BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil penelitian dan implementasi yang telah dilakukan sebelumnya, dapat disimpulkan bahwa aplikasi permainan edukasi kolaboratif ini dapat menurunkan biaya infrastruktur secara signifikan, membantu merangsang daya pikir anak-anak terutama siswa/i di sekolah dasar karena lebih interaktif sehingga proses pembelajaran menjadi lebih menarik untuk banyak pengguna.

5.2 Saran

Dari hasil penelitian dan implementasi yang telah dilakukan, disarankan pengembangan sistem dilakukan pada peningkatan user interface yang lebih menarik lagi, penambahan soal dan pemain, sehingga diharapkan dapat diciptakan sistem yang lebih user friendly kepada pengguna, konten yang lebih bervariasi serta interaksi pengguna yang lebih banyak.

DAFTAR PUSTAKA

[1] Plass, Jan. 2010. *A New Way of Learning: Researchers Explore the Allure of Computer Gaming.* http://research.microsoft.com/en-us/collaboration/focus/cs/src_gaming.aspx

[2] MSDN. 2011. *Welcome to MultiPoint Mouse SDK.* http://msdn.microsoft.com/en-us/library/ee906605.aspx

[3] International Data Corp. 2010. PC *Acessories Market is Vibrant*. http://www.boston.com/business/ticker/2010/06/pc_accessories.html

[4] Kentaro,Toyama. 2007. *Technology for Emerging Markets*. Redmond: Microsoft Research.

[5] Metro TV. 2010. *PesonaEdu menjadi Narasumber Kick Andy Show*. http://www.pesonaedu.com/berita.php

[6] Microsoft, Inc. 2009. *Benefits of Windows MultiPoint Mouse SDK*. http://www.microsoft.com/multipoint/mouse-sdk/benefits.aspx

[7] TeknoMind. 2011. *Indonesia Butuh Konten Lokal*. www.teknojurnal.com/2011/01/31/indonesia-butuh-konten-lokal/

[8] Sommerville, Ian. 2010. *Software Engineering.* Boston: Addison Wesley.

Lampiran

Source Code Program

```
namespace Multipoint.Common
{
///<summary>
/// Provides a link between MouseID and Mouse Cursors to ensure
mice
///  maintain their same cursors when other mice attached or
removed
///</summary>
publicclassCursorAssignment
    {
public System.Drawing.Bitmap Cursor;
publicint MouseID;

public CursorAssignment(System.Drawing.Bitmap cursor, int mouseID)
        {
this.Cursor = cursor;
this.MouseID = mouseID;
        }
    }

///<summary>
/// Contains a collection of CursorAssignment objects
///</summary>
publicsealedclassCursorAssignments
    {
privatestaticreadonlyCursorAssignments instance =
newCursorAssignments();

privateCursorAssignment teacherAssignment;

        #region Initialise Cursor Assignments

///<summary>
///  Array of cursorAssignment objects to keep track of cursors
and the mice to which they're assigned
///  Assign MouseID of -1 initially to indicate that there is no
Cursor currently unassigned
///</summary>
privateCursorAssignment[] assignments = newCursorAssignment[]
        {
newCursorAssignment(Properties.Resources.elephant, -1),
newCursorAssignment(Properties.Resources.tiger, -1),
newCursorAssignment(Properties.Resources.hippo, -1),
newCursorAssignment(Properties.Resources.lion, -1),
newCursorAssignment(Properties.Resources.rooster2, -1),
newCursorAssignment(Properties.Resources.boy_01, -1),
newCursorAssignment(Properties.Resources.girl_03, -1),
newCursorAssignment(Properties.Resources.boy_02, -1),
```

```
newCursorAssignment(Properties.Resources.boy_03, -1),
newCursorAssignment(Properties.Resources.boy_04, -1),
newCursorAssignment(Properties.Resources.boy_05, -1),
newCursorAssignment(Properties.Resources.boy_06, -1),
newCursorAssignment(Properties.Resources.boy_07, -1),
newCursorAssignment(Properties.Resources.boy_08, -1),
newCursorAssignment(Properties.Resources.cat, -1),
newCursorAssignment(Properties.Resources.cow, -1),
newCursorAssignment(Properties.Resources.dog, -1),
newCursorAssignment(Properties.Resources.donkey, -1),
newCursorAssignment(Properties.Resources.giraffe, -1),
newCursorAssignment(Properties.Resources.girl_01, -1),
newCursorAssignment(Properties.Resources.girl_02, -1),
newCursorAssignment(Properties.Resources.girl_04, -1),
newCursorAssignment(Properties.Resources.girl_05, -1),
newCursorAssignment(Properties.Resources.girl_06, -1),
newCursorAssignment(Properties.Resources.girl_07, -1),
newCursorAssignment(Properties.Resources.horse, -1),
newCursorAssignment(Properties.Resources.monkey, -1),
newCursorAssignment(Properties.Resources.ox, -1),
newCursorAssignment(Properties.Resources.pig, -1),
newCursorAssignment(Properties.Resources.rabbit, -1),
newCursorAssignment(Properties.Resources.rihino, -1),
newCursorAssignment(Properties.Resources.rooster, -1)
        };

///<summary>
/// Prevents a default instance of the CursorAssignments class
from being created
/// other than the singleton
///</summary>
private CursorAssignments()
        {
this.TeacherAssignment =
newCursorAssignment(Properties.Resources.zebra, -1);
        }

publicstaticCursorAssignments Instance
        {
get
            {
return instance;
            }
        }

///<summary>
/// Gets the mouse that has been assigned to the teacher.
///</summary>
publicCursorAssignment TeacherAssignment
        {
get
            {
returnthis.teacherAssignment;
            }
privateset
            {
```

```
this.teacherAssignment = value;
            }
        }

publicCursorAssignment[] Assignments
        {
get { returnthis.assignments; }
        }

        #endregion
    }
}
```

```
namespace Multipoint.Common
{
using System.Collections.Specialized;
using System.IO;
using System.Windows;
using System.Windows.Media.Imaging;
using Microsoft.MultiPoint.CommonTypes;
using Microsoft.MultiPoint.MousePlugIn;
using Microsoft.MultiPoint.SDK;

///<summary>
/// A helper class that contains various functions related to
MultiPoint functionality
///</summary>
publicstaticclassMouseHelper
    {
///<summary>
/// Helper method that will create a Bitmap image out of a Bitmap
///</summary>
///<param name="b">The bitmap image that has to be
processed</param>
///<returns>The bitmapimage created from the bitmap
parameter</returns>
publicstaticBitmapImage CreateBitmapImage(System.Drawing.Bitmap b)
        {
BitmapImage bmpimg = newBitmapImage();
MemoryStream memStream = newMemoryStream();
            bmpimg.BeginInit();
            b.MakeTransparent(System.Drawing.Color.White);
            b.Save(memStream,
System.Drawing.Imaging.ImageFormat.Png);
            bmpimg.StreamSource = memStream;
            bmpimg.EndInit();
return bmpimg;
        }

///<summary>
/// Check if there are too many mice, if so disable extras
/// TODO: WP1 ? CheckIfTooManyMice
///</summary>
```

```
///<param name="maxPlayers">The maximum number of allowed
players</param>
///<returns>true if too many mice, otherwise false</returns>
publicstaticbool CheckIfTooManyMice(int maxPlayers)
        {
// Too many mice - disable extras
if (MultiPointSDK.Instance.MouseDeviceList.Count > maxPlayers)
            {
// Disable extra mice.
for (int i = maxPlayers; i
<MultiPointSDK.Instance.MouseDeviceList.Count - 1; i++)
                {

((MultiPointMouseDevice)((DeviceInfo)MultiPointSDK.Instance.MouseD
eviceList[i]).DeviceVisual).Visible = false;
                }

returntrue;
            }
else
            {
returnfalse;
            }
        }

///<summary>
/// Checks if there are too few mice connected
/// TODO: WP1 ? CheckIfTooFewMice
///</summary>
///<param name="minPlayers">This is the minimum number of players
required</param>
///<returns>True if too few mice, otherwise False</returns>
publicstaticbool CheckIfTooFewMice(int minPlayers)
        {
if (MultiPointSDK.Instance.MouseDeviceList.Count < minPlayers)
            {
returntrue;
            }
else
            {
returnfalse;
            }
        }

///<summary>
/// Assign cursor to mouse object
///</summary>
///<param name="mouseObject">The mouse to assign a cursor
to</param>
publicstaticvoid AssignCursorToMouse(DeviceInfo mouseObject)
            {
CursorAssignment assignment =
System.Array.Find(CursorAssignments.Instance.Assignments,
delegate(CursorAssignment ca) { return ca.MouseID == -1; });
MultiPointMouseDevice mouseDevice =
(MultiPointMouseDevice)mouseObject.DeviceVisual;
```

```
                mouseDevice.CursorImage =
CreateBitmapImage(assignment.Cursor);
                assignment.MouseID =
int.Parse(mouseObject.DeviceID);
            }

///<summary>
/// Remove cursor from mouse object
///</summary>
///<param name="mouseObject">The mouse to remove the cursor
from</param>
publicstaticvoid RemoveCursorFromMouse(DeviceInfo mouseObject)
        {
int mouseID = int.Parse(mouseObject.DeviceID);
if (CursorAssignments.Instance.TeacherAssignment.MouseID ==
mouseID)
            {
CursorAssignments.Instance.TeacherAssignment.MouseID = -1;
            }
else
            {
CursorAssignment assignment =
System.Array.Find(CursorAssignments.Instance.Assignments,
delegate(CursorAssignment ca) { return ca.MouseID == -1; });
if (assignment != null)
                {
                    assignment.MouseID = -1;
                }
            }
        }

///<summary>
/// Associates the supplied mouse with the teacher.
