesture.

**Solutions:**

To tackle the issue of flexibility I utilized an inheritance hierarchy with its root at an abstract menu item class. This class only stores its position relative to its parent/owner. It also allows retrieval of its type. This will be explained later. It is defined as follows:

public abstract class MenuItem

{

public MenuItem(Vector2 relativePosition)

{

this.RelativePosition = relativePosition;

}

public Vector2 RelativePosition { get; protected set; }

public abstract Type Type { get; }

public abstract void Draw(Vector2 parentPosition);

}

I created a number of sub-classes which inherit from this. These include; A menu image class which draws a sprite centred on its relative position, a menu text class which renders text centred on its relative position and an abstract selectable menu item class which contains a bounding box, a name and a method to check if its bounding box has been hit. From the selectable menu item I created a button sub-class which stores a menu image for its background, a menu image for its icon and a menu text item to display its name.

I then created a generic menu class which stores a collection of menu items and a collection of selectable item names. As items are added to the menu, the list of selectable item names is generated. This is one of the places the Type property of the menu item class is used. This is done by taking advantage of polymorphism, as so:

public void AddItem(MenuItem menuItem)

{

if (typeof(MenuSelectableItem).IsAssignableFrom(menuItem.Type))

{

this.selectableItemNames.Add((menuItem as MenuSelectableItem).Name.ToUpperInvariant());

}

this.menuItems.Add(menuItem);

}

This class also implements methods to check for a selection based on a string input (for voice commands) and/or based on a position input. This is also done using the Type attribute and polymorphism. The following is the selection check based on the name passed in. (Some defensive code has been removed for simplicity.)

public string CheckForSelection(string name)

{

string itemFound = null;

foreach (MenuItem menuItem in this.menuItems)

{

if (typeof(MenuSelectableItem).IsAssignableFrom(menuItem.Type))

{

if (name.ToUpperInvariant() == (menuItem as MenuSelectableItem).Name.ToUpperInvariant())

{

itemFound = (menuItem as MenuSelectableItem).Name;

break;

}

}

}

return itemFound;

}

Although not implemented yet, this class allows for querying of the selectable item names contained within, which will allow for procedurally loaded grammar sets based on the contents of all the menus in the game.

I created a menu manager class to manage the active menus and to call events which change the game state. I created a menu factory class to abstract the creation of menus from the manager. The menus are stored in a dictionary with the menu type as the key and the menu as the value. This allows the manager to just change the active menu type and then update the corresponding menu stored at the active type’s location.

The menus are created in the manager as so:

this.ActiveMenu = MenuType.Main;

this.menus.Add(MenuType.Main, MenuFactory.CreateMainMenu(contentManager, spriteBatch, this.screenDimensions / 2.0f));

And updated as so, where selectionPosition and selectionName are parameters of the menu manager’s update method:

if (this.menus[this.ActiveMenu] != null)

{

string selectedItemName = null;

if (selectionPosition != Vector2.Zero)

{

selectedItemName = this.menus[this.ActiveMenu].CheckForSelection(selectionPosition);

}

else if (selectionName != null)

{

selectedItemName = this.menus[this.ActiveMenu].CheckForSelection(selectionName);

}

// State logic here based on selectedItemName

}

The manager sends the following events which are implemented by the game to control game state:

/// <summary>

/// An event triggered on the user selecting exit.

/// </summary>

public event Action<bool> OnQuitGameDetected;

/// <summary>

/// An event triggered on the user selecting to continue gameplay.

/// </summary>

public event Action<bool> OnResumeGameDetected;

/// <summary>

/// An event triggered on the user selecting to start a new level.

/// </summary>

public event Action<bool> OnBeginLevelDetected;

I implemented a push gesture, as mentioned in the implement gesture tracking LO, to act as a selection gesture while the game is in the menu state. I also created a class containing string constants which are used for naming selectable menu items and in the creation of grammar for voice commands. I intend to replace this with procedurally set voice commands, based on the contents of the menus.

Note: In the following video the red circle represents the users hand position, translated into screen coordinates. Also any actions which are happening without any on screen input are happening as a result of voice commands.



### LO: Configure Visual Studio to create an installer package which will download and install custom dependencies

**Goal:**

Figure out how to configure the Visual Studio “publish” functionality to include custom dependencies in the installer package and to download these dependencies.

**Problems:**

1. Research adding custom dependencies to a Visual Studio generated installation package.
2. Once I had the installer working, the installed game kept crashing when the Kinect sensor was plugged in on any computer except my development machine.

**Solutions:**

I did some research into adding custom dependencies and found this StackOverflow post (<http://stackoverflow.com/questions/1334436/adding-custom-prerequsites-to-visual-studio-setup-project>) which further linked me to a msdn article (<http://msdn.microsoft.com/en-us/library/ms165429.aspx>) on how to create custom “bootstrappers” for Visual Studio.

Basically, to add custom dependencies to an installation package in Visual Studio you first have to create custom bootstrapper files for the dependencies and then add these files to the directory that Visual Studio reads its additional dependencies list from.

