Report for depth-based interactive Kinect application

SM3603 Assignment 1

Tam Chin Pang
ID: 56226481

24 February 2022

Introduction

Motivation

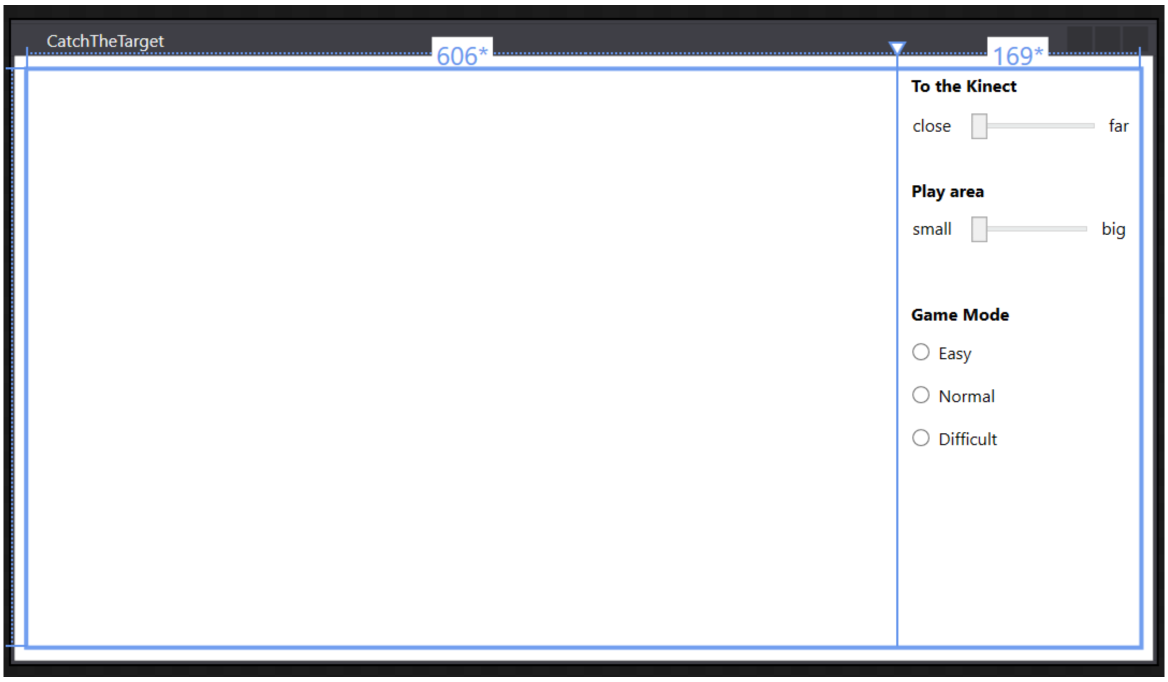
NUI (Nature User Interface) has gained increasing popularity worldwide. In this course assignment, I am required to develop a WPF application supporting depth-enabled interactivity to explore the possibility of NUI. Encouraged by the shooting games, the game with a similar mechanic but using the body's motion tracking as the input is eventually set as the orientation of the project.

What-is-used

Kinect and visual studio are used as the hardware and software in this project respectively.

Solution

Interface



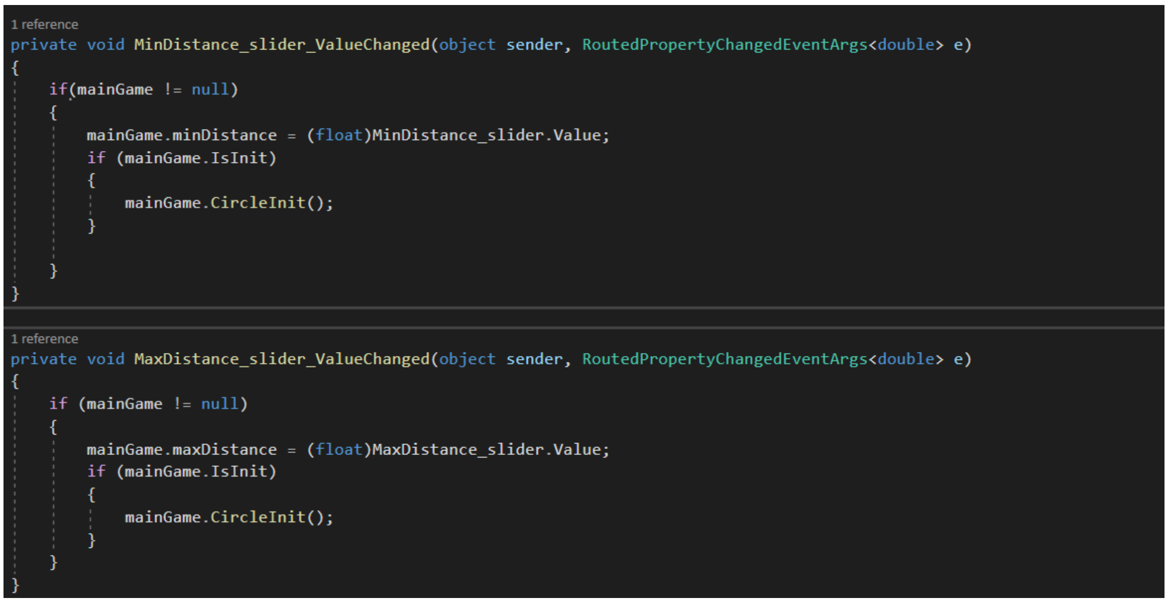
In the left cell, the game screen is placed. In the right cell, the setting of the game is shown, which allows players to do modifications at any time.

