

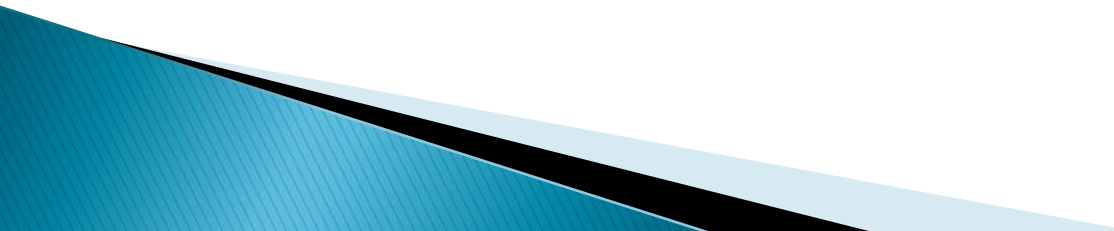
# GUI design in C++

Using MFC (Microsoft Foundation Classes)

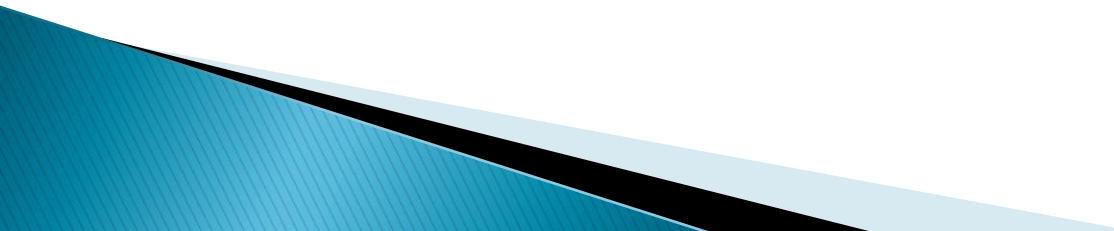
By: Rishinder Paul

Drawing basic shapes &  
Strategies for your project

# Contents

- ▶ Recap from previous slides.
  - ▶ The CPoint Class
  - ▶ The CRect Class
    - Drawing a Rectangle
    - Drawing a Circle/Ellipse
  - ▶ Understanding the CPen & CBrush Classes
  - ▶ Things to do first
    - Understanding the Game Map
    - Strategies for developing the Map
    - Choosing the best Strategy
- 

# Some recap.

- ▶ In the previous slides we learnt how to draw a line in MFC application.
  - ▶ We used the function `MoveTo(x,y)` to initialize the starting point.
  - ▶ We used The function `LineTo(x,y)` to mark the end point.
  - ▶ As both of these functions take 'x' and 'y' coordinates as arguments, we can just pass a single argument of the type `CPoint`, instead of two separate ones.
- 

# Understanding CPoint class

- ▶ CPoint constructor.
  - `CPoint p(50,70);`
- ▶ Constructs a point with coordinate x as 50 and y as 70.
- ▶ Hence to draw the same line using the CPoint object, we write the following code:
  - `CPoint p1(50,50), P2(100,100);`
  - `pDC->MoveTo(p1);`
  - `pDC->LineTo(p2);`

# CPoint Arithmetic

- ▶ We can add/subtract one point from the other.
- ▶ Suppose we have a point  $p1(x1,y1)$  and another  $p2(x2,y2)$ .
- ▶  $p1 + p2$  means a point  $(x1 + x2, y1 + y2)$
- ▶  $p1 - p2$  means a point  $(x1 - x2, y1, y2)$

# Understanding CRect Class

- ▶ The CRect class deals with rectangles.
- ▶ Its constructor accepts the top-left and bottom-right corner points as arguments.
  - `CRect r(p1,p2);`
- ▶ To draw this rectangle, the code is:
  - `pDC->Rectangle(r);`

