

GUI design in C++

Using MFC (Microsoft Foundation Classes)

By: Rishinder Paul

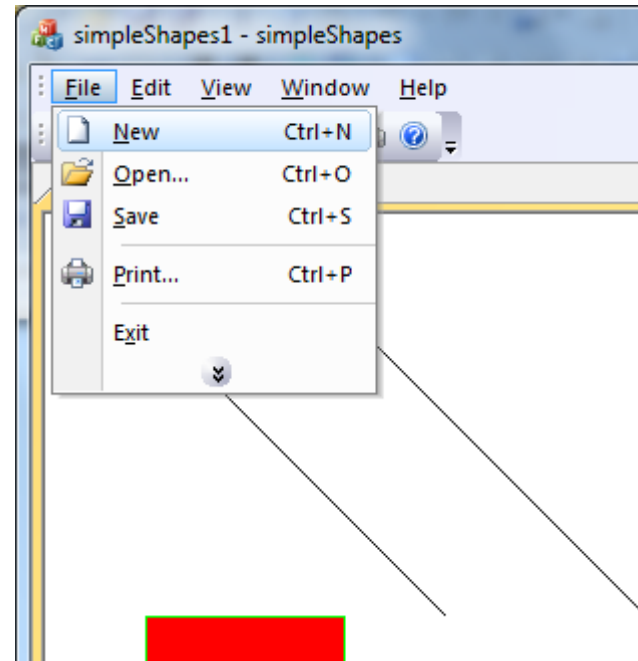
Menu, Mouse Events & Serialization

Contents

- ▶ Working with Menu
 - Editing the menu
 - Event-Handling
- ▶ Mouse Events
 - Default Right-Click
 - List of mouse events
 - Creating a new event
 - Understanding Mouse Coordinates
 - Some tips!
- ▶ Serialization
 - Setup.
 - Implementation.
 - Storing/Retrieving objects from files.

Menu Items – Intro

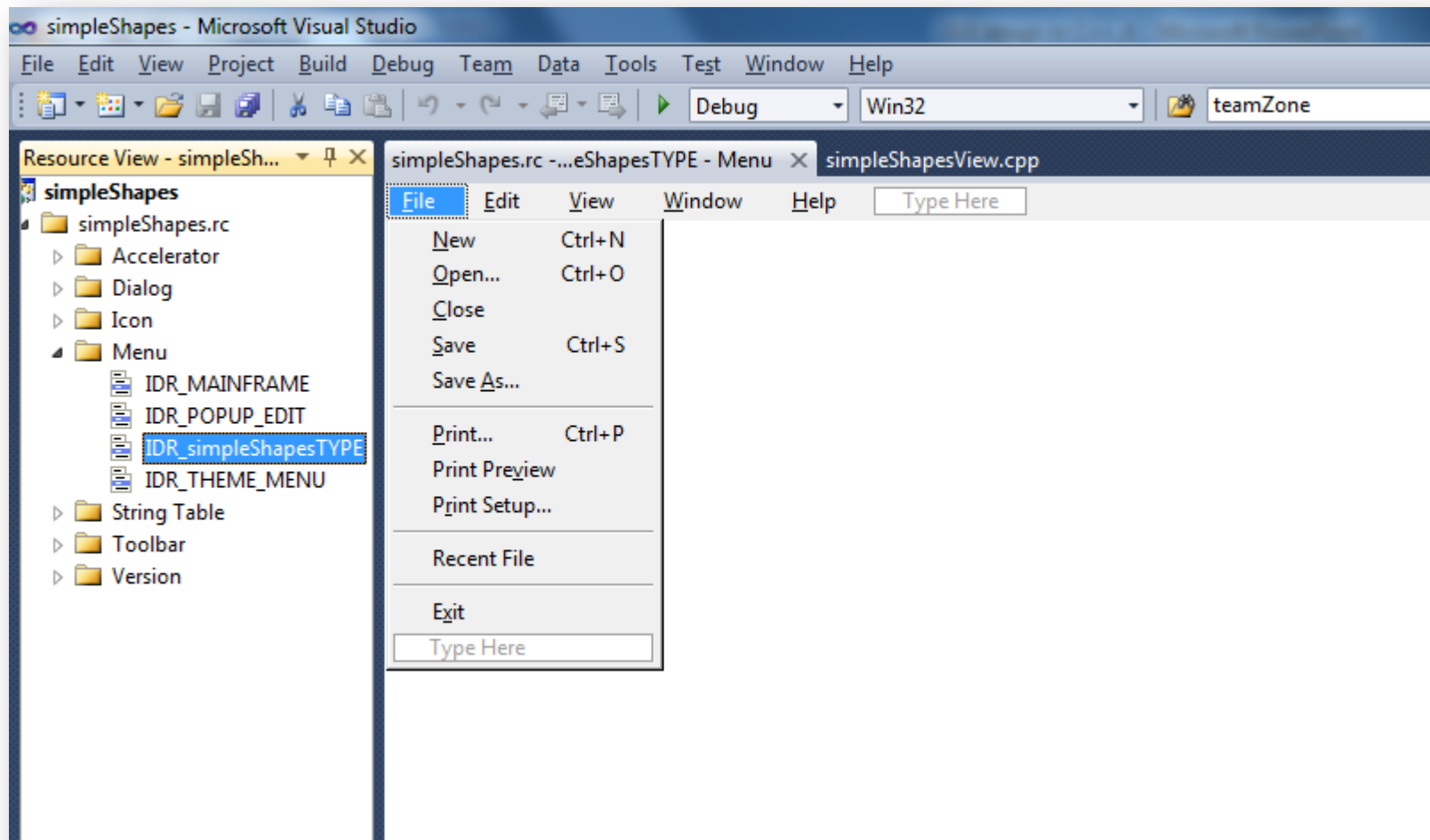
- ▶ Menu helps in grouping various functionalities of our application.
- ▶ There are some menu items provided in MFC by default.
- ▶ Our Goal is to create our own menu item and program it to do something, like displaying a **Message box**.
- ▶ For this tutorial, we will be using the simpleShapes Project as an example.