///</summary>
///<param name="mouseObject">The mouse to assign to the
teacher.</param>
publicstaticvoid AssignTeacherMouse(DeviceInfo mouseObject)
        {
            RemoveCursorFromMouse(mouseObject);

MultiPointMouseDevice mouseDevice =
(MultiPointMouseDevice)mouseObject.DeviceVisual;
            mouseDevice.CursorImage =
CreateBitmapImage(CursorAssignments.Instance.TeacherAssignment.Cur
sor);

int mouseID = int.Parse(mouseObject.DeviceID);
CursorAssignments.Instance.TeacherAssignment.MouseID = mouseID;
        }

///<summary>
/// Excludes the specified mouse devices from being displayed by
MultiPoint.
///</summary>
///<param name="excludedDevices">A list of hardware ids obtained
from device manager to excluded.</param>
```

```
///<remarks>
/// It's possible that there may be devices present on the system
whose drivers are incorrectly
/// indentifying themselves as mouse devices, or other pointing
devices which are currently
/// disconnected (such as a Wacom tablet), that will appear to
Windows as connected
/// mouse devices. This will cause MultiPoint to draw additional
and unwanted pointers.
/// During the creation of this sample the keyboard driver
registered itself as a mouse device
/// with Windows and needed to be disabled.<para/>
/// Devices can be indentified by checking <see
cref="DeviceInfo.DeviceName"/>, this property
/// contains a concatenation of the driver details, one of these
being the hardware id of the device.
/// The hardware id can be found using the device manager in
Windows, by expanding the
/// 'Mice and other pointing devices" node and right clicking on
the mouse device. This will
/// display the properties dialog for mouse, the hardware id can
be found in the 'Details' tab
///</remarks>
publicstaticvoid ExcludeDevices(StringCollection excludedDevices)
        {
foreach (DeviceInfo device
inMultiPointSDK.Instance.MouseDeviceList)
            {
foreach (string excludedDevice in excludedDevices)
                {
if (device.DeviceName.Contains(excludedDevice))
                    {

((MultiPointMouseDevice)device.DeviceVisual).Visible = false;
break;
                    }
                }
            }
        }
    }
}
```

```
namespace Multipoint.QuizSample
{
///<summary>
/// Class that maintains the current time shown on the clock.
///</summary>
publicpartialclassClock
    {
///<summary>
/// A value indicating whether the timer is complete
```

```
///</summary>
privatebool completed = false;

privateint hr, min, sec;

///<summary>
/// Initializes a new instance of the Clock class
///</summary>
///<param name="hr1">How many hours</param>
///<param name="min1">How many minutes</param>
///<param name="sec1">How many seconds</param>
public Clock(string hr1, string min1, string sec1)
        {
this.Completed = false;
this.hr = int.Parse(hr1);
this.min = int.Parse(min1);
this.sec = int.Parse(sec1);
        }

///<summary>
/// Gets or sets a value indicating whether the timer is complete
///</summary>
publicbool Completed
        {
get
            {
return completed;
            }

set
            {
this.completed = value;
            }
        }

///<summary>
/// Gets Current seconds value
///</summary>
publicint Sec
        {
get { returnthis.sec; }
        }

///<summary>
/// Gets Current minutes value
///</summary>
publicint Min
        {
get { returnthis.min; }
        }

///<summary>
/// Gets Current hours value
///</summary>
publicint Hr
        {
```

```
get { returnthis.hr; }
        }

///<summary>
/// Decrements the Clock by 1 second
///</summary>
publicvoid Decrement()
        {
// Decrement the hours/mins/secs
if (this.sec > 0)
            {
this.sec -= 1;
            }
else
            {
// sec = 0
if (this.min > 0)
                {
this.min -= 1;
this.sec = 59;
                }
else
                {
// min = 0
if (this.hr > 0)
                    {
this.hr -= 1;
this.min = 59;
this.sec = 59;
                    }
else
                    {
// hr = 0
this.Completed = true;
PlayerManager.Instance.HandleActivityCompletion();
                    }
                }
            }
        }

///<summary>
/// How much time is left
///</summary>
///<returns>string representation of the time remaining</returns>
publicstring TimeLeft()
        {
returnthis.hr.ToString() + ":" + this.min.ToString() + ":" +
this.sec.ToString();
        }
    }
}
```

```
namespace Multipoint.QuizSample
{
```

```
using Microsoft.MultiPoint.CommonTypes;

///<summary>
/// Player Class maintains info an individual player and its
connected mouse,
/// including the MouseObject, and its performance in the activity
///</summary>
publicclassPlayer
    {
///<summary>
/// The actual MouseObject attached to the player.
///</summary>
privateDeviceInfo mouseObject;

///<summary>
/// Number of questions answered.
///</summary>
privateint questionsAnswered;

///<summary>
/// Number of questions answered correctly
///</summary>
privateint questionsAnsweredCorrectly;

///<summary>
/// Initializes a new instance of the Player class
///</summary>
///<param name="mouseObject">The Mouse DeviceInfo object to attach
to this player</param>
public Player(DeviceInfo mouseObject)
        {
if (mouseObject != null)
            {
this.MouseObject = mouseObject;
this.QuestionsAnswered = 0;
this.QuestionsAnsweredCorrectly = 0;
            }
        }

///<summary>
/// Gets or sets The actual MouseObject attached to the player
///</summary>
publicDeviceInfo MouseObject
        {
get
            {
return mouseObject;
            }

set
            {
this.mouseObject =  value;
            }
        }

///<summary>
```

```
/// Gets or sets Number of questions answered
///</summary>
publicint QuestionsAnswered
        {
get
            {
return questionsAnswered;
            }

set
            {
this.questionsAnswered = value;
            }
        }

///<summary>
/// Gets or sets Number of questions answered correctly
///</summary>
publicint QuestionsAnsweredCorrectly
        {
get
            {
return questionsAnsweredCorrectly;
            }

set
            {
this.questionsAnsweredCorrectly = value;
            }
        }

///<summary>
/// Gets The ID of the MouseObject attached to this player
///</summary>
publicint MouseId
        {
get
            {
returnthis.MouseObject.ID;
            }
        }

///<summary>
/// Gets The DeviceID of the MouseObject attached to this player
///</summary>
publicstring MouseDeviceId
        {
get
            {
returnthis.MouseObject.DeviceID;
            }
        }
    }
}
```

```
namespace Multipoint.QuizSample
{
using System;
using System.Collections.Generic;
using System.Windows;
using Microsoft.MultiPoint.CommonTypes;
using Microsoft.MultiPoint.MousePlugIn;
using Microsoft.MultiPoint.SDK;

///<summary>
/// Singleton class that maintains the list of players,
///  and manages all player functionality
///</summary>
publicsealedclassPlayerManager
    {
        #region Properties

///<summary>
/// maximum number of mice allowed
///</summary>
publicconstint MaximumPlayers = 6;

///<summary>
/// minimum number of mice allowed
///</summary>
publicconstint MinimumPlayers = 1;

privatestaticreadonlyPlayerManager instance = newPlayerManager();

///<summary>
/// List of PlayerRegion (user control) objects
///</summary>
privateIList<PlayerRegion> playerRegionList =
newList<PlayerRegion>(MaximumPlayers);

///<summary>
/// List of Player objects
///</summary>
privateIList<Player> playerList = newList<Player>(MaximumPlayers);

///<summary>
/// Indicates whether the mice are currently frozen or not
///</summary>
privatebool miceFrozen = false;

        #endregion

private PlayerManager()
        {
        }

        #region Singleton Class Implementation

publicstaticPlayerManager Instance
        {
get
```

```
            {
return instance;
            }
        }

        #endregion

///<summary>
/// Gets the player region list
///</summary>
publicIList<PlayerRegion> PlayerRegionList
        {
get
            {
returnthis.playerRegionList;
            }
        }

        #region Public Functions

///<summary>
/// Initialize the connected mice, create player objects,
///  assign cursor to mouse and add player regions to canvas
///  Called by WindowMain
///</summary>
///<param name="wm">Reference to WindowMain Instance</param>
publicvoid InitializeConnectedMice(WindowMain wm)
        {
try
            {
for (int i = 0; i <MultiPointSDK.Instance.MouseDeviceList.Count;
i++)
                {
DeviceInfo mouseObject =
(DeviceInfo)MultiPointSDK.Instance.MouseDeviceList[i];
Player p = this.CreateNewPlayerAndAddToList(mouseObject);

Common.MouseHelper.AssignCursorToMouse(mouseObject);
                    wm.AddPlayerRegionToCanvas(p);
                }
            }
catch (Exception ex)
            {
                wm.AddToTextBox("There was a problem initializing
mice: " + ex.Message);
            }
        }

///<summary>
/// Instantiates a new player object, and adds it to the player
list
/// Adds the mouseObject passed in to the player object
///</summary>
///<param name="mouseObject">The mouse object to create a new
player for</param>
```

```
///<returns>The newly created player (null if player not
created)</returns>
publicPlayer CreateNewPlayerAndAddToList(DeviceInfo mouseObject)
        {
try
            {
Player pl = newPlayer(mouseObject);
this.playerList.Add(pl);
return pl;
            }
catch
            {
returnnull;
            }
        }

///<summary>
/// Get player region corresponding to mouseId passed in
///</summary>
///<param name="mouseId">The mouseId of the player</param>
///<returns>The player region for the specified mouseId</returns>
publicPlayerRegion GetPlayerRegionByDeviceID(string mouseId)
        {
if (mouseId.Length >= 0)
            {
for (int i = 0; i <this.PlayerRegionList.Count; i++)
                {
if (mouseId == this.PlayerRegionList[i].MouseDevice.DeviceID)
                    {
returnthis.PlayerRegionList[i];
                    }
                }

returnnull;
            }
else
            {
returnnull;
            }
        }

///<summary>
/// Get the player object that corresponds to the mouse that was
clicked
///</summary>
///<param name="mouseId">The MouseId to use to retrieve the Player
by</param>
///<returns>The player based on the MouseId</returns>
publicPlayer GetPlayerByDeviceID(int mouseId)
        {
if (mouseId >= 0)
            {
for (int i = 0; i <this.playerList.Count; i++)
                {
if (mouseId == this.playerList[i].MouseId)
                    {
```

```
returnthis.playerList[i];
                    }
                }

returnnull;
            }
else
            {
returnnull;
            }
        }

///<summary>
/// Adds the PlayerRegion object passed in as a parameter to the
PlayerRegionList collection
///</summary>
///<param name="region">The region to add to the list</param>
publicvoid AddPlayerToRegionList(PlayerRegion region)
        {
this.PlayerRegionList.Add(region);
        }

///<summary>
/// Removes the PlayerRegion object passed in as a parameter from
the PlayerRegionList collection
///</summary>
///<param name="region">The region to remove from the list</param>
publicvoid RemovePlayerFromRegionList(PlayerRegion region)
        {
this.PlayerRegionList.Remove(region);
        }

///<summary>
/// Adds the PlayerRegion object passed in as a parameter to the
PlayerRegionList collection
///</summary>
///<param name="mouseId">The mouseId of the player to be
removed</param>
publicvoid RemovePlayerFromRegionList(string mouseId)
        {
this.PlayerRegionList.Remove(this.GetPlayerRegionByDeviceID(mouseI
d));
        }

///<summary>
/// Wrapper function for CheckIfTooManyMice() function
///</summary>
///<returns>True if too many, otherwise false</returns>
publicbool CheckIfTooManyMice()
        {
return Common.MouseHelper.CheckIfTooManyMice(MaximumPlayers);
        }

///<summary>
/// Wrapper function for CheckIfTooFewMice() function
///</summary>
```

```
///<returns>True if too few, otherwise false</returns>
publicbool CheckIfTooFewMice()
        {
return Common.MouseHelper.CheckIfTooFewMice(MinimumPlayers);
        }

///<summary>
/// If more mice are allowed, creates new player object, assigns
mouse to player, and returns player,
/// else returns null.
