# starPad

Lecture #4.5: starPad SDK

http://pen.cs.brown.edu/starpad.html



# Architecture

- Framework to facilitate ink recognition
- Provides classes to mimic and extend functionality in *System.Windows.Ink* namespace
  - *InqCanvas*
  - *Stroq*
  - *StroqCollection*

# InqCanvas

- Similar to *InkCanvas*
  - Receives and displays user strokes
  - Stores ink in *Stroqs*
  - Provides different stylus level events, e.g. *StroqCollected*, *StylusOutOfRange*
  - Also provides a *SystemStylusGesture* event
    - Supports gestures such as Tap, Flick
    - **Note**: Some gestures require Windows Vista to work properly



# InqCanvas Example

- Add an *InqCanvas* in XAML
- Capture the *StroqCollected* event
- Capture the *StylusInRange* event
- Capture the *SystemStylusGesture* event

```
<StarPadSDK_Inq:InqCanvas x:Name="inqCanvas"
StroqCollected="inqCanvas_StroqCollected"
StylusInRange="inqCanvas_StylusInRange"
StylusSystemGesture="inqCanvas_StylusSystemGesture"
Background="#00000000"/>
```

# Recognition

- Provided by *MathRecognition* class
  - Online recognition of letters, numbers, symbols
  - Not too good at recognizing complex shapes
  - User can provide a callback function
  - Recognition results are also available from the *MathRecognition* object at all times
- Pass a *StroqCollection* to *MathRecognition*
  - *MathRecognition* knows when a stroke is added or removed from *StroqCollection*



# Recognition Example

- Setting up the recognizer

```
StroqCollection _mathStroqs = new StroqCollection();
MathRecognition _mrec = new MathRecognition(_mathStroqs);
_mrec.EnsureLoaded();
_mrec.ParseUpdated += _mrec_ParseUpdated;
```

- Adding strokes to the collection

```
void inqCanvas_StroqCollected(object sender,
        InqCanvas.StroqCollectedEventArgs e)
{
       _mathStroqs.Add(e.Stroq);
}
```

# Recognition Example – Cont.

- Callback function is called whenever new strokes are added to the recognizer

```
private void _mrec_ParseUpdated(MathRecognition source,
                                Recognition chchanged,
                                bool updateMath) {
       foreach(Parser.Range range in _mrec.Ranges) {
              Expr expr = range.Parse.expr;
              //Do something
       }
}
```

- Recognized expressions can also be manipulated without callback



# Expressions

- Recognized expressions are stored in a tree structure – *Expr*
  - Each level of the tree is organized as [Head, Args]
    - Head = numerical operation (e.g. +,-,..)
    - Args = list of argument Exprs
- Results from recognizer in the form of *Ranges*
  - Each *Range* corresponds to a section of text
    - *Range* has a *Parse*
    - *Parse* has an *Expr*, which can be drawn, executed, converted to MathML

# Expressions – Cont.

- *Expr* is an abstract class
  - Useful subclasses
    - *CompositeExpr*
    - *IntegerNumber* / *DoubleNumber*
    - *LetterSym*
    - *WellKnownSym*
  - Example: -1 is a *CompositeExpr*
    - [Head = -, Args= IntegerNumber (1)]



# Solving Expressions

- starPad provides integration with computational engines, e.g. Mathematica, Maple
- Given an *Expr*
  - If it has variables, but values are known
  - Substitute values and evaluate answer
- Useful Classes
  - *Engine*
  - *Text*

# Using Exprs

- Example: Equation Solver
  - Given an equation
    - Verify if recognized *Expr* is an equation
    - Scan equation for variables
    - Prompt user for values
    - Solve the equation
    - Print answer



# Basic Error Checking

```
//Argument is (Expr expression)

if (!(expression is CompositeExpr)) return ;

CompositeExpr expr = (CompositeExpr)expression;

//confirm that composite expr is an expression
//1-it should have at least two args
//2-lhs should be a variable
//3-head should be an equals sign
if (expr.Head != WellKnownSym.equals ||
   expr.Args.Length < 2 ||
    !(expr.Args[0] is LetterSym))
   return;
```

# Checking For Equations

```
bool isEq = isEquation(composite_expr.Args[1]);

private static bool isEquation(Expr expr)
{
   bool equation = false;
   //if a single LetterSym exists in this part of the expr,
   //it is an equation.
   if (expr is LetterSym)  equation = true;
   else if (expr is CompositeExpr)
   {
         CompositeExpr cexpr = (CompositeExpr)expr;
         foreach (Expr ex in cexpr.Args)
         equation = equation || isEquation(ex);
   }

   return equation;
}
```