# Drawing a Rectangle

- ▶ In the OnDraw() function, we add the following code:
  - CPoint p1(50,50), p2(200,200);
  - CRect r(p1,p2);
  - pDC->Rectangle(r);

# Drawing a Circle/Ellipse

- ▶ We use the CRect object to create a circle or any other ellipse.
- ▶ The resulting ellipse resides inside the rectangle's boundaries i.e. it inscribes the rectangle.
- ▶ Following is the statement for invoking the Ellipse function of our drawing object:
  - `pDC->Ellipse(r);`



# CPen Class

- ▶ Pen defines how a particular shape will have its border i.e. its outline.
- ▶ We use the CPen object to create a pen for our use.
- ▶ Format for CPen constructor:
  - CPen(*Pen\_Type*, *Pen\_Width*, *Pen\_Color*)

# CBrush Class

- ▶ Brush defines how a particular shape will have its fill.
- ▶ We use the CBrush object to create a brush for our use.
- ▶ Format for CPen constructor:
  - CBrush(*Brush\_Color*)

# Red and Green 😊

- ▶ To select a **green** brush and a thin **red** pen, we can write the following code:
  - `CPen aPen(PS_SOLID,1,RGB(255,0,0));`
  - `CBrush aBrush(RGB(0,255,0));`
- ▶ `RGB()` lets us choose the type of color we need for our drawing.
- ▶ Before using any of the brushes or pens, we have to select them first by using:
  - `pDC->SelectObject(aPen);`

# Things to do First 😊

- ▶ Well, we cannot develop an application as long as we don't have the conceptual idea about its structure and functioning.
- ▶ For this we need to know:
  - What features are needed now.
  - What may be needed in the future.
  - How we should begin it.
  - How to finish it.

# GUI isn't everything ☹️

- ▶ We cannot create any application by just using GUI.
- ▶ First of all we have to prepare a design strategy.
- ▶ As we are using the Object Oriented Approach, our first concern should be to understand what all classes do we need to implement and what should be the relationships between them before diving into drawing and GUI. 😊

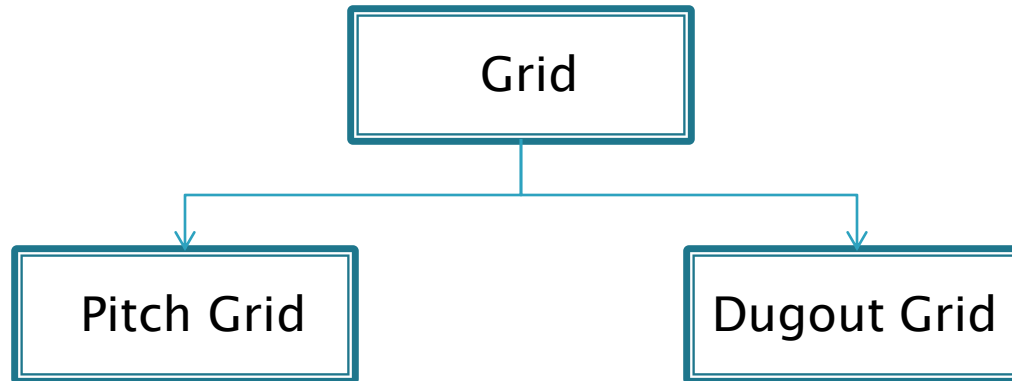
# Understanding The Game Map



# Design Strategies

- ▶ As there are multiple solutions to a single problem, there exist multiple strategies for developing our game map
- ▶ Strategy 1:
  - Have a class named Grid, which is the parent class for PitchGrid and DugoutGrid.
- ▶ Strategy 2:
  - Have two entirely different classes for Pitch and Dugout.
- ▶ Strategy 3:
  - Combination of Strategy 1 and Strategy 2

# Exploring Strategy 1



- ▶ Pitch and Dugout grids have mostly the same characteristics like edge-length, color, etc.
- ▶ One major difference between them is that dugout cannot contain the ball.
- ▶ Another difference is that Dugout cannot allow free player movements.