About the setting:

1 “To the Kinect” setting provides players a slider to do the modification according to their minimum playing distance to the Kinect.

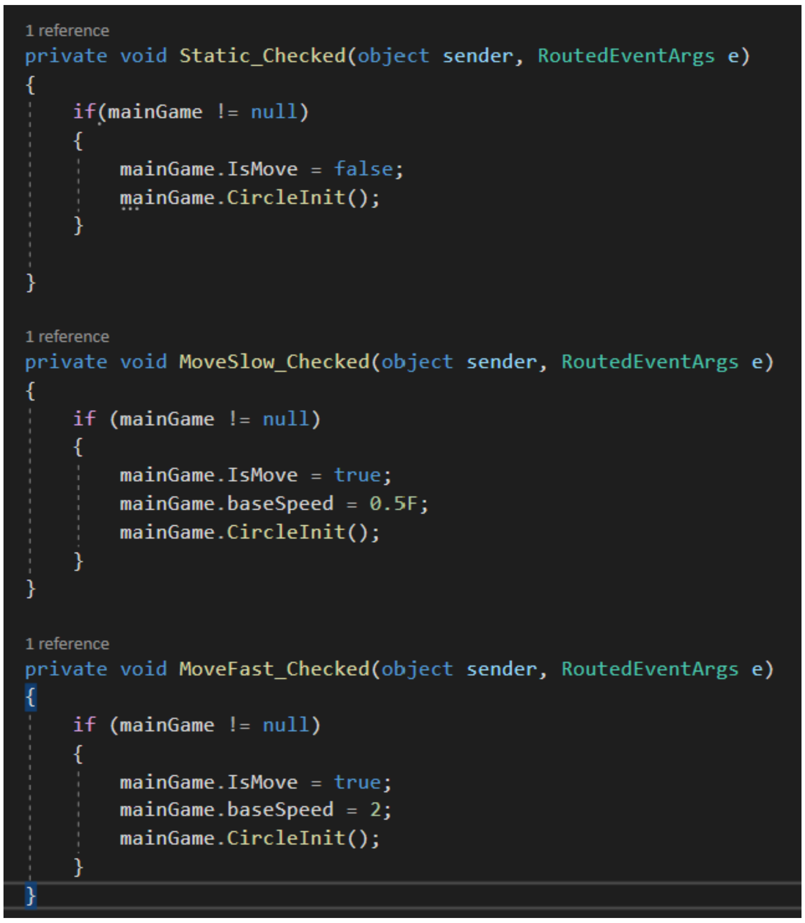
2 “Play area” setting also provides the input function for the players to do the changes according to the size of their play area.

3 “Game Mode” settings provide radio buttons for the players to select the difficulty of the game.



```
1 reference
private void MinDistance_slider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if(mainGame != null)
    {
        mainGame.minDistance = (float)MinDistance_slider.Value;
        if (mainGame.IsInit)
        {
            mainGame.CircleInit();
        }

    }
}

1 reference
private void MaxDistance_slider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    if (mainGame != null)
    {
        mainGame.maxDistance = (float)MaxDistance_slider.Value;
        if (mainGame.IsInit)
        {
            mainGame.CircleInit();
        }
    }
}
```

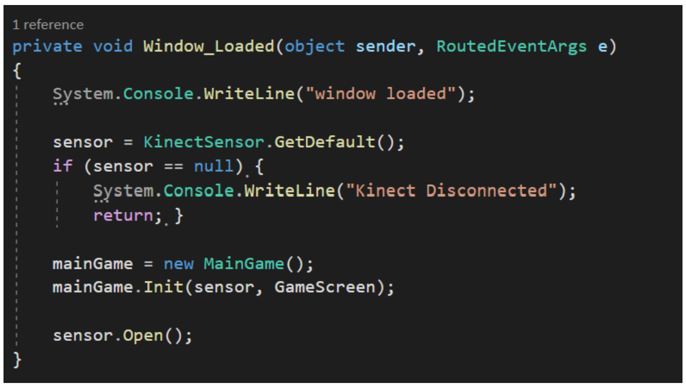
This is the coding part of sliders.



```
1 reference
private void Static_Checked(object sender, RoutedEventArgs e)
{
    if(mainGame != null)
    {
        mainGame.IsMove = false;
        mainGame.CircleInit();
    }

}

1 reference
private void MoveSlow_Checked(object sender, RoutedEventArgs e)
{
    if (mainGame != null)
    {
        mainGame.IsMove = true;
        mainGame.baseSpeed = 0.5F;
        mainGame.CircleInit();
    }
}

1 reference
private void MoveFast_Checked(object sender, RoutedEventArgs e)
{
    if (mainGame != null)
    {
        mainGame.IsMove = true;
        mainGame.baseSpeed = 2;
        mainGame.CircleInit();
    }
}
```