///</summary>
///<param name="e">Device information</param>
///<returns>Player object</returns>
publicPlayer HandleMouseArrival(DeviceNotifyEventArgs e)
        {
if (!Common.MouseHelper.CheckIfTooManyMice(MaximumPlayers))
            {
Player p = this.CreateNewPlayerAndAddToList(e.DeviceInfo);

Common.MouseHelper.AssignCursorToMouse(e.DeviceInfo);

QuizMaster.Instance.ActivityComplete = this.CheckIfTooFewMice();

return p;
            }
else
            {
returnnull;
            }
        }

///<summary>
/// Remove player from PlayerList. If there are now too few mice,
the activity is set to complete (paused)
///</summary>
///<param name="e">Device information</param>
publicvoid HandleMouseRemoval(DeviceNotifyEventArgs e)
        {
this.playerList.Remove(this.GetPlayerByDeviceID(e.DeviceInfo.ID));

Common.MouseHelper.RemoveCursorFromMouse(e.DeviceInfo);
QuizMaster.Instance.ActivityComplete = this.CheckIfTooFewMice();
        }

///<summary>
/// Monitors movement of the mouse object, and ensures that it
stays within the bounds of the User Control
///</summary>
///<param name="wm">Reference to the main WindowMain class</param>
///<param name="e">Mouse information</param>
publicvoid HandleMouseMovement(WindowMain wm, RoutedEventArgs e)
        {
/// receive the RoutedEventArgs and cast it to a
MultiPointMouseEventArgs object
MultiPointMouseEventArgs args = e asMultiPointMouseEventArgs;
if (args == null)
```

```
            {
return;
            }

/// get the DeviceID of the mouse device who's movement is
currently being tracked,
/// and assign it to a local string variable, deviceID
string deviceID = args.DeviceInfo.DeviceID;

/// Since we'll need to interact with the visual representation of
the Mouse Cursor,
/// we'll get the DeviceVisual from the arguments, cast it to a
MultiPointMouseDevice
/// object, and assign it to a local variable (deviceVisual)
MultiPointMouseDevice deviceVisual = args.DeviceInfo.DeviceVisual
asMultiPointMouseDevice;

/// iterate through the collection of PlayerRegions in the
PlayerRegionList, and find
/// the PlayerRegion to which the currently moving mouse belongs
for (int i = 0; i <this.PlayerRegionList.Count; i++)
            {
if (this.PlayerRegionList[i].MouseDevice.DeviceID == deviceID)
                {
/// get the current screenPosition of the mouse as a Point object
Point screenPosition = deviceVisual.GetPosition();

/// To discover the boundaries of the Player Region, we'll create
a Point object
/// containing the top-left co-ordinates of the PlayerRegion in
question, by calling
/// the PointToScreen function of the UserControl (using the
reference to WindowMain
/// that was passed in to the function as parameter wm)
Point point =
wm.MainWindowWrap.Children[i].PointToScreen(newPoint(0.0, 0.0));

/// build a rectangle by calculating the bottom and right
positions using the
/// PlayerRegions ActualWidth and ActualHeight properties
Rect rect = newRect(
                                                point.X,
                                                point.Y,
this.PlayerRegionList[i].ActualWidth,
this.PlayerRegionList[i].ActualHeight);

                    #if (DEBUG)

string tlX = rect.TopLeft.X.ToString();
if (tlX.IndexOf(".") > -1)
                        {
                            tlX = tlX.Remove(tlX.IndexOf("."));
                        }

string tlY = rect.TopLeft.Y.ToString();
if (tlY.IndexOf(".") > -1)
```

```
                        {
                            tlY = tlY.Remove(tlY.IndexOf("."));
                        }

string brX = rect.BottomRight.X.ToString();
if (brX.IndexOf(".") > -1)
                        {
                            brX = brX.Remove(brX.IndexOf("."));
                        }

string brY = rect.BottomRight.Y.ToString();
if (brY.IndexOf(".") > -1)
                        {
                            brY = brY.Remove(brY.IndexOf("."));
                        }

// Debug output:
this.PlayerRegionList[i].rect.Text = string.Format("Rectangle: TL
{0},{1} BR {2},{3}", tlX, tlY, brX, brY);
this.PlayerRegionList[i].coords.Text = string.Format(" [ X: {0} Y:
{1} ]", screenPosition.X.ToString(), screenPosition.Y.ToString());

                    #endif

/// It's a good idea to reduce the size of the rectangle, so that
the user is forced
/// to stay "within" the bounds of the rectangle and cannot reach
the edges.  We'll
/// reduce the dimensions of the rectangle by 10 units in all
directions using the
/// Inflate function
                    rect.Inflate((double)-10.0, (double)-10.0);

/// test the screenPosition of the mouse against the allowed
screen dimensions of
/// the rect object, and if they fall outside the allowed regions,
we'll update the
/// `corresponding screenPosition co-ordinate back to the boundary
of the rect
if (screenPosition.X < rect.Left)
                    {
                        screenPosition.X = rect.Left;
                    }

if (screenPosition.X > rect.Right)
                    {
                        screenPosition.X = rect.Right;
                    }

if (screenPosition.Y < rect.Top)
                    {
                        screenPosition.Y = rect.Top;
                    }

if (screenPosition.Y > rect.Bottom)
                    {
```

```
                        screenPosition.Y = rect.Bottom;
                    }

/// update the mouse cursor's position to the new screenPosition

deviceVisual.SetPosition((int)screenPosition.X,
(int)screenPosition.Y);
                }
            }
        }

///<summary>
/// Complete the activity and display activity completion
information
///</summary>
///<remarks>
/// Sets ActivityComplete to true on the QuizMaster class,
retrieves players and sorts
/// by score (correct answers), then displays a notification of
the winner
///</remarks>
publicvoid HandleActivityCompletion()
        {
QuizMaster.Instance.ActivityComplete = true;

/// Retrieve list of players, ordered by
QuestionsAnsweredCorrectly descending
List<PlayerRegion> players =
newList<PlayerRegion>(this.PlayerRegionList);
            players.Sort(delegate(PlayerRegion
region1,PlayerRegion region2){return
region1.Player.QuestionsAnsweredCorrectly>region2.Player.Questions
AnsweredCorrectly ? -1 : 1;});

/// Loop through the PlayerRegionList and call QuizCompleted on
each,
///  indicating the first in the list as the winner
int i = 0;
int winningScore = -1;

foreach (PlayerRegion pr in players)
            {
if (i == 0)
                {
                    winningScore =
(pr.Player.QuestionsAnsweredCorrectly > 0) ?