Resource View

- ▶ Open the Resource View (**Ctrl+Shift+E**).
- ▶ Expand the resource view tree.
- ▶ Expand the “Menu” Folder.
- ▶ Double click the Menu Item with name:
 - IDR_yourFileNameTYPE
 - IDR_simpleShapesTYPE in this case
- ▶ You will see the window with opened “File” menu.

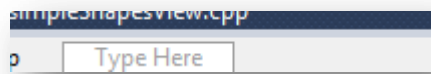
Default Menu



Editing the menu

- ▶ To add a new menu item:

- Click on the “Type Here” box.

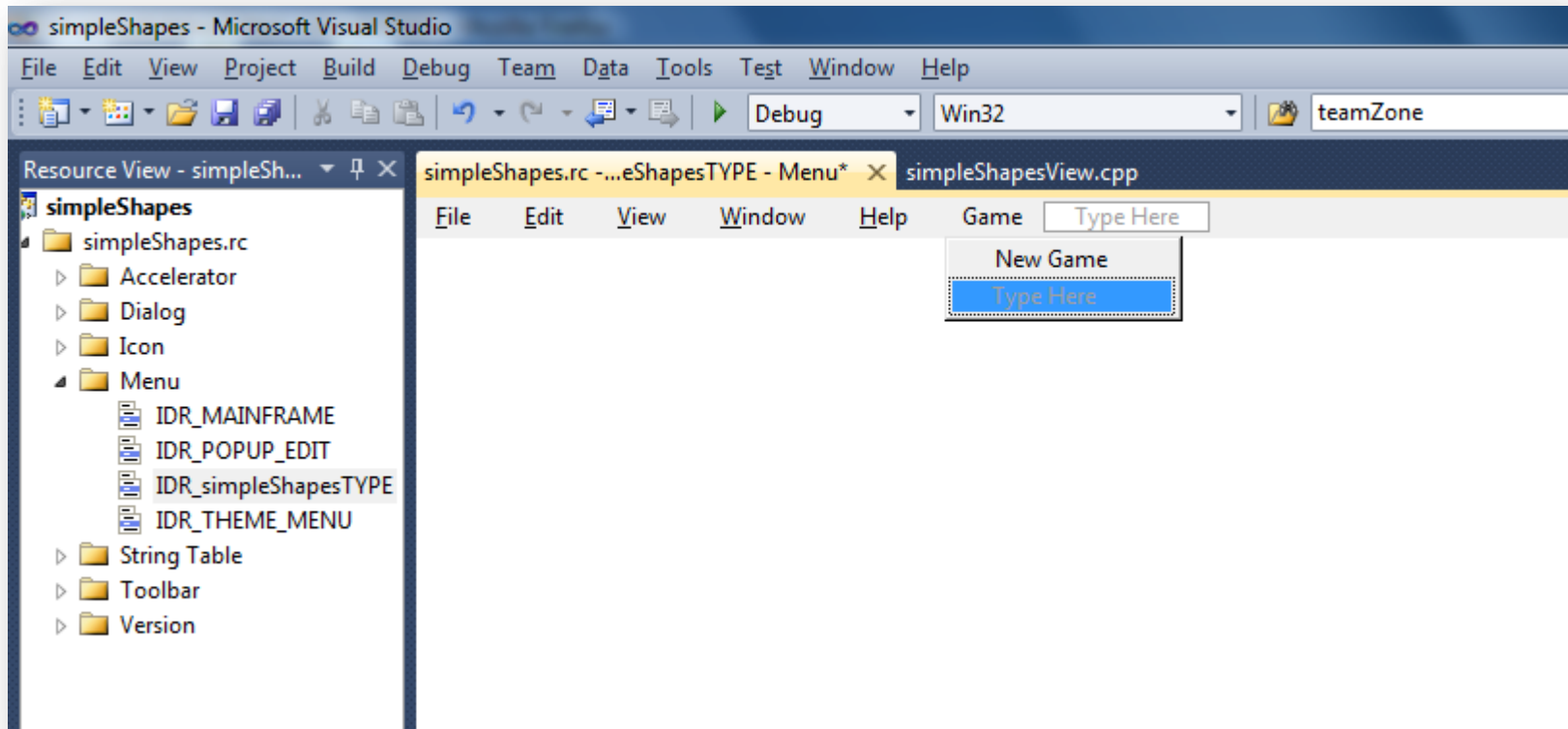


- Type the menu name, for example “Game”.
- Hit enter.

- ▶ To add a submenu:

- Click on the Newly created Menu Item.
- The same “Type Here” box will appear.
- Click on that box.
- Type the submenu name, example “New Team”.
- Hit Enter