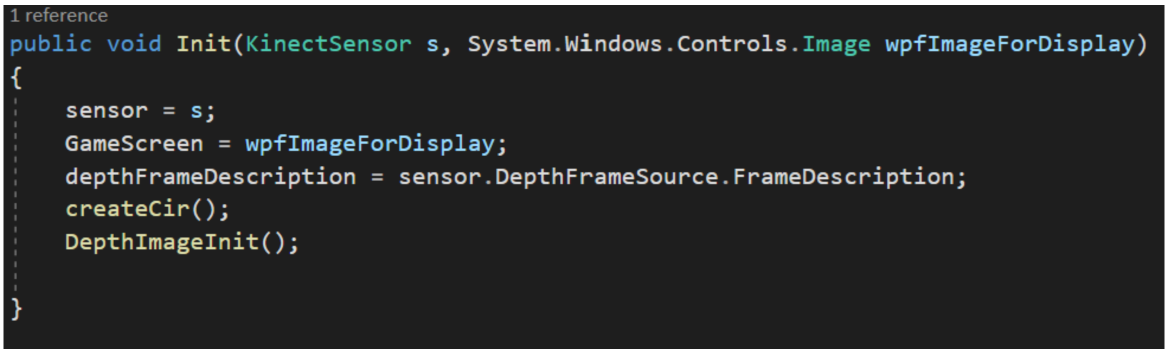
This is the coding part of radio buttons for the change of game mode

Initialisation of game screen



```
1 reference
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    System.Console.WriteLine("window loaded");

    sensor = KinectSensor.GetDefault();
    if (sensor == null) {
        System.Console.WriteLine("Kinect Disconnected");
        return; }

    mainGame = new MainGame();
    mainGame.Init(sensor, GameScreen);

    sensor.Open();
}
```

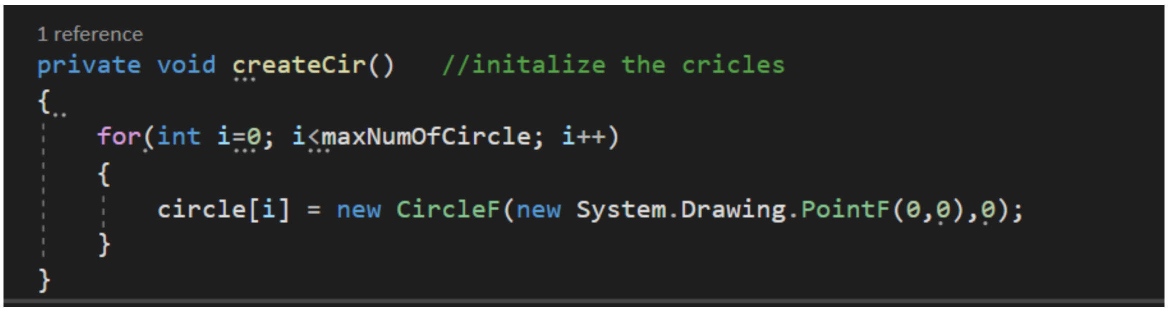
When game screen is initially loaded, the Kinect sensor is opened and initialises the mainGame program



```
1 reference
public void Init(KinectSensor s, System.Windows.Controls.Image wpfImageForDisplay)
{
    sensor = s;
    GameScreen = wpfImageForDisplay;
    depthFrameDescription = sensor.DepthFrameSource.FrameDescription;
    createCir();
    DepthImageInit();

}
```