pr.Player.QuestionsAnsweredCorrectly : -1;
                }


pr.QuizCompleted(pr.Player.QuestionsAnsweredCorrectly ==
winningScore);
                i++;
            }
        }
```

```
///<summary>
/// Disable movement of the mice, and pause the activity
///</summary>
publicvoid ToggleMiceFrozen()
        {
this.miceFrozen = !this.miceFrozen;

for (int i = 0; i <this.PlayerRegionList.Count; i++)
            {
this.PlayerRegionList[i].MPMouseDevice.DisableMovement =
this.miceFrozen;
            }

// Pause the timer
QuizMaster.Instance.ActivityComplete = this.miceFrozen;
        }

        #endregion
    }
}
```

```
namespace Multipoint.QuizSample
{
using System;
using System.Xml;
using System.Collections.Generic;

///<summary>
/// singleton class that maintains the list of quiz questions and
returns a
///  new question to PlayerManager.cs when requested
///</summary>
publicsealedclassQuizMaster
    {
        #region Properties

///<summary>
/// A value indicating whether or not the activity is complete
(true) or active (false)
///</summary>
privatebool activityComplete;

privatestaticreadonlyQuizMaster instance = newQuizMaster();

///<summary>
/// List of quiz question objects
///</summary>
privateList<QuizQuestion> quizQuestionList =
newList<QuizQuestion>();

        #endregion
```

```
///<summary>
/// Prevents a default instance of the QuizMaster class from being
created - loads quiz questions into the QuestionsList collection
///</summary>
private QuizMaster()
        {
/// Load contents of the quiz qestions xml file
this.PopulateQuestionsListFromXML();
        }

publicstaticQuizMaster Instance
        {
get { return instance; }
        }

///<summary>
/// Gets or sets a value indicating whether or not the activity is
complete (true) or active (false)
/// Can be  used to pause the activity too
///</summary>
publicbool ActivityComplete
        {
get
            {
return activityComplete;
            }

set
            {
this.activityComplete = value;
            }
        }

        #region Public Functions

///<summary>
/// Loops through all visible regions and displays a question on
each
///</summary>
publicvoid DisplayInitialQuestions()
        {
for (int i = 0; i <PlayerManager.Instance.PlayerRegionList.Count;
i++)
            {
QuizQuestion q = this.GetQuestionFromQuestionList(true);
PlayerManager.Instance.PlayerRegionList[i].ShowQuestion(q.Question
, q.Answers, q.CorrectAnswer);
                q.Asked = true;
            }
        }

///<summary>
/// queries the QuizQuestionList and returns a question
///</summary>
///<param name="random">randomize the returned question
order?</param>
```

```
///<returns>a QuizQuestion object</returns>
publicQuizQuestion GetQuestionFromQuestionList(bool random)
        {
QuizQuestion q = null;

if (random)
            {
int count = this.quizQuestionList.FindAll(delegate(QuizQuestion
qq) { return qq.Asked == false; }).Count;

if (count > 0)
                {
List<QuizQuestion> unasked =
quizQuestionList.FindAll(delegate(QuizQuestion qq) { return
qq.Asked == false; });
                    q = unasked[this.RandomNumber(0, count)];
                }
            }
else
            {
                q = quizQuestionList.Find( delegate(QuizQuestion
qq){ return qq.Asked == false; });
            }

return q;
        }

        #endregion

        #region Private Functions

///<summary>
/// Load contents of the quiz qestions xml file
///</summary>
privatevoid PopulateQuestionsListFromXML()
        {
XmlDocument quizDoc = newXmlDocument();
            quizDoc.Load("Resources/QuizFiles/Quiz1.xml");

/// Load contents of the quiz qestions xml file into
QuizQuestionList:
this.quizQuestionList = newList<QuizQuestion>();
XmlNodeList list = quizDoc.GetElementsByTagName("Question");
for (int i = 0; i < list.Count; i++)
            {
XmlNode question = list[i];
XmlElement xquestion = question asXmlElement;
if (xquestion != null)
                {
QuizQuestion q = newQuizQuestion();
List<string> answers = newList<string>();
foreach (XmlNode qcontent in xquestion.ChildNodes)
                    {
XmlElement xcontent = qcontent asXmlElement;
if (xcontent != null)
                        {
```

```
switch(xcontent.Name)
                                {
case"Content":
                                        q.Question =
xcontent.InnerText;
break;
case"Tip":
                                        q.Tip = xcontent.InnerText;
break;
case"Options":
foreach (XmlNode qoption in xcontent.ChildNodes)
                                        {
XmlElement xoption = qoption asXmlElement;
if (xoption != null)
                                            {

answers.Add(xoption.InnerText);
string iscorrect = xoption.GetAttribute("correct");
if (!string.IsNullOrEmpty(iscorrect)
&&string.CompareOrdinal(iscorrect, "true") == 0)
                                                {
                                                    q.CorrectAnswer =
xoption.InnerText;
                                                }
                                            }
                                        }

break;
                                }
                            }
                        }

                        q.Answers = answers.ToArray();
this.quizQuestionList.Add(q);
                    }
                }
            }

///<summary>
/// Generates a random integer
///</summary>
///<param name="min">The minimum value</param>
///<param name="max">The maximum value</param>
///<returns>A random integeter between min and max</returns>
privateint RandomNumber(int min, int max)
            {
Random random = newRandom();
return random.Next(min, max);
            }

            #endregion
        }
}
```

```
namespace Multipoint.QuizSample
{
///<summary>
/// A simple class representing a single quiz question
///</summary>
publicclassQuizQuestion
    {
///<summary>
/// The text of the question.
///</summary>
privatestring question;

///<summary>
/// An array of answers
///</summary>
privatestring[] answers;

///<summary>
/// Corresponds to index in Answers[] that is correct.
///</summary>
privatestring correctAnswer;

///<summary>
/// A tip for the question.
///</summary>
privatestring tip;

///<summary>
/// A value indicating whether the question as asked so it doesn't
get asked again.
///</summary>
privatebool asked;

public QuizQuestion()
        {
        }

///<summary>
/// Gets or sets The text of the question
///</summary>
publicstring Question
        {
get
            {
return question;
            }

set
            {
this.question = value;
            }
        }

///<summary>
/// Gets or sets An array of answers
///</summary>
```

```
publicstring[] Answers
        {
get
            {
return answers;
            }

set
            {
this.answers = value;
            }
        }

///<summary>
/// Gets or sets Corresponds to index in Answers[] that is correct
///</summary>
publicstring CorrectAnswer
        {
get
            {
return correctAnswer;
            }

set
            {
this.correctAnswer = value;
            }
        }

///<summary>
/// Gets or sets A tip for the question.  Currently not used
///</summary>
publicstring Tip
        {
get
            {
return tip;
            }

set
            {
this.tip = value;
            }
        }

///<summary>
/// Gets or sets a value indicating whether the question as asked
so it doesn't get asked again
///</summary>
publicbool Asked
        {
get
            {
return asked;
            }
```

```
set
            {
this.asked = value;
            }
        }
    }
}
```

```
namespace Multipoint.QuizSample
{
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using Microsoft.MultiPoint.CommonTypes;
using Microsoft.MultiPoint.Controls;
using Microsoft.MultiPoint.MousePlugIn;

///<summary>
/// Interaction logic for PlayerRegion.xaml
///<remarks>
/// Implements IMultiPointMouseEvents Interface to enable it to
interact with the multipoint mouse devices
///  (in this case listening to the MouseMove event).
