Collision detection

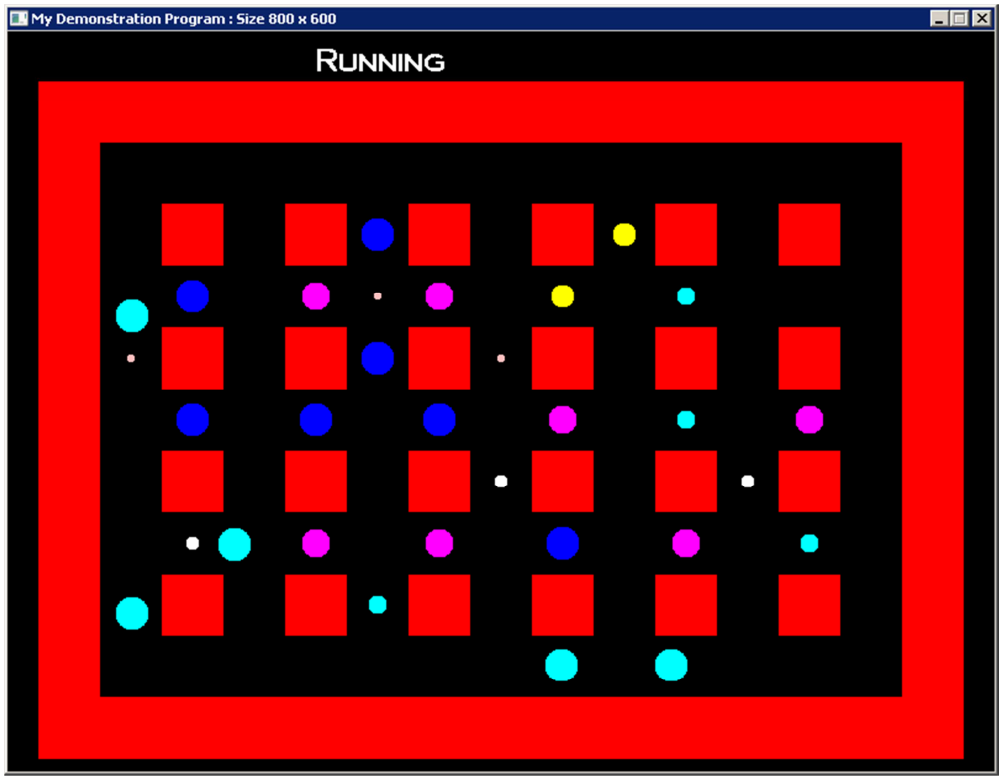
Please use Demo4 to follow this ‘tutorial. Enable it by following these instructions:

- Open mainfunction.cpp
- Include Demo4Main.h at the top
- Comment out whatever object you are currently creating. E.g. “BouncingBallMain oMain;”
- Add a new line saying:
  Demo4Main oMain;

i.e. it will now read:

```
// Needs one of the following #includes, to include the class definition
#include "BouncingBallMain.h"
#include "MyProjectMain.h"
#include "Demo2Main.h"
#include "Demo2aMain.h"
#include "Demo3Main.h"
#include "DemoAMain.h"
#include "Demo4Main.h"

#define BASE_SCREEN_WIDTH 800
#define BASE_SCREEN_HEIGHT 600



int main(int argc, char *argv[])
{
	int iResult;

	// Needs just one of the two following lines:
	//BouncingBallMain oMain;
	//MyProjectMain oMain;
	//Demo2Main oMain;
	//Demo2aMain oMain;
	//Demo3Main oMain;
	//DemoAMain oMain;
	Demo4Main oMain;
```

- Build the program
- Execute the program and watch it working
- You should get something like on the next page. Press space to continue.
  If you do not get the text showing up then copy the font file from your debug directory to whatever your working directory is. Probably your src directory?



As the blue circles move around they expand and shrink.

When two blue circles collide then one of them ‘teleports’ to a new location around the edge.

All of this is done in Demo4Object.cpp

Demo4Object.cpp : the displayable object file

The objects shrink and grow because the radius of each object is determined by the current time.

See this code in Draw():

```
int iTick = m_pMainEngine->GetTime()/20; // 1 per 20ms
int iFrame = iTick % 30;
int iSize = 10 + iFrame;
if ( iFrame > 15 )
      iSize = 10 + (30-iFrame);
```

This code is responsible for working out a radius (iSize) based upon the current time. The radius will change by 1 pixel every 20 ticks. Try changing the 20 to 200 and see what happens.