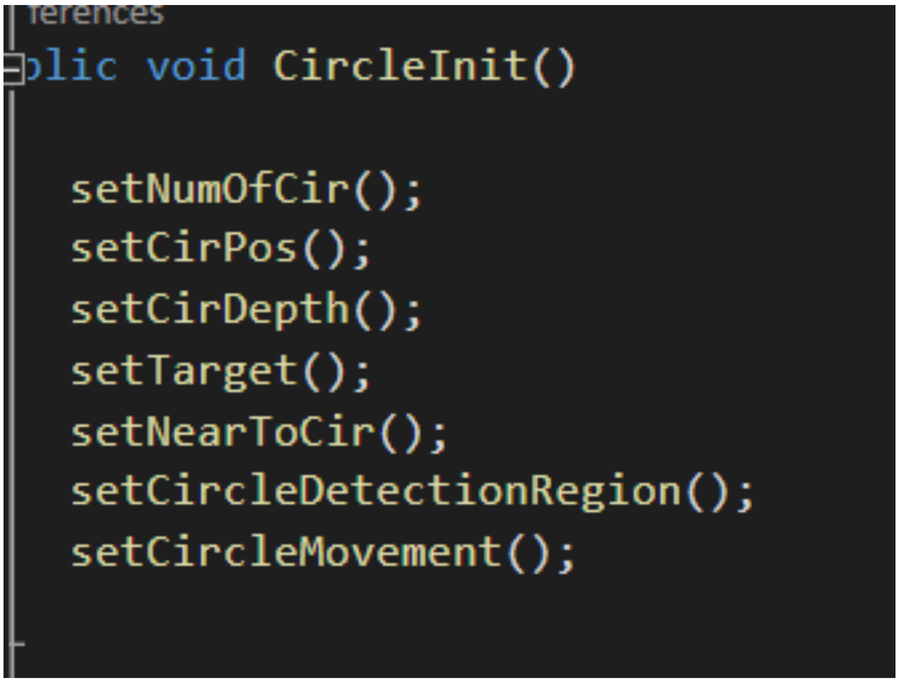
Circles are created as the main game items and the depth image is initialised as the main input of the game.

Initialisation of circles



```
1 reference
private void createCir()   //initalize the cricles
{
    for(int i=0; i<maxNumOfCircle; i++)
    {
        circle[i] = new CircleF(new System.Drawing.PointF(0,0),0);
    }
}
```

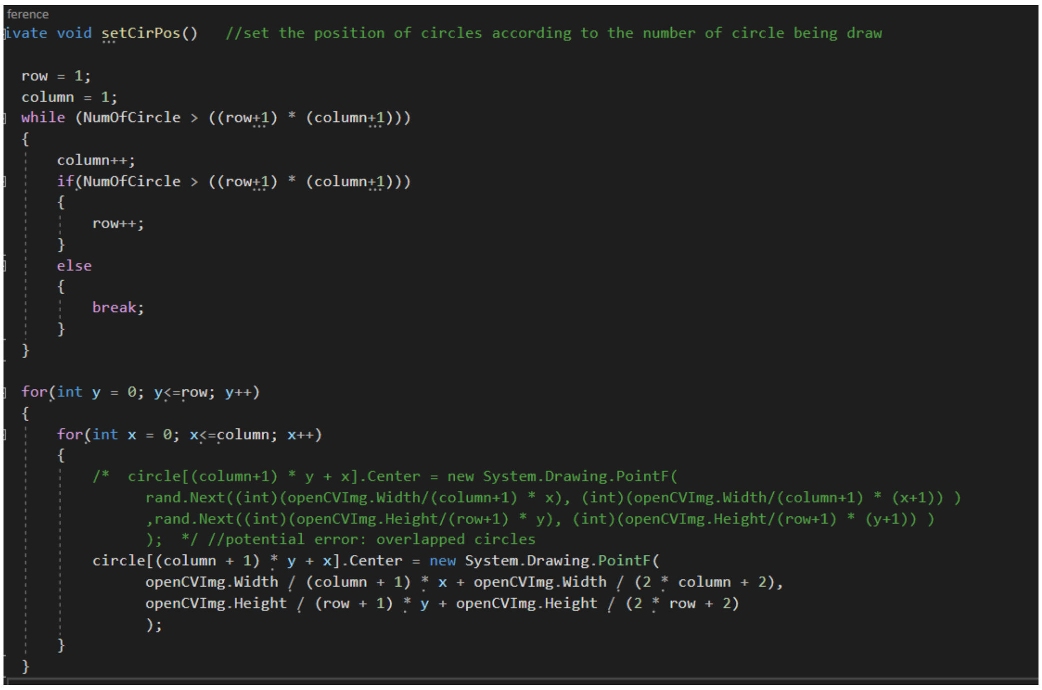
An array of circle objects is prepared



```
ferences
plic void CircleInit()

    setNumOfCir();
    setCirPos();
    setCirDepth();
    setTarget();
    setNearToCir();
    setCircleDetectionRegion();
    setCircleMovement();
```

Initialisation of all attributes of them will be done in the first frame: Assigning a random number of circle generation —> Setting positions for them —> setting depths for them —> set one of the circles as the target —> reset the nearToCir to the default value, meaning that players' detected hand is far away from circles —> draw detection regions[rectangle] for all circles —> set the initial speeds and moving angles for them.