///</remarks>
///</summary>
publicpartialclassPlayerRegion : UserControl,
IMultiPointMouseEvents
    {
///<summary>
/// The mouse device
///</summary>
privateDeviceInfo mouseDevice;

///<summary>
/// The player
///</summary>
privatePlayer player;

///<summary>
/// The MultiPoint mouse device
///</summary>
privateMultiPointMouseDevice mpMouseDevice;

        #region Constructor

///<summary>
/// Initializes a new instance of the PlayerRegion class
///</summary>
///<remarks>
/// Calls InitializeComponent, assigns the Player object passed in
as a parameter to a local
/// object (myPlaner) for use later, gets the MouseObject from the
player and assigns to local
```

```
/// object (MouseDevice), gets the MultiPointMouseDevice from the
MouseDevice and assigns to
/// a local object (MPMouseDevice), initializes the SelectedOption
value to -1, and sets up
/// the PlayerRegion_Loaded event handler
///</remarks>
///<param name="player">Player object to be associated with the
region</param>
public PlayerRegion(Player player)
        {
            InitializeComponent();
this.Player = player;
this.MouseDevice = this.Player.MouseObject;
this.MPMouseDevice = this.MouseDevice.DeviceVisual
asMultiPointMouseDevice;
this.Loaded += newRoutedEventHandler(this.PlayerRegion_Loaded);
        }

        #endregion

        #region IMultiPointMouseEvents Interface Methods

#pragma warning disable 0067

publiceventRoutedEventHandler MultiPointMouseDownEvent;

publiceventRoutedEventHandler MultiPointMouseEnterEvent;

publiceventRoutedEventHandler MultiPointMouseLeaveEvent;

publiceventRoutedEventHandler MultiPointMouseLeftButtonDownEvent;

publiceventRoutedEventHandler MultiPointMouseLeftButtonUpEvent;

publiceventRoutedEventHandler MultiPointMouseMoveEvent;

publiceventRoutedEventHandler MultiPointMouseRightButtonDownEvent;

publiceventRoutedEventHandler MultiPointMouseRightButtonUpEvent;

publiceventRoutedEventHandler MultiPointMouseUpEvent;

publiceventRoutedEventHandler MultiPointMouseWheelEvent;

publiceventRoutedEventHandler MultiPointPreviewMouseDownEvent;

publiceventRoutedEventHandler
MultiPointPreviewMouseLeftButtonDownEvent;

publiceventRoutedEventHandler
MultiPointPreviewMouseLeftButtonUpEvent;

publiceventRoutedEventHandler MultiPointPreviewMouseMoveEvent;

publiceventRoutedEventHandler
MultiPointPreviewMouseRightButtonDownEvent;
```

```
publiceventRoutedEventHandler
MultiPointPreviewMouseRightButtonUpEvent;

publiceventRoutedEventHandler MultiPointPreviewMouseUpEvent;

publiceventRoutedEventHandler MultiPointPreviewMouseWheelEvent;

#pragma warning restore 0067

        #endregion

        #region Properties

///<summary>
/// Gets or sets the mouse device
///</summary>
publicDeviceInfo MouseDevice
        {
get
            {
return mouseDevice;
            }

set
            {
this.mouseDevice = value;
            }
        }

///<summary>
/// Gets or sets the player
///</summary>
publicPlayer Player
        {
get
            {
return player;
            }

set
            {
this.player = value;
            }
        }

///<summary>
/// Gets or sets the MultiPoint mouse device
///</summary>
publicMultiPointMouseDevice MPMouseDevice
        {
get
            {
return mpMouseDevice;
            }
```

```
set
            {
this.mpMouseDevice = value;
            }
        }

        #endregion

        #region Public Functions

///<summary>
/// Displays a new question and answers on the panel
///</summary>
///<param name="questionContent">The text of the question</param>
///<param name="answers">Array of answers</param>
///<param name="correctAnswer">The correct answer</param>
publicvoid ShowQuestion(string questionContent, string[] answers,
string correctAnswer)
        {
if ((questionContent.Trim().Length > 0)
&& (answers.Length >= 2)
&& (!string.IsNullOrEmpty(answers[0]))
&& (!string.IsNullOrEmpty(answers[1]))
&& (correctAnswer.Trim().Length > 0))
            {
this.quizAnswers.Children.Clear();

this.quizQuestion.Text = questionContent;
for (int i = 0; i < answers.Length; i++)
                {
if (!string.IsNullOrEmpty(answers[i]))
                    {
this.AddQuizAnswerButton(answers[i],
string.Equals(answers[i].Trim().ToUpper(),
correctAnswer.Trim().ToUpper()));
                    }
                }
            }
        }

///<summary>
/// Quiz is finished - hide questions and answers, and display
winner details
///</summary>
///<param name="winner">Whether the player was the winner or
not</param>
///<remarks>
/// Called by PlayerManager/ Timer
///</remarks>
publicvoid QuizCompleted(bool winner)
        {
// Hide the quiz answer options
this.quizAnswers.Children.Clear();

this.quizQuestion.Text = "Kuis Selesai";
```

```
if (winner)
            {
// Show the winner message
this.quizWinner.Visibility = Visibility.Visible;
            }

// Hide the question result textblock
this.quizQuestionResult.Visibility = Visibility.Hidden;
        }

        #endregion

        #region Private Functions

///<summary>
/// Moves the mouse to a position around the middle of the user
control
///</summary>
privatevoid ResetMouse()
        {
this.UpdateLayout();
if (this.MouseDevice != null)
            {
Rect rect = newRect(this.PointToScreen(newPoint(0.0, 0.0)),
newSize(this.ActualWidth, this.ActualHeight));
this.MPMouseDevice.SetPosition((int)((rect.Left + rect.Right) /
2.0), (int)((rect.Top + rect.Bottom) / 2.0));
            }
        }

///<summary>
/// Adds a button to the quizAnswers WrapPanel, and sets up
Multipoint mouse button handlers
///</summary>
///<param name="answer">The text of the answer to display on the
button</param>
///<param name="correctAnswer">Represents if the button contains
the correct answer or not</param>
privatevoid AddQuizAnswerButton(string answer, bool correctAnswer)
        {
            Microsoft.MultiPoint.Controls.
MultiPointButton m = newMultiPointButton();

            m.Style = (Style)FindResource("quizAnswerButton");
            m.Content = answer;
if (correctAnswer)
            {
                m.Name = "Correct";
            }

this.quizAnswers.Children.Add(m);

// Define MultiPoint mouse button handlers.
            m.MultiPointClick +=
newRoutedEventHandler(this.MultiPointButton_MultiPointClick);
        }
```

```
///<summary>
/// Determines if the answer supplied is correct or not, and
displays the result accordingly
///</summary>
///<param name="isCorrect">True if correct</param>
///<param name="style">The style to use to display the
answer</param>
privatevoid ShowAnswerResult(bool isCorrect, string style)
        {
            quizQuestionResult.Text = isCorrect ? "Benar" :
"Salah";
            quizQuestionResult.Style = (Style)FindResource(style);
if (isCorrect)
            {
this.Player.QuestionsAnsweredCorrectly += 1;
            }
        }

///<summary>
/// Hides the answer result from the screen
///</summary>
privatevoid HideAnswerResult()
        {
            quizQuestionResult.Visibility = Visibility.Hidden;
        }

///<summary>
/// Gets and displays a new question from the question list
///</summary>
///<remarks>
/// If there are no questions remaining, the
HandleActivityCompletion function on
/// PlayerManager is called.
///</remarks>
privatevoid ShowNewQuestion()
        {
QuizQuestion q =
QuizMaster.Instance.GetQuestionFromQuestionList(true);
if (q != null)
            {
                q.Asked = true;
this.ShowQuestion(q.Question, q.Answers, q.CorrectAnswer);
            }
else
            {
// questions have run out - end the activity
PlayerManager.Instance.HandleActivityCompletion();
            }
        }

        #endregion

        #region Event Handlers

///<summary>
```

```
/// Event handler that fires after the user control has loaded
///</summary>
///<remarks>
/// Sets regionHeader text and image, and calls ShowNewQuestion
and ResetMouse
///</remarks>
///<param name="sender">The sender of the event</param>
///<param name="e">Event information</param>
privatevoid PlayerRegion_Loaded(object sender, RoutedEventArgs e)
        {
this.regionHeader.Text = string.Format("Pemain #{0}",
(MouseDevice.ID + 1).ToString());
            regionHeaderImage.Source =
((MultiPointMouseDevice)((DeviceInfo)MouseDevice).DeviceVisual).Cu
rsorImage;

this.ShowNewQuestion();
this.ResetMouse();
        }

///<summary>
/// Event handler for Application buttons
///</summary>
///<remarks>
/// Gets device specific information from the RoutedEventArgs by
casting to MultiPointMouseArgs,
/// sets the current variable to the device specific ID of the
mouse device that clicked the
/// button, and if that mouse device is the one assigned to the
Player Region, it continues on to:
///  incremenet the QuestionsAnswered property by 1, check if the
answer is correct or not, updates
///  the quizScore accordingly, and finally calls ShowNewQuestion
function.
///</remarks>
///<param name="sender">The Button Controls</param>
///<param name="e">The routed event arguments</param>
privatevoid MultiPointButton_MultiPointClick(object sender,
RoutedEventArgs e)
        {
MultiPointButton multipointbutton = (MultiPointButton)sender;

// Check whether the button has already been clicked
if (multipointbutton.Content != null)
            {
// Convert the RoutedEventArgs to MultiPointMouseArgs in order to
determine device specific ID.
MultiPointMouseEventArgs multipointargs =
(MultiPointMouseEventArgs)e;

//// Set current to device specific ID - which device clicked the
button?
int current = multipointargs.DeviceInfo.ID;

// Check if the mouse device that clicked the button is the device
allocated to this control
```

```
if (this.MouseDevice.ID == current)
                    {
if (this.Player != null)
                        {
this.Player.QuestionsAnswered += 1;

if (string.Equals(multipointbutton.Name, "Correct"))
                            {
this.ShowAnswerResult(true, "quizQuestionResultStyle_Correct");
                            }
else
                            {
this.ShowAnswerResult(false, "quizQuestionResultStyle_Wrong");
                            }

this.quizScore.Text = string.Format(
"{0}benar dari {1}soal",
this.Player.QuestionsAnsweredCorrectly.ToString(),
this.Player.QuestionsAnswered.ToString());
this.ShowNewQuestion();
                        }
                    }
                }
            }

            #endregion
        }
}
```

```
namespace Multipoint.QuizSample
{
using System;
using System.Windows;
using System.Windows.Controls;

///<summary>
/// Interaction logic for Timer.xaml
///</summary>
///<remarks>
/// User Control that displays a timer on the screen.
///</remarks>
publicpartialclassTimer : UserControl
    {
///<summary>
/// Specify whether or not the timer should display a number for
hour
///</summary>
privatebool showHour = false; // Show the hour on the timer?

///<summary>
/// This is an instance of the class which keeps track of the time
being displayed
///</summary>
```

```
privateClock clk;

///<summary>
/// The DispatcherTimer instance
///</summary>
private System.Windows.Threading.DispatcherTimer timerClock;

///<summary>
/// Initializes a new instance of the Timer class
///</summary>
///<remarks>
/// Creates a new instance of DispatcherTimer, sets the TimeSpan
(currently 1 second)
/// creates a new instance of Clock class (specifying 1 minute)
and sets up the
/// handler to manage TimerClock tick events.