Next it assumes that there will be 30 different size possibilities (i.e. frames/images to consider). The code below increases this to 300 if you want to try it. Note that the iFrame and (300-iFrame) now need dividing by 10 to ensure that the max size is not too big.

```
int iTick = m_pMainEngine->GetTime()/20; // 1 per 20ms
int iFrame = iTick % 300;
int iSize = 10 + iFrame/10;
if ( iFrame > 150 )
      iSize = 10 + (300-iFrame)/10;
```

The code assumes that for the first 15 frames the object is growing. For the next it is shrinking. So it adds either the frame number (if it is growing) or 30-FrameNumber (if it is shrinking) to a basic size to work out the radius.

Try some numbers from 0 to 30 to see what this does and why it makes them shrink and grow I turn.

Now that we understand how bid the objects are, we can look at how the collision detection is done.

Collision Detection

I will assume that if you are trying to collision detect circles, you know the length of the radius of each and the position of the centre of the circle.

Go to the Demo4Object::DoUpdate() function.

This code asks the game engine for information about all of the moving objects, one at a time:

```
GetDisplayableObject( iObjectId )
```

This code will get a pointer to each object in turn (including the current object itself):

```
DisplayableObject* pObject;
for ( int iObjectId = 0 ;
       (pObject = m_pMainEngine->GetDisplayableObject( iObjectId )
                  ) != NULL ;
      iObjectId++ )
{
```

For each object, the aim is to find out if it overlaps the current object.

Firstly, exclude the current object from consideration:

```
if ( pObject == this ) // This is us, skip it
      continue;
```

Next, find out how far apart the centres are along the x and y axis (take the position of one away from the other). This could be negative but we are about to square the values so it won't matter.

```
int iXDiff = pObject->GetXCentre() - m_iCurrentScreenX;
int iYDiff = pObject->GetYCentre() - m_iCurrentScreenY;
```

The definitions of the Get…Centre() functions assume that you have set up the drawing area correctly, and that your circle is centred on it:

```
/** Get the X centre position of the object */
int GetXCentre() { return m_iCurrentScreenX + m_iStartDrawPosX +
m_iDrawWidth/2; }

/** Get the Y centre position of the object */
int GetYCentre() { return m_iCurrentScreenY + m_iStartDrawPosY +
m_iDrawHeight/2; }
```

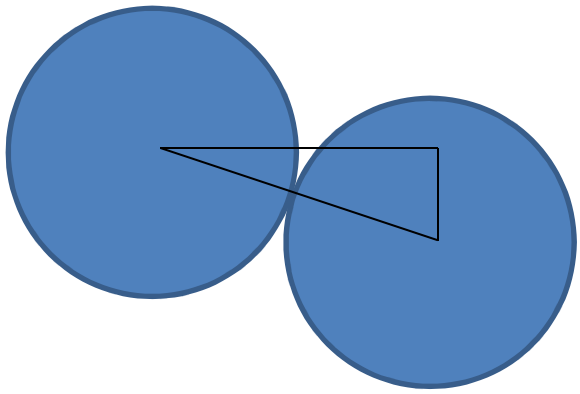
The next part of the code works out how big the circles are at the moment, using the same code as was used to draw them:

```
// Estimate the size - by re-calculating it
int iTick = iCurrentTime/20; // 1 per 20ms
```

```
int iFrame = iTick % 30;
int iSize = 10 + iFrame;
if ( iFrame > 15 )
      iSize = 10 + (30-iFrame);
int iSizeOther = iSize; // Assume both the same size
```

You can have `iSize` and `iSizeOther` different if you wish, but I know that all circles in my demo game are the same size.

Now we know all of the information we need to know:



- We know the horizontal distance between the centres (`iXDiff`).
- We know the vertical distance between the centres (`iYDiff`).
- We can work out how far apart the centres are (use Pythagoras's theorem).
- The interesting thing about circles is that every point on the edge is the same distance away from the centre.
- So if the edges are touching, the distance apart will be `iSizeOther + iSize`. If it is less than this then they are overlapping.
- What we really want to know then is whether the distance apart is less than `iSizeOther+iSize`.

i.e.: $x^2 + y^2 = z^2$, where z is the distance apart. So if z^2 is less than min_distance^2 they are overlapping.

```
if ( ((iXDiff*iXDiff)+(iYDiff*iYDiff)) < (iSizeOther+iSize)*(iSizeOther+iSize)) )
```

Where the first part the distance they really are apart (calculated using pythagoras' theorem) and the second part is the distance that they are allowed to be apart.

Strictly speaking, we should check whether the square root of the first part (i.e. the length of the hypotenuse) is less than `(iXDiff*iXDiff)+(iYDiff*iYDiff)),` but working out square roots can be expensive for computers so it is easier just to square both sides.

If they collide, then appropriate action is taken. Hopefully you can work out the rest ☺