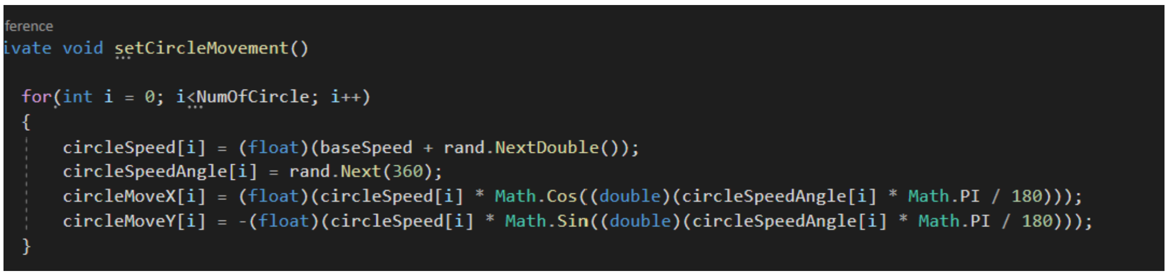
Grid System for Circles



```
ference
ivate void setCirPos()   //set the position of circles according to the number of circle being draw

  row = 1;
  column = 1;
  while (NumOfCircle > ((row+1) * (column+1)))
  {
      column++;
      if(NumOfCircle > ((row+1) * (column+1)))
      {
          row++;
      }
      else
      {
          break;
      }
  }

  for(int y = 0; y<=row; y++)
  {
      for(int x = 0; x<=column; x++)
      {
          /*  circle[(column+1) * y + x].Center = new System.Drawing.PointF(
                rand.Next((int)(openCVImg.Width/(column+1) * x), (int)(openCVImg.Width/(column+1) * (x+1)) )
                ,rand.Next((int)(openCVImg.Height/(row+1) * y), (int)(openCVImg.Height/(row+1) * (y+1)) )
                );  */ //potential error: overlapped circles
          circle[(column + 1) * y + x].Center = new System.Drawing.PointF(
                openCVImg.Width / (column + 1) * x + openCVImg.Width / (2 * column + 2),
                openCVImg.Height / (row + 1) * y + openCVImg.Height / (2 * row + 2)
                );
      }
  }
```

One of the interesting programming problems in the project to be set the optimised grid structure uniformly positioning all the circles for every random generation of numbers of them. To solve the problem, the number of rows' and columns' lines is firstly found using the relationship of "Number of circles = row*column". A nested for-loop is then used to assign every position of circles using the grid system initialised with the number of rows' and columns' lines.

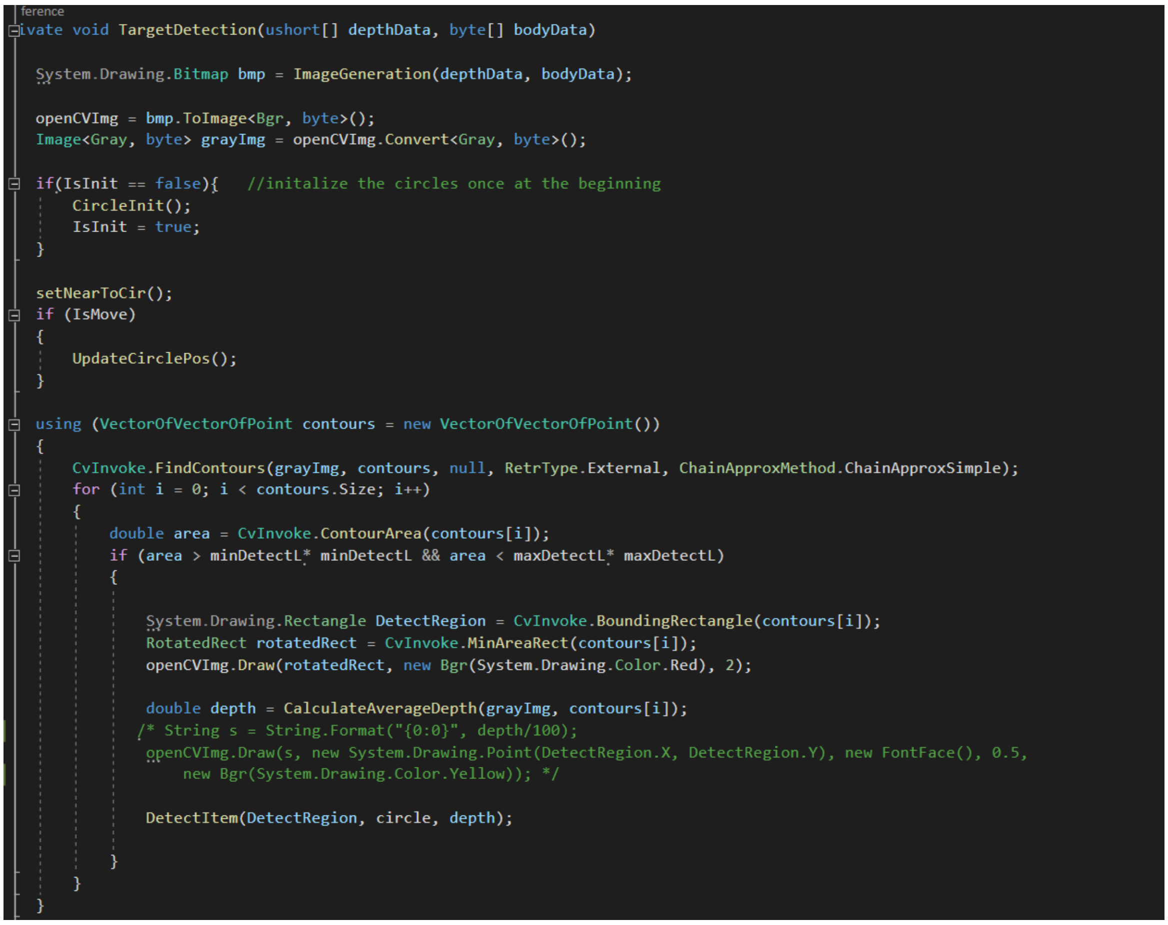
Random movement for Circles



```
ference
ivate void setCircleMovement()

  for(int i = 0; i<NumOfCircle; i++)
  {
      circleSpeed[i] = (float)(baseSpeed + rand.NextDouble());
      circleSpeedAngle[i] = rand.Next(360);
      circleMoveX[i] = (float)(circleSpeed[i] * Math.Cos((double)(circleSpeedAngle[i] * Math.PI / 180)));
      circleMoveY[i] = -(float)(circleSpeed[i] * Math.Sin((double)(circleSpeedAngle[i] * Math.PI / 180)));
  }
```