///</remarks>
public Timer()
        {
            InitializeComponent();

this.timerClock = new System.Windows.Threading.DispatcherTimer();
this.timerClock.Interval = newTimeSpan(0, 0, 1);
this.timerClock.IsEnabled = true;
this.clk = newClock("0", "5", "0");
this.timerClock.Tick += newEventHandler(this.TimerClock_Tick);
        }

///<summary>
/// The TimerClock Tick Event handler
///</summary>
///<remarks>
/// First ensures that the activity is not yet complete, then
decrements the Clock object
/// and updates the display of the clock. If the completed
property on the Clock object
/// is true, the TimerClock is disabled.
///</remarks>
///<param name="sender">The timer that raised the tick
event</param>
///<param name="e">Event information</param>
privatevoid TimerClock_Tick(object sender, EventArgs e)
        {
// Only decrement the clock if the activity is still in play
if (!QuizMaster.Instance.ActivityComplete)
            {
this.clk.Decrement();
if (this.showHour)
                {
                    hr.Text = this.clk.Hr.ToString();
                }
else
                {
                    hr.Visibility = Visibility.Collapsed;
                    hrLabel.Visibility = Visibility.Collapsed;
                }
```

```
this.min.Text = this.clk.Min.ToString();
this.sec.Text = this.clk.Sec.ToString();
if (this.clk.Completed)
                {
this.timerClock.IsEnabled = false;
                }
            }
        }
    }
}
```

```
namespace Multipoint.QuizSample
{
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using Microsoft.MultiPoint.CommonTypes;
using Microsoft.MultiPoint.MousePlugIn;
using Microsoft.MultiPoint.SDK;


publicpartialclassWindowMain : Window, IMultiPointMouseEvents
    {
        #region Constructor

///<summary>
/// Initializes a new instance of the WindowMain class
///</summary>
///<remarks>
/// Sets up the Window_Loaded and KeyDown_Event Event handlers,
and calls InitializeComponent()
/// to initialize the WPF controls
///</remarks>
public WindowMain()
        {
            InitializeComponent();

this.Loaded += newRoutedEventHandler(this.Window_Loaded);
this.KeyDown += newKeyEventHandler(this.KeyDown_Event);
        }

        #endregion

        #region IMultiPointMouseEvents Interface Methods

        #pragma warning disable 0067

///<summary>
/// Declare all required Event Handlers to complete the
implementation of the IMultiPointMouseEvents
```

```
/// Interface, which enables MultiPoint Mouse events to be
captured and handled
///</summary>
publiceventRoutedEventHandler MultiPointMouseDownEvent;

publiceventRoutedEventHandler MultiPointMouseEnterEvent;

publiceventRoutedEventHandler MultiPointMouseLeaveEvent;

publiceventRoutedEventHandler MultiPointMouseLeftButtonDownEvent;

publiceventRoutedEventHandler MultiPointMouseLeftButtonUpEvent;

publiceventRoutedEventHandler MultiPointMouseMoveEvent;

publiceventRoutedEventHandler MultiPointMouseRightButtonDownEvent;

publiceventRoutedEventHandler MultiPointMouseRightButtonUpEvent;

publiceventRoutedEventHandler MultiPointMouseUpEvent;

publiceventRoutedEventHandler MultiPointMouseWheelEvent;

publiceventRoutedEventHandler MultiPointPreviewMouseDownEvent;

publiceventRoutedEventHandler
MultiPointPreviewMouseLeftButtonDownEvent;

publiceventRoutedEventHandler
MultiPointPreviewMouseLeftButtonUpEvent;

publiceventRoutedEventHandler MultiPointPreviewMouseMoveEvent;

publiceventRoutedEventHandler
MultiPointPreviewMouseRightButtonDownEvent;

publiceventRoutedEventHandler
MultiPointPreviewMouseRightButtonUpEvent;

publiceventRoutedEventHandler MultiPointPreviewMouseUpEvent;

publiceventRoutedEventHandler MultiPointPreviewMouseWheelEvent;

        #pragma warning restore 0067

        #endregion

            #region Members and Properties

///<summary>
/// Gets Wrap Panel into which Player Regions are added
///</summary>
publicWrapPanel MainWindowWrap
            {
get
                {
```

```
return containerWrap;
                }
            }

            #endregion

        #region Public Functions

///<summary>
/// Adds a message to the status text box, and scrolls the text
box to the end to ensure the
/// new message is visible
///</summary>
///<param name="textToAdd">The text to add to the text box</param>
publicvoid AddToTextBox(string textToAdd)
        {
            statustextbox.Text = string.Format("{0}\n{1}",
statustextbox.Text, textToAdd);
            statustextbox.ScrollToEnd();
        }

///<summary>
/// Adds a new player region to the containerWrap Wrap Panel
///</summary>
///<remarks>
/// Creates a new PlayerRegion object (passing in the Player
parameter as a paremeter),
/// registers the name of the PlayerRegion using the MouseDeviceID
of the mouse assigned
/// to the region, sets a background SolidColorBrush to ensure
mouse events are captured,
/// adds the newly created PlayerRegion to the containerWrap
WrapPanel, and finally adds
/// the new PlayerRegion to the PlayerRegionList in the
PlayerManager class
///</remarks>
///<param name="p">a Player object</param>
publicvoid AddPlayerRegionToCanvas(Player p)
        {
try
            {
// Create new PlayerRegion object, and register a unique name
using the MouseDeviceId to identify it
PlayerRegion r = newPlayerRegion(p);
this.RegisterName(string.Format("region{0}", p.MouseDeviceId), r);

// Need to set a background colour in order for MouseMove events
to fire correctly:
                r.Background =
newSolidColorBrush(Colors.Transparent);

// Add the new player region to the container wrap panel:
                containerWrap.Children.Add(r);

// Add new region to collection by mean of the PlayerManager
Singleton class instance
```

```
PlayerManager.Instance.AddPlayerToRegionList(r);
            }
catch (Exception ex)
            {
this.AddToTextBox("Add Mouse Area to Canvas Exception: " +
ex.Message);
            }
        }

        #endregion

        #region private functions

///<summary>
/// Initialize Microsoft MultiPoint SDK.
///</summary>
///<remarks>
/// Initializes the SDK by calling the InitializeSDK function in
the Common.MouseHelper class,
/// and sets up DeviceArrivalEvent and DeviceRemoveCompleteEvent
handlers to handle Mouse Device
/// connections and disconnections.
///</remarks>
///<returns>true or false</returns>
privatebool InitializeSDK()
        {
try
            {
// After the window is loaded we need to initialize MultiPoint
MultiPointSDK.Instance.Initialize(this);

// Setup event handlers for Plugging and unplugging of devices
MultiPointSDK.Instance.DeviceArrivalEvent +=
newEventHandler<DeviceNotifyEventArgs>(this.MultiPointObject_Devic
eArrivalEvent);
MultiPointSDK.Instance.DeviceRemoveCompleteEvent +=
newEventHandler<DeviceNotifyEventArgs>(this.MultiPointObject_Devic
eRemoveCompleteEvent);
            }
catch (Exception ex)
            {
string exMessage = "Multipoint Intialization Failure: ";
MultiPointException multiPointEx = ex asMultiPointException;

if (multiPointEx != null)
                {
                    exMessage = exMessage +
multiPointEx.ErrorMessage;

if (multiPointEx.InnerException != null)
                    {
                        exMessage = exMessage + ". InnerException:
" + multiPointEx.InnerException.Message;
                    }

if (multiPointEx.HelpLink != null)
```

```
                    {
                        exMessage = exMessage + "\nFor more
information, see Help and Support Center at " +
multiPointEx.HelpLink;
                    }

this.AddToTextBox(exMessage);
                }
else
                {
this.AddToTextBox(exMessage + ex.Message);
                }

returnfalse;
            }

returntrue;
        }

///<summary>
/// Removes the PlayerRegion containing name corresponding to
mouseDeviceId (passed in as a
/// parameter) from the containerWrap WrapPanel and from the
PlayerRegionList in the PlayerManager
/// class
///</summary>
///<param name="mouseDeviceId">the DeviceId of the mouse
associated with the region to be removed</param>
privatevoid RemovePlayerRegion(string mouseDeviceId)
        {
try
            {
// Remove the PlayerRegion User Control from the containerWrap
WrapPanel
this.containerWrap.Children.Remove(PlayerManager.Instance.GetPlaye
rRegionByDeviceID(mouseDeviceId));

// Remove the PlayerRegion from the PlayerRegionList collection
PlayerManager.Instance.RemovePlayerFromRegionList(mouseDeviceId);
            }
catch (Exception ex)
            {
this.AddToTextBox("Remove Mouse Exception: " + ex.Message);
            }
        }

///<summary>
/// Displays the initial questions for regions, by calling
DisplayInitialQuestions() on QuizMaster
/// singleton class
///</summary>
privatevoid DisplayInitialQuestionsOnPlayerRegions()
        {
try
            {
QuizMaster.Instance.DisplayInitialQuestions();
```

```
            }
catch (Exception ex)
            {
this.AddToTextBox("Display Questions Exception: " + ex.Message);
            }
        }

        #endregion

        #region event handlers

///<summary>
/// Called when the window is initially loaded. It will load all
the controls and
/// initialize the SDK, check that there are not too few or too
many mice, and if
/// both conditions are met it will Intialize the connected mice,
add and event handler
/// to listen for Mouse Move events (required to monitor the
location of the mouse to
/// bound it to its corresponding Player Region), and finally
calls
/// DisplayInitialQuestionsOnPlayerRegions function.