# Getting List of Variables

```
List<LetterSym> vars = readVars(composite_expr.Args[1]);

private static List<LetterSym> readVars(Expr expr)
{
   List<LetterSym> list = new List<LetterSym>();
   //every letterSym is a different variable
   if (expr is LetterSym)  list.Add((LetterSym)expr);
   else if (expr is CompositeExpr)
   {
         CompositeExpr cexpr = (CompositeExpr)expr;
         foreach (Expr ex in cexpr.Args)
         {
                  List<LetterSym> pList = readVars(ex);
                  list.AddRange(pList);
         }
   }
   return list;
}
```

# Getting Values

```
private static Hashtable readValues(List<LetterSym> variables)
{
   Hashtable values = new Hashtable();
   string input = "";
   foreach (LetterSym var in variables)
   {
        Console.WriteLine("" + var.Letter + "=");

        /*TODO: read input somehow from the user*/
        double value = double.Parse(input);
        values.Add(var, value);
   }

   return values;
}
```



# Substituting Values

```
private static Expr substitute(Expr equation,
                        List<LetterSym> variables,
                        Hashtable values)
{
   Expr substituted = equation.Clone();
   foreach (LetterSym var in variables)
   {
        Expr original = (Expr)var;
        Expr replacement = Text.Convert("" + (double)values[var]);
        substituted = Engine.Substitute(   substituted,
                                           original,
                                           replacement);
   }

   return substituted;
}
```

# Evaluation

- Evaluation is straightforward

```
Expr result = Engine.Approximate(substituted);
```

- Result is still an *Expr*
  - Have to read answer from the RHS of this *Expr*



# Reading A Number

```
double answer = readDouble(result.Args[1]);

private static double readDouble(Expr num)
{
   if (num is CompositeExpr &&
      ((CompositeExpr)num).Args[0] is IntegerNumber)
   {
      return -1 *
      ((IntegerNumber)((CompositeExpr)num).Args[0]).Num.AsDouble();
   }
   else if (num is IntegerNumber)
      return ((IntegerNumber)num).Num.AsDouble();
   else
      return ((DoubleNumber)num).Num;
}
```

# Getting Alternates

- Need a *Recognition* for each stroke/group
  - *Recognition.alt* can be a *char* or a *string*
  - Get this from *MathRecognition.Charreco.Classification*
- See showSidebarAlts in MathRecoScaffold



# General Format For Alternates

```
foreach (Stroq stroke in strokes)
{
    Recognition recog = recognizer.Charreco.Classify( recognizer.Sim[stroke], true );

    if ( recog != null )
    {
          string alt;
          char temp = recog.alt.Character;

          if ( temp != 0 ) alt = temp.ToString();
          else
          {
                    alt = recog.alt.Word;

                    if ( alt == null )
                    {
                              alt = recog.alt.ToString();
                              alt = alt.Substring( 1, alt.Length - 2 );
                    }
          }
    }
}
```

# Getting Alternates For A Single Recognition

```
private void showSidebarAlts( ICollection<Recognition>
   recogs, StroqCollection stroqs )
{
   …
   if ( recogs.Count == 1 )
   {
        Recognition rr = recogs.Single();

        for ( int i = 0; i < rr.alts.Length; i++ )
        {
               char c = rr.alts[i].Character;

               if ( c != 0 ) { //Alternate is c.ToString() }
               else { //Alternate is rr.alts[i].Word }
        }
   }
}
```



# Setup to Work at Home

- Already done in the lab
- gacutil /i "c:\program files\microsoft sdks\windows\v6.0a\bin\ialoader.dll“
- gacutil /i "C:\Program Files\Reference Assemblies\Microsoft\Tablet PC\v1.7\iawinfx.dll"
- gacutil /i "C:\Program Files\Reference Assemblies\Microsoft\Tablet PC\v1.7\iacore.dll“
- copy "c:\Program Files\FSharp-1.9.6.16\bin\gac\FSharp.PowerPack.Build.Tasks.dll" "C:\Program Files\MSBuild\FSharp\1.0\FSharp.PowerPack.Build.Tasks.dll"

# Readings

- starPad Readme
  - http://pen.cs.brown.edu/starPadSDKDoc/README.html