Another problem related to the setup of circles is their movement initialisation. Sin&Cos relationship is applied to obtain the x,y changes in every frame for each circle using its random assigned speed and angle.

Frame Updates for game screen

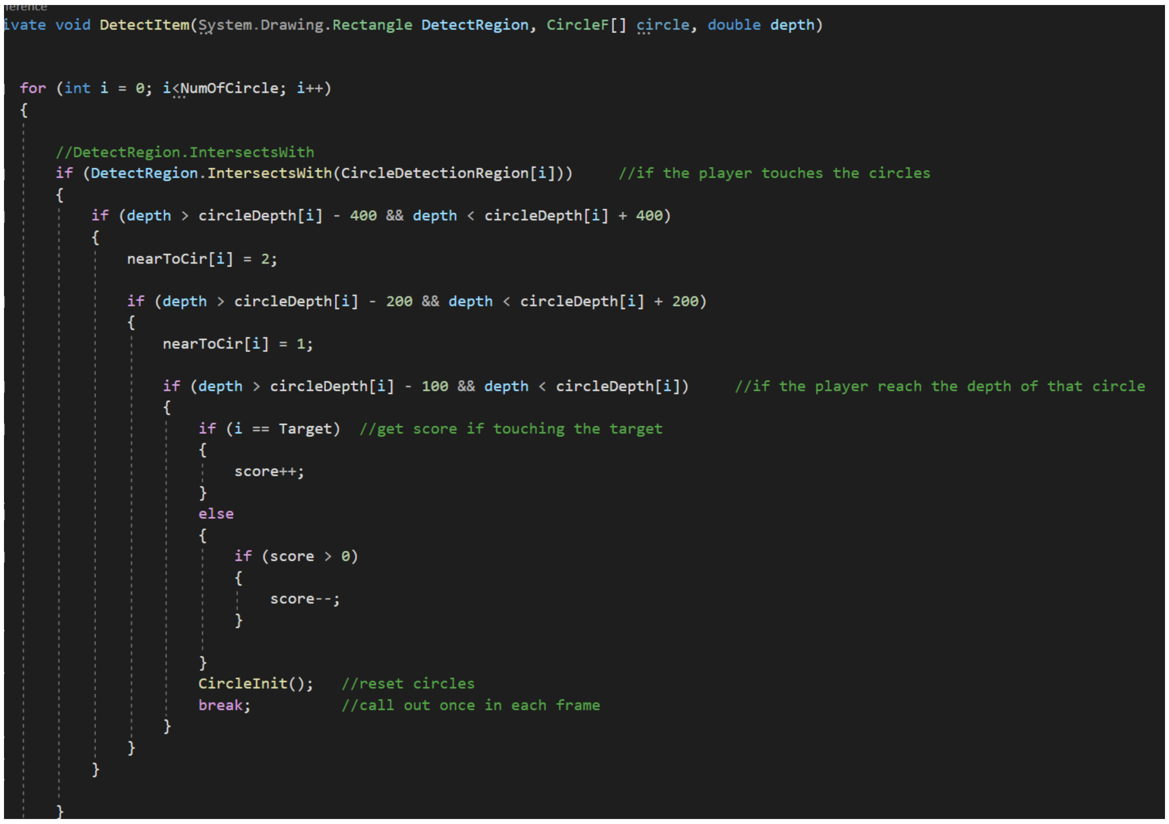


```
ference
ivate void TargetDetection(ushort[] depthData, byte[] bodyData)

    System.Drawing.Bitmap bmp = ImageGeneration(depthData, bodyData);

    openCVImg = bmp.ToImage<Bgr, byte>();
    Image<Gray, byte> grayImg = openCVImg.Convert<Gray, byte>();

    if(IsInit == false){   //initalize the circles once at the beginning
        CircleInit();
        IsInit = true;
    }

    setNearToCir();
    if (IsMove)
    {
        UpdateCirclePos();
    }

    using (VectorOfVectorOfPoint contours = new VectorOfVectorOfPoint())
    {
        CvInvoke.FindContours(grayImg, contours, null, RetrType.External, ChainApproxMethod.ChainApproxSimple);
        for (int i = 0; i < contours.Size; i++)
        {
            double area = CvInvoke.ContourArea(contours[i]);
            if (area > minDetectL* minDetectL && area < maxDetectL* maxDetectL)
            {

                System.Drawing.Rectangle DetectRegion = CvInvoke.BoundingRectangle(contours[i]);
                RotatedRect rotatedRect = CvInvoke.MinAreaRect(contours[i]);
                openCVImg.Draw(rotatedRect, new Bgr(System.Drawing.Color.Red), 2);

                double depth = CalculateAverageDepth(grayImg, contours[i]);
               /* String s = String.Format("{0:0}", depth/100);
                openCVImg.Draw(s, new System.Drawing.Point(DetectRegion.X, DetectRegion.Y), new FontFace(), 0.5,
                    new Bgr(System.Drawing.Color.Yellow)); */

                DetectItem(DetectRegion, circle, depth);

            }
        }
    }
```