///</summary>
///<param name="sender">The window object</param>
///<param name="e">The routed event arguments</param>
privatevoid Window_Loaded(object sender, RoutedEventArgs e)
        {
try
            {
if (this.InitializeSDK())
                {
if (!PlayerManager.Instance.CheckIfTooManyMice())
                    {
// Check if too few mice
if (!PlayerManager.Instance.CheckIfTooFewMice())
                        {
// Initialize and connect mice, and add them to the canvas
// Need to pass in 'this' so that the PlayerManager can add new
mouse areas to the canvas
PlayerManager.Instance.InitializeConnectedMice(this);

// Add an event handler to listen for movement of the multipoint
mice
// This is required to monitor the location of the mouse, and keep
it within its allowed boundary
MultiPointMouseEvents.AddMultiPointMouseMoveHandler(this,
newRoutedEventHandler(this.MultipointObject_MouseMove));

// Show a question on each player region
this.DisplayInitialQuestionsOnPlayerRegions();
                        }
else
                        {
// too few mice
```

```
// TODO: Pause the Activity, prevent regions from painting until
enough mice connected
this.AddToTextBox(string.Format(
"Too few mice are connected. This activity requires a minimum of
{0} players",
PlayerManager.MinimumPlayers.ToString()));
                        }
                    }
else
                    {
// too many mice
this.AddToTextBox(String.Format(
"Too many mice are connected. This activity allows a maximum of
{0} players",
PlayerManager.MaximumPlayers.ToString()));
                    }
                }
            }
catch (MultiPointException ex)
            {
this.AddToTextBox("Window Load Failure, Multipoint Exception: " +
ex.ErrorMessage);
            }
catch (Exception ex)
            {
this.AddToTextBox("Window Load Failure, Exception: " +
ex.Message);
            }
        }

///<summary>
/// Handles keyboard events
///</summary>
///<remarks>
/// Adds event handlers for the Escape key (closes the Activity)
and Space key (freezes all the mice)
///</remarks>
///<param name="sender">The sender of the keydown</param>
///<param name="e">The event information</param>
privatevoid KeyDown_Event(object sender, KeyEventArgs e)
        {
switch (e.Key)
            {
caseKey.Escape: // Close the Activity
Application.Current.Shutdown();
break;
caseKey.Space: // Freeze all mice
PlayerManager.Instance.ToggleMiceFrozen();
break;
            }
        }

///<summary>
/// This handler is called when a new device is added to the
machine
```

```
/// A new Player object is created by PlayerManager and passed
back, and then it is added to the Canvas
/// If there are already too many mice already attached to the
machine, the cursor of the mouse that is
/// plugged in will be hidden by the called
AddPlayerRegionToCanvas function.
///</summary>
///<param name="sender">The sender of the arrival event</param>
///<param name="e">Event information</param>
privatevoid MultiPointObject_DeviceArrivalEvent(object sender,
DeviceNotifyEventArgs e)
        {
// Continue working only if the device that is plugged in is a
mouse
if (e.DeviceInfo.DeviceType == DeviceType.Mouse)
            {
try
                {
Player p = PlayerManager.Instance.HandleMouseArrival(e);
if (p != null)
                    {
this.AddPlayerRegionToCanvas(p);
                    }

this.AddToTextBox(string.Format("\nMouse # {0} was added",
e.DeviceInfo.ID.ToString()));
                }
catch (Exception ex)
                {
this.AddToTextBox("\nProblem with device arrival: " + ex.Message);
                }
            }
        }

///<summary>
/// This handler is called when a  device is removed from the
machine.
/// This function removes the device visual for the player whose
mouse is unplugged.
/// It also removes the player's region from the canvas.
///</summary>
///<param name="sender">The sender of the event</param>
///<param name="e">Event information</param>
privatevoid MultiPointObject_DeviceRemoveCompleteEvent(object
sender, DeviceNotifyEventArgs e)
        {
// Ensure that the device that was unplugged was a mouse device
if (e.DeviceInfo.DeviceType == DeviceType.Mouse)
            {
try
                {
PlayerManager.Instance.HandleMouseRemoval(e);
this.RemovePlayerRegion(e.DeviceInfo.DeviceID);
this.AddToTextBox(string.Format("\nMouse # {0} was removed",
e.DeviceInfo.ID.ToString()));
                }
```

```
catch (Exception ex)
                {
this.AddToTextBox("\nProblem with Device Removal: " + ex.Message);
                }
            }
        }

///<summary>
/// Event handler which handles mouse movements.
/// It calls PlayerManager.HandleMouseMovement function, which is
///  responsible for keeping mouse cursors within their allowed
boundaries
///</summary>
///<param name="sender">The sender of the event</param>
///<param name="e">Event information</param>
privatevoid MultipointObject_MouseMove(object sender,
RoutedEventArgs e)
        {
PlayerManager.Instance.HandleMouseMovement(this, e);
        }

        #endregion
    }
}
```

```
namespace EduColGame
{
///<summary>
/// Interaction logic for Petunjuk.xaml
///</summary>
publicpartialclassPetunjuk : Window
    {
public Petunjuk()
        {
            InitializeComponent();
        }
    }
}
```

```
namespace EduColGame
{
///<summary>
/// Interaction logic for tentangAplikasi.xaml
///</summary>
publicpartialclasstentangAplikasi : Window
    {
public tentangAplikasi()
        {
            InitializeComponent();
        }
```

```
privatevoid image1_ImageFailed(object sender,
ExceptionRoutedEventArgs e)
        {

        }

privatevoid button1_Click(object sender, RoutedEventArgs e)
        {
this.Close();
        }
    }
}
```

```
namespace EduColGame
{
///<summary>
/// Interaction logic for testMouse.xaml
///</summary>
publicpartialclasstestMouse : Window
    {
///<summary>
/// Logic
///</summary>
publicMultiPointSDK MultiPointManager = MultiPointSDK.Instance;
publicList<Bitmap> cursorList = newList<Bitmap>();

public testMouse()
        {
            InitializeComponent();
        }

privatevoid Window_Loaded(object sender, RoutedEventArgs e)
        {
            InitCursorList();
        }

privatevoid Window_KeyDown(object sender, KeyEventArgs e)
        {
if (e.Key == Key.M)
            {
                ActivateMouse(this);
            }
elseif (e.Key == Key.D)
            {
                DeactivateMouse(this);
            }
        }

privatevoid button1_Click(object sender, RoutedEventArgs e)
        {
this.Close();
        }

        #region Multipoint Activation
```

```
///<summary>
/// Activate Multipoint to this window
///</summary>
///<param name="Window"></param>
privatevoid ActivateMouse(Window Window)
        {
            MultiPointManager.RegisterMouseDevice();
            MultiPointManager.CurrentWindow = this;
            MultiPointManager.DrawMouseDevices();
MultiPointSDK.HideSystemCursor();
MultiPointSDK.SystemCursorPosition = new System.Windows.Point(0,
0);

int i = 0;
foreach (var mouse in MultiPointManager.MouseDeviceList)
            {

var mouseObject =
((MultiPointMouseDevice)((DeviceInfo)mouse).DeviceVisual);
                mouseObject.MouseSpeed = 2;
                mouseObject.CursorImage =
ConvertBitmapToBitmapImage(selectCursor(i));
                i++;
            }
        }

///<summary>
/// Deactivate Multipoint to this window
///</summary>
///<param name="wm"></param>
privatevoid DeactivateMouse(Window wm)
        {
            MultiPointManager.UnRegisterMouseDevice();
            MultiPointManager.CurrentWindow = wm;
MultiPointSDK.ShowSystemCursor();
//MultiPointSDK.SystemCursorPosition = new
System.Windows.Point(80, 80);

        }
        #endregion

        #region CursorImage Manipulation
///<summary>
/// mengembalikan bitmap image
///</summary>
///<param name="b"></param>
///<returns></returns>
publicstaticBitmapImage ConvertBitmapToBitmapImage(Bitmap b)
        {
BitmapImage bmpimg = newBitmapImage();

            System.IO.MemoryStream memStream = new
System.IO.MemoryStream();
            bmpimg.BeginInit();
            b.MakeTransparent(System.Drawing.Color.White);
```

```
            b.Save(memStream,
System.Drawing.Imaging.ImageFormat.Png);
            bmpimg.StreamSource = memStream;
            bmpimg.EndInit();
return bmpimg;
        }

///<summary>
/// Add resource to List
///</summary>
publicvoid InitCursorList()
        {
            cursorList.Add(Properties.Resources.lion);
            cursorList.Add(Properties.Resources.monkey);
            cursorList.Add(Properties.Resources.ox);
            cursorList.Add(Properties.Resources.pig);
            cursorList.Add(Properties.Resources.rabbit);

        }

///<summary>
/// Select Cursor from Cursor List
///</summary>
///<param name="i"></param>
///<returns></returns>
publicBitmap selectCursor(int i)
        {
if (i < cursorList.Count)
            {
return cursorList[i];
            }
else
            {
return cursorList[0];
            }
        }

privatevoid about_Copy1_Click(object sender,
System.Windows.RoutedEventArgs e)
        {
      this.Close(); // TODO: Add event handler implementation
here.
        }
        #endregion

privatevoid button1_Click_1(object sender, RoutedEventArgs e)
        {
/*
            string str = @"C:\Documents and
Settings\n\Desktop\mpk\mm\EduColGame\EduColGame\bin\Release\Multip
oint.QuizSample.exe";
            Process process = new Process();
            process.StartInfo.FileName = str;
            process.Start();
            */
```

```
            System.Diagnostics.Process.Start(@"C:\Documents and
Settings\n\Desktop\mpk\mm\EduColGame\EduColGame\bin\Release\Multip
oint.QuizSample.exe");

        }
    }
}
```

```
namespace EduColGame
{
///<summary>
/// Interaction logic for Window1.xaml
///</summary>
publicpartialclassMainMenu : Window
    {
privatevoid Keluar_Click(object sender,
System.Windows.RoutedEventArgs e)
        {
      // TODO: Add event handler implementation here.
                this.Close();
        }

privatevoid setMouse_Click(object sender,
System.Windows.RoutedEventArgs e)
        {
      // TODO: Add event handler implementation here.