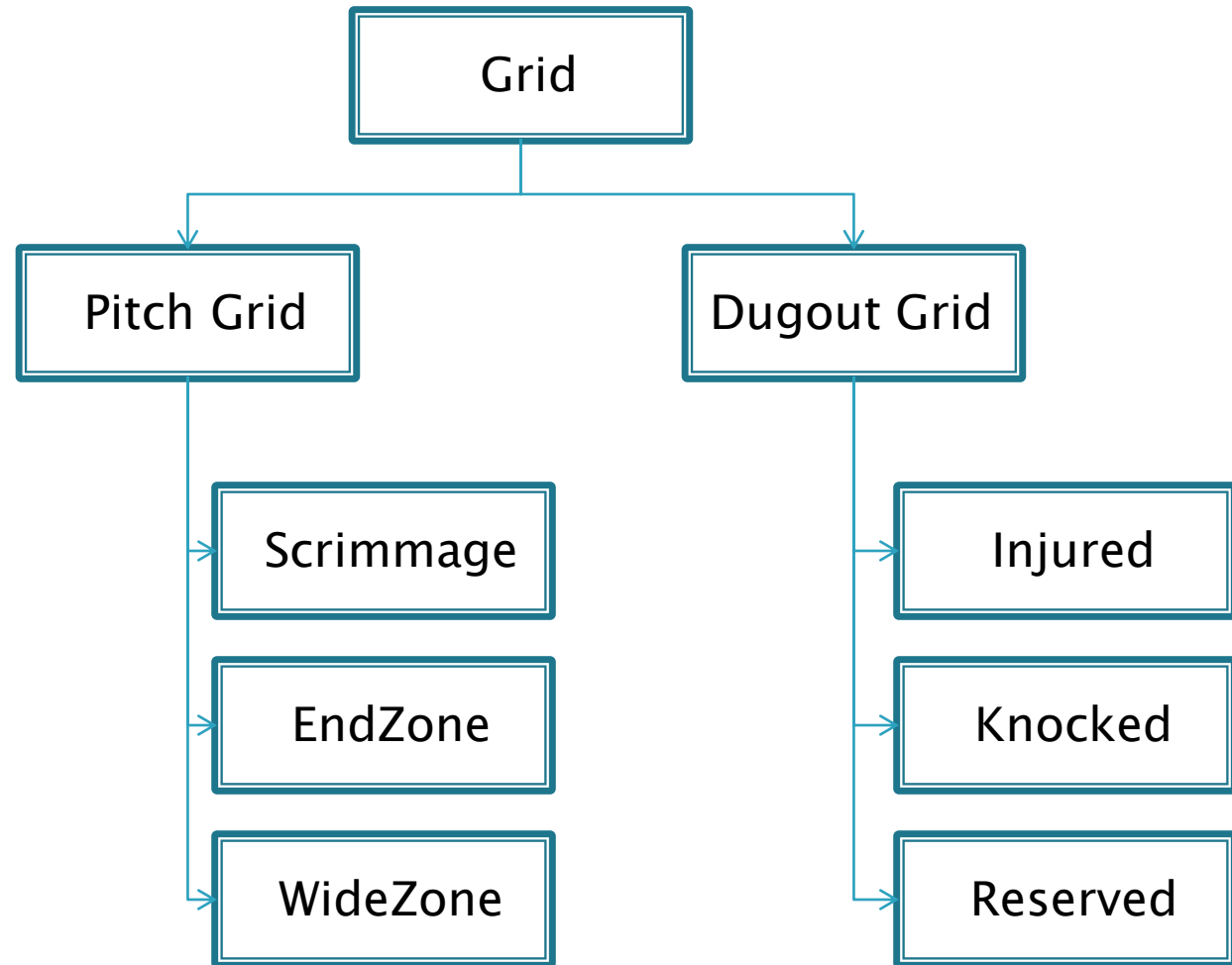
# Exploring Strategy 2

Pitch Grid

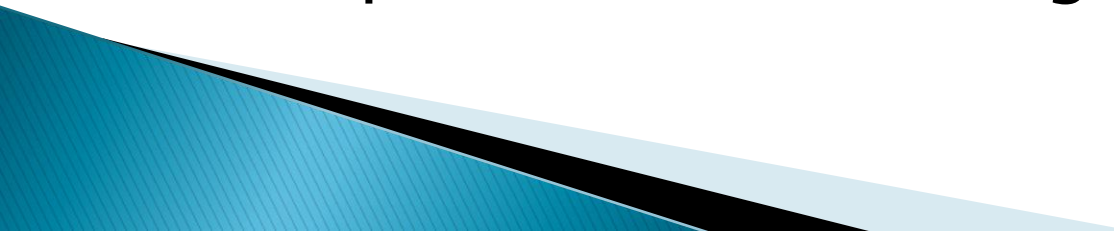
Dugout Grid

- ▶ Pitch and Dugout are two different classes, which do not share much in common.
- ▶ Pitch can become the parent class for the classes WideZone, EndZone and ScrimmageZone.
- ▶ Dugout can become the parent class for the classes Injured, Knocked and Reserved.