This is the image processing for every frame:

Convert the depth bitmap into a processable image —> do the circle initialisation mentioned above once when first frame arrives —> reset nearToCir —> update the positions of circles if normal or difficult game mode is selected —> draw the detection region for hand(s) —> test if the region intersects with any of the circles —> convert the image back to bitmap —> update the frame

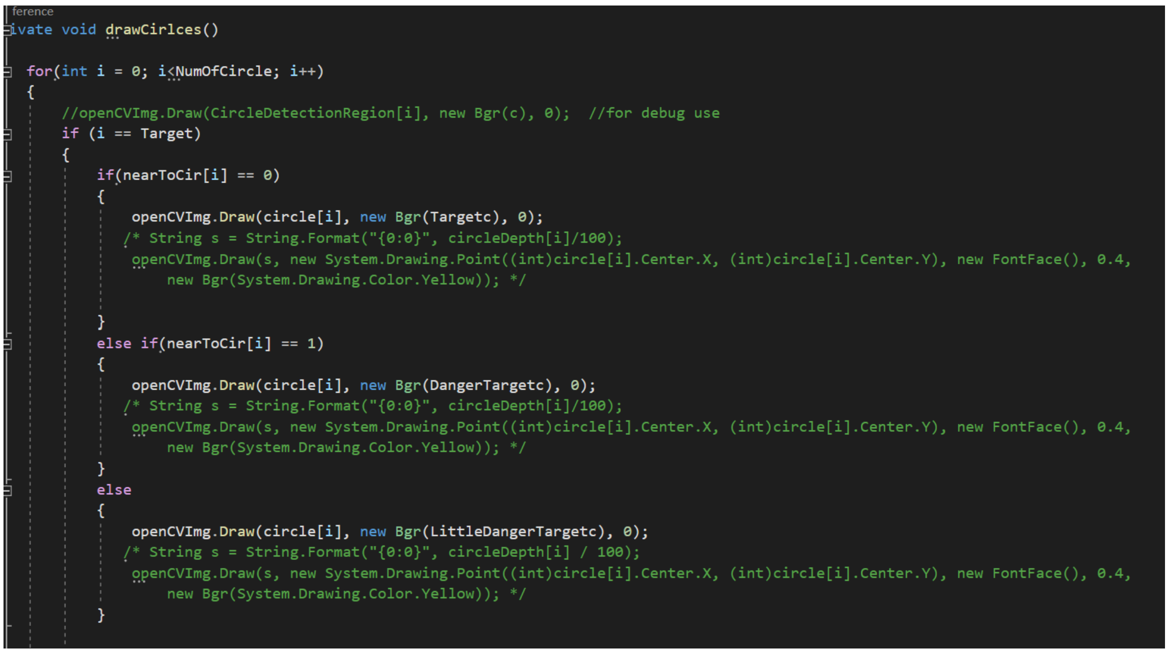
Circles’ Detection



```
erence
ivate void DetectItem(System.Drawing.Rectangle DetectRegion, CircleF[] circle, double depth)

 for (int i = 0; i<NumOfCircle; i++)
 {
     //DetectRegion.IntersectsWith
     if (DetectRegion.IntersectsWith(CircleDetectionRegion[i]))     //if the player touches the circles
     {
         if (depth > circleDepth[i] - 400 && depth < circleDepth[i] + 400)
         {
             nearToCir[i] = 2;

             if (depth > circleDepth[i] - 200 && depth < circleDepth[i] + 200)
             {
                 nearToCir[i] = 1;

                 if (depth > circleDepth[i] - 100 && depth < circleDepth[i])     //if the player reach the depth of that circle
                 {
                     if (i == Target)  //get score if touching the target
                     {
                         score++;
                     }
                     else
                     {
                         if (score > 0)
                         {
                             score--;
                         }

                     }
                     CircleInit();   //reset circles
                     break;          //call out once in each frame
                 }
             }
         }

     }
```

This function does detection for each circle. If the player’s detected region is reaching one of the circles closer and closer, the nearToCir index changes from 0->2->1. Eventually, circles will be reset and the score will be increased by one now when the player touches the target circle staying in its depth. Or if the one player touches are not the target, the score will be reduced by one and the circle will also be reset.