        }

privatevoid image1_MouseDown(object sender,
System.Windows.Input.MouseButtonEventArgs e)
        {
      // TODO: Add event handler implementation here.

        }

privatevoid image2_MouseDown(object sender,
System.Windows.Input.MouseButtonEventArgs e)
        {
      // TODO: Add event handler implementation here.
        }

privatevoid image4_MouseDown(object sender,
System.Windows.Input.MouseButtonEventArgs e)
        {

        }

privatevoid image4_MouseEnter(object sender, MouseEventArgs e)
        {

        }

privatevoid image2_MouseEnter(object sender, MouseEventArgs e)
        {
```

```
        }

privatevoid image2_MouseDown_1(object sender, MouseButtonEventArgs
e)
        {
var sM = newtestMouse();
            sM.Show();
        }


privatevoid tebakKartu_ImageFailed(object sender,
ExceptionRoutedEventArgs e)
        {

        }

privatevoid image4_ImageFailed(object sender,
ExceptionRoutedEventArgs e)
        {

        }

privatevoid image2_ImageFailed(object sender,
ExceptionRoutedEventArgs e)
        {

        }

privatevoid Petunjuk_MouseDown(object sender,
System.Windows.Input.MouseButtonEventArgs e)
        {
      // TODO: Add event handler implementation here.
                var sM = newPetunjuk();
            sM.Show();
        }

privatevoid about_Click(object sender,
System.Windows.RoutedEventArgs e)
        {
      // TODO: Add event handler implementation here.
                var sM = newtentangAplikasi();
            sM.Show();
        }
    }
}
```

XML

```
?xmlversion="1.0"?>
<Quizxmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<Questions>
<Question>
<Content> 2 + 6 = ? </Content>
```

```
<Options>
<optioncorrect="true">8</option>
<optioncorrect="false">4</option>
<optioncorrect="false">7</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Dimanakah bintang berada ? </Content>
<Options>
<optioncorrect="true">Langit</option>
<option>Tanah</option>
<option>Sumur</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Daun banyak ditemukan dalam bentuk ? </Content>
<Options>
<option>Bulat</option>
<optioncorrect="true">Lonjong</option>
<option>Persegi</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Kelelawar mencari makan pada saat ? </Content>
<Options>
<optioncorrect="true">Malam</option>
<option>Siang</option>
<option>Petang</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Bumi dalam urutan planet ke berapa dalam tata surya ?
</Content>
<Options>
<option>10</option>
<option>2</option>
<optioncorrect="true">3</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 17 - 5 = ? </Content>
<Options>
<option>11</option>
<option>18</option>
<optioncorrect="true">12</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 4 x 5 = ? </Content>
<Options>
```

```
<option>24</option>
<optioncorrect="true">20</option>
<option>31</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 16 - 8 = ? </Content>
<Options>
<option>19</option>
<optioncorrect="true">8</option>
<option>70</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Katak adalah hewan yang hidup di dua alam. Istilah ini
sering disebut ?  </Content>
<Options>
<option>Karnivora</option>
<option>Berdarah panas</option>
<optioncorrect="true">Amfibi</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Istilah herbivora adalah sebutan untuk hewan yang
memakan ? </Content>
<Options>
<option>Bangkai</option>
<optioncorrect="true">Tumbuhan</option>
<option>Mie Rebus</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Garpu tala menyerupai huruf apa ?  </Content>
<Options>
<option>U</option>
<optioncorrect="true">Y</option>
<option>X</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 7 x 10 = ? </Content>
<Options>
<option>43</option>
<option>78</option>
<optioncorrect="true">70</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 5 + 7 = ?  </Content>
<Options>
```

```
<optioncorrect="true">13</option>
<option>3</option>
<option>24</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 49 / 7 = ? </Content>
<Options>
<option>44</option>
<option>2</option>
<optioncorrect="true">7</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 44 / 4 = ? </Content>
<Options>
<option>12</option>
<optioncorrect="true">11</option>
<option>9</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 50 + 99 = ? </Content>
<Options>
<option>178</option>
<option>56</option>
<optioncorrect="true">149</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 77 * 10 = ? </Content>
<Options>
<option>679</option>
<option>908</option>
<optioncorrect="true">770</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Kucing adalah hewan yang ? </Content>
<Options>
<option>Memamah biak</option>
<option>Bertelur</option>
<optioncorrect="true">Beranak</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Kuda mempunyai berapa kaki ? </Content>
<Options>
<optioncorrect="true">4</option>
<option>3</option>
```

```
<option>2</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Untuk apakah anjing menjulurkan lidah ? </Content>
<Options>
<option>Bergaya aja</option>
<optioncorrect="true">Menurunkan suhu tubuh</option>
<option>Mengeluarkan bebauan</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Tepung terbuat dari ? </Content>
<Options>
<option>Pasir</option>
<option>Air</option>
<optioncorrect="true">Beras</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 654 - 55 = ? </Content>
<Options>
<option>674</option>
<option>345</option>
<optioncorrect="true">599</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Ikan bernafas dengan ? </Content>
<Options>
<optioncorrect="true">Insang</option>
<option>Paru-paru</option>
<option>Sayap</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Air mengalir dari hulu ke ? </Content>
<Options>
<option>Laut</option>
<optioncorrect="true">Hilir</option>
<option>Langit</option>
<option>Awan</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Apa sebutan untuk binatang yang memakan daging ?
</Content>
<Options>
<optioncorrect="true">Karnivora</option>
<option>Omnivora</option>
```

```
<option>Herbivora</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Lebah mencari nektar pada ? </Content>
<Options>
<optioncorrect="true">Bunga</option>
<option>Sarangnya</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Kupu-kupu berasal dari ?  </Content>
<Options>
<option>Batu</option>
<option>Burung</option>
<optioncorrect="true">Ulat</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Uap adalah jenis lain dari ? </Content>
<Options>
<optioncorrect="true">Air</option>
<option>Batu</option>
<option>Es</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Minyak dan air adalah jenis dari ? </Content>
<Options>
<option>Benda Padat</option>
<option>Udara</option>
<optioncorrect="true">Cairan</option>
</Options>
<Tip />
</Question>
<Question>
<Content> E=MC^2 adalah rumus terkenal dari ? </Content>
<Options>
<option>Newton</option>
<optioncorrect="true">Einstein</option>
<option>Galileo</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Benda untuk menurunkan tegangan listrik ? </Content>
<Options>
<option>Palu</option>
<optioncorrect="true">Trafo</option>
<option>Obeng</option>
</Options>
<Tip />
```

```
</Question>
<Question>
<Content> Kendaraan di udara  disebut ? </Content>
<Options>
<option>Mobil</option>
<option>Kapal</option>
<optioncorrect="true">Pesawat</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Pink adalah bagian dari warna ? </Content>
<Options>
<optioncorrect="true">Merah</option>
<option>Putih</option>
<option>Orange</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 15 x 6 = ? </Content>
<Options>
<optioncorrect="true">90</option>
<option>88</option>
<option>98</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Ular termasuk hewan ? </Content>
<Options>
<optioncorrect="true">Melata</option>
<option>Berkaki</option>
<option>Bersayap</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Anggur sering ditemui di daerah ? </Content>
<Options>
<optioncorrect="true">Pegunungan</option>
<option>Dataran Rendah</option>
<option>Laut</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 56 / 8 = ? </Content>
<Options>
<optioncorrect="true">7</option>
<option>9</option>
<option>8</option>
</Options>
<Tip />
</Question>
<Question>
```

```
<Content> 66 x 5 = ? </Content>
<Options>
<option>342</option>
<option>678</option>
<optioncorrect="true">330</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 32 + 98 = ? </Content>
<Options>
<option>111</option>
<optioncorrect="true">130</option>
<option>879</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Siapakah pahlawan dalam uang lembaran Rp 5000,- ?
</Content>
<Options>
<option>Cut Tari</option>
<optioncorrect="true">Tuanku Imam Bonjol</option>
<option>Pangeran Antasari</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Premium dan Pertamax adalah jenis dari ? </Content>
<Options>
<option>Cairan Pupuk</option>
<optioncorrect="true">Bahan bakar</option>
<option>Minuman</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Lampu pijar adalah temuan ? </Content>
<Options>
<option>James Watt</option>
<optioncorrect="true">Thomas Alfa Edison</option>
<option>Kabayan</option>
</Options>
<Tip />
</Question>
<Question>
<Content> Burung yang tidak bisa terbang adalah ? </Content>
<Options>
<optioncorrect="true">Penguin</option>
<option>Angsa</option>
<option>Ular</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 45 + 77 = ?  </Content>
```

```
<Options>
<option>145</option>
<optioncorrect="true">122</option>
<option>179</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 78 x 5 = ? </Content>
<Options>
<optioncorrect="true">390</option>
<option>567</option>
<option>234</option>
</Options>
<Tip />
</Question>
<Question>
<Content> 0 x 12 = ?</Content>
<Options>
<optioncorrect="true">0</option>
<option>12</option>
<option>120</option>
</Options>
<Tip />
</Question>
</Questions>
</Quiz>
```