The msdn article actually had some incorrect information in it. I found this out by reading the installation log of a failed reinstallation. It turned out that the XML below was the offender. The product and property tags were the wrong way around which was causing installer errors.

<InstallChecks>

<MsiProductCheck

Product="IsMsiInstalled"

Property="{XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}"/>

</InstallChecks>

The XXXXX….XXXXX section is the GUID of the installed msi file. This is required to bypass the installation of components which are already installed. I tried to find the GUIDs for my dependencies but to no avail. I ended up finding this blog (<http://elmaskubilay.blogspot.ie/2012/06/find-guid-globally-unique-identifier-of.html>) on how to get the list of programs and their msi GUID values. I ran these steps on my own machine and then searched through the generated file for the correct GUIDs. This worked a treat.

The next step was to get the installer package to download the dependencies instead of including them in the installation package. To do this I had a look at some of the package and product files which were already there. It turned out that if you included a link to the download location in the package.xml file, referenced this in the product.xml file and changed the CopyAllPackageFiles value as below I could achieve this result.

Product.xml:

<PackageFiles CopyAllPackageFiles="IfNotHomeSite">

<PackageFile Name="KinectRuntime-v1.6-Setup.exe" HomeSite="KinectRedistHomeSite" />

</PackageFiles>

Package.xml:

<Strings>

<String Name="KinectRedistHomeSite">DownloadLocationHere</String>

<Strings>

The issue with the installed game crashing when the Kinect sensor was connected turned out to be due to a lack of support, in the redistributable, for the Xbox Kinect sensor. The Kinect for Windows runtime only supports the Kinect for Windows device. I found this out by writing a basic logger and outputting states of objects before and after their initialization. I implemented a simple check for this and defaulted to keyboard input if status of the device was not supported. The code to do this is in the following line in the learning outcome “Use the Kinect sensor as an input device”.

if (this.kinectSensor.Status == KinectStatus.Connected)

### LO: Implement voice commands based on the Kinect's audio stream

**Goal:**

Using the Kinect sensor and the Microsoft Speech platform to accept voice input and convert a set of speech commands into in-game commands.

**Problems:**

1. I need to research how audio is retrieved from the Kinect for Windows SDK.
2. I need to research how to use the Microsoft Speech platform to interpret sounds as words.
3. I need to get these two technologies working together.

**Solution:**

I found the following sources as a guide and created a solution that meets the project’s needs:

The Purple Book: (<http://channel9.msdn.com/coding4fun/kinect/The-Purple-Book-Using-Kinect-for-Windows-with-XNA>)

SpeechBasics-WPF tutorial: Contained in the Kinect developer toolkit browser which is installed as part of the Kinect for Windows SDK.

The following is the event which is fired when the speech engine may have detected a registered piece of grammar. SpeechConfidenceThreshold is a constant which defines the cut off level of confidence at which to ignore false positives.

private void SpeechRecognized(object sender, SpeechRecognizedEventArgs e)

{

if (e.Result.Confidence >= InputManager.SpeechConfidenceThreshold)

{

this.selectedWord = e.Result.Semantics.Value.ToString();

}

}

The following starts the speech engine and adds two words to the grammar recognized by the speech engine instance. The second last line sets the Kinect sensor’s audio stream as the input of the speech engine. The SelectableNames class contains string constants. It is also possible to build grammar sets from predefined files but this does not suit my needs, as I intend to load all of the menu control grammar dynamically.

RecognizerInfo recognizerInfo = this.GetKinectRecognizer();

this.speechEngine = new SpeechRecognitionEngine(recognizerInfo.Id);

Choices grammarChoices = new Choices();

grammarChoices.Add(new SemanticResultValue(SelectableNames.PlayButtonName.ToLower(), SelectableNames.PlayButtonName));

grammarChoices.Add(new SemanticResultValue(SelectableNames.PauseCommandName.ToLower(), SelectableNames.PauseCommandName));

GrammarBuilder grammarBuilder = new GrammarBuilder();

grammarBuilder.Culture = recognizerInfo.Culture;

grammarBuilder.Append(grammarChoices);

Grammar grammar = new Grammar(grammarBuilder);

this.speechEngine.LoadGrammar(grammar);

this.speechEngine.SpeechRecognized += this.SpeechRecognized;

this.speechEngine.SetInputToAudioStream(this.kinectSensor.AudioSource.Start(), new SpeechAudioFormatInfo(EncodingFormat.Pcm, 16000, 16, 1, 32000, 2, null));

this.speechEngine.RecognizeAsync(RecognizeMode.Multiple);

This code here instructs the speech engine to use the Kinect specific language pack for the culture set on the device (I have a default value but it is omitted here for the sake of simplicity).

RecognizerInfo kinectRecognizer = null;

foreach (RecognizerInfo recognizer in SpeechRecognitionEngine.InstalledRecognizers())

{

string value;

recognizer.AdditionalInfo.TryGetValue("Kinect", out value);

if ("True".Equals(value, StringComparison.OrdinalIgnoreCase) && CultureInfo.CurrentCulture.Name.Equals(recognizer.Culture.Name, StringComparison.OrdinalIgnoreCase))

{

kinectRecognizer = recognizer;

break;

}

}