Update of Circles



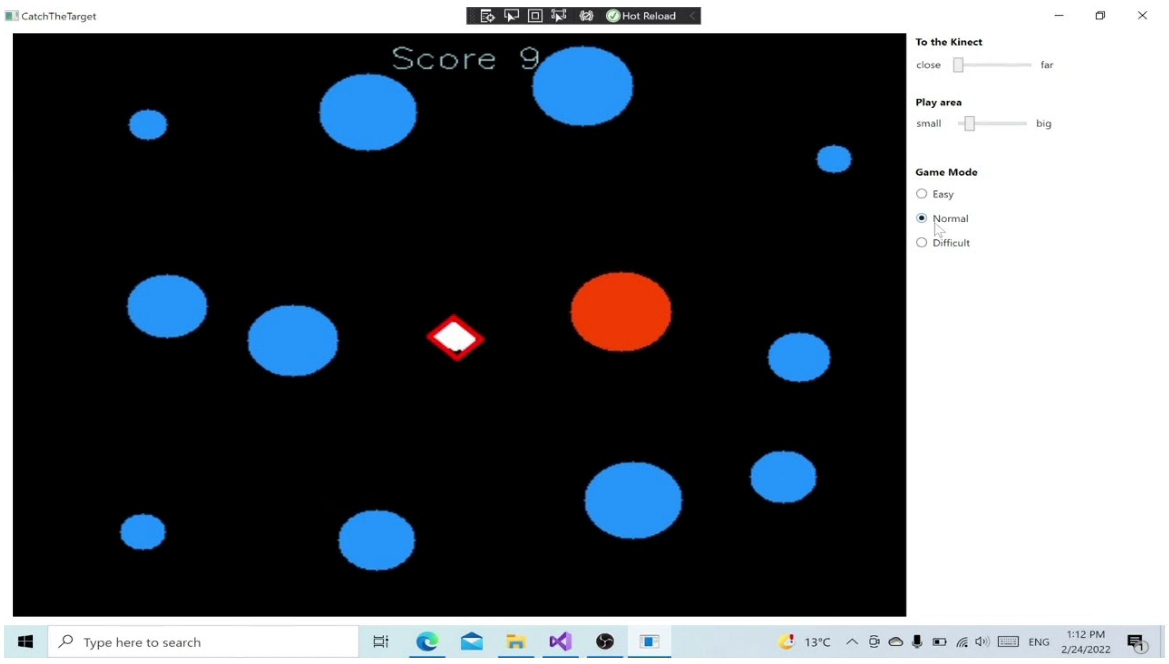
```
ference
ivate void drawCirlces()

 for(int i = 0; i<NumOfCircle; i++)
 {
     //openCVImg.Draw(CircleDetectionRegion[i], new Bgr(c), 0);  //for debug use
     if (i == Target)
     {
         if(nearToCir[i] == 0)
         {
             openCVImg.Draw(circle[i], new Bgr(Targetc), 0);
            /* String s = String.Format("{0:0}", circleDepth[i]/100);
             openCVImg.Draw(s, new System.Drawing.Point((int)circle[i].Center.X, (int)circle[i].Center.Y), new FontFace(), 0.4,
                 new Bgr(System.Drawing.Color.Yellow)); */

         }
         else if(nearToCir[i] == 1)
         {
             openCVImg.Draw(circle[i], new Bgr(DangerTargetc), 0);
            /* String s = String.Format("{0:0}", circleDepth[i]/100);
             openCVImg.Draw(s, new System.Drawing.Point((int)circle[i].Center.X, (int)circle[i].Center.Y), new FontFace(), 0.4,
                 new Bgr(System.Drawing.Color.Yellow)); */
         }
         else
         {
             openCVImg.Draw(circle[i], new Bgr(LittleDangerTargetc), 0);
            /* String s = String.Format("{0:0}", circleDepth[i] / 100);
             openCVImg.Draw(s, new System.Drawing.Point((int)circle[i].Center.X, (int)circle[i].Center.Y), new FontFace(), 0.4,
                 new Bgr(System.Drawing.Color.Yellow)); */
         }
```

The colors of circles on the screen is determined by the index of nearToCir. When the player is close to the circles, the colors of them turn darker and darker.

Future Works

More possibilities of NUI will be explored in the future. Skeleton-based detection, for example, can be implemented to allow computers to understand the body language of humans, which enables more kinds of applications to be developed.

Demo Video


CatchTheTarget

Score 9

To the Kinect
close — far

Play area
small — big

Game Mode
○ Easy
[✓ Normal]
○ Difficult

https://youtu.be/Ta_QEuZlwvM

References

Code

T4_BlobDetection

T5_BodilyInteraction