# Exploring Strategy 3

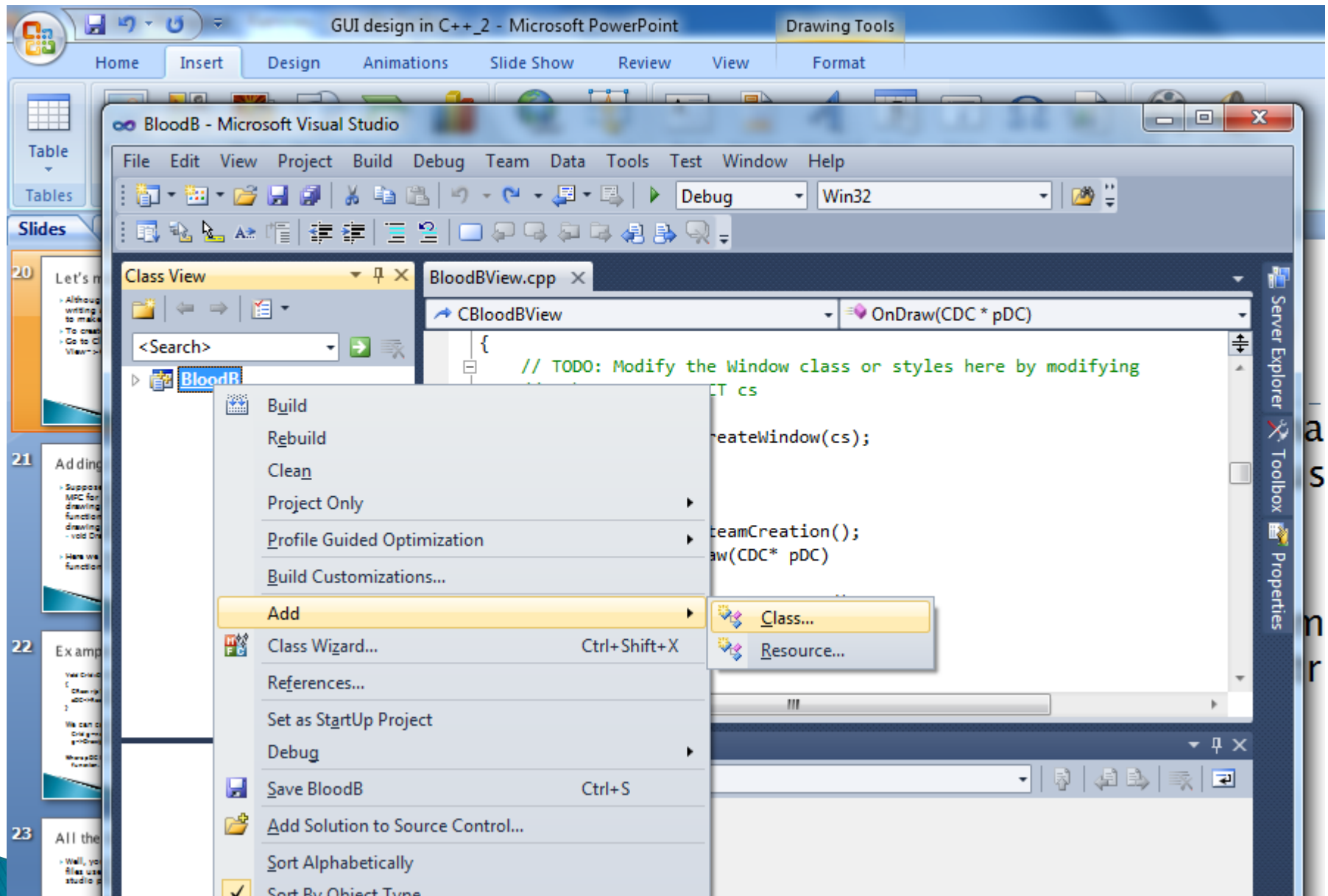


# GUI and our Strategy


- ▶ Once we have finalized which strategy will suit our project, we can start building the classes.
  - ▶ Suppose if we want to use strategy – 1 for our project. We have to design the classes and implement all the relationships between them.
  - ▶ Once our program structure is ready, we can now proceed to GUI design. 😊
- 

# Let's make it simpler 😊

- ▶ Although we can create classes manually by writing all the code, we can also use the IDE to make it more easy for us.
- ▶ To create a class:
  - Open class view from menu bar View->Class View or By hitting Ctrl+Shift+C.
  - Right Click on BloodB->Add->Class
  - Select C++ Class form the list and click “Add”
  - Choose a class name e.g. “Grid”. If this class is inherited from another class, just mention that class's name in the “base” space and click “Finish”



Generic C++ Class Wizard - BloodB

 **Welcome to the Generic C++ Class Wizard**

Class name:

.h file:  ...

.cpp file:  ...

Base class:

Access:  ▼

☐ Virtual destructor

☐ Inline

☐ Managed

Finish Cancel

# Adding GUI to our classes

- ▶ Suppose if we want our 'Grid' class to use MFC for drawing its objects, we have to add a drawing function in its definition. This function should accept an object of the drawing class as its argument. E.g.
  - `void Draw(CDC*);`
- ▶ Here we have chosen the name of the function as Draw.

# Using our class

- ▶ As soon as we create a class by using the class wizard, it creates a header file for that class with the same name. E.g. for our “Grid” class we will have a “Grid.h” file.
- ▶ To use this class we have to include it in our View class, where we are doing all our work.
- ▶ Hence, in the BloodBView.cpp file, along with other includes, we write: `#include “Grid.h”`



# Example Draw() function

```
Void Grid::Draw(CDC* aDC)
{
    CRect r(p1,p2);
    aDC->Rectangle(r);
}
```

We can call this function in our program as:

```
Grid* g=new Grid();
g->Draw(pDC);
```

Where pDC is the argument for our OnDraw() function in “BloodBView.cpp” file.

# All the best!

- ▶ Well, you can always ask for the source code files used in these slides, in the form of visual studio projects by email.
- ▶ You can always ask your questions by email. My email ID is [rishinder2007@yahoo.com](mailto:rishinder2007@yahoo.com)
- ▶ Try to go through MSDN's online tutorials to develop an understanding about Visual C++ and the MFC Libraries at:  
<http://msdn.microsoft.com/en-ca/visualc/bb496952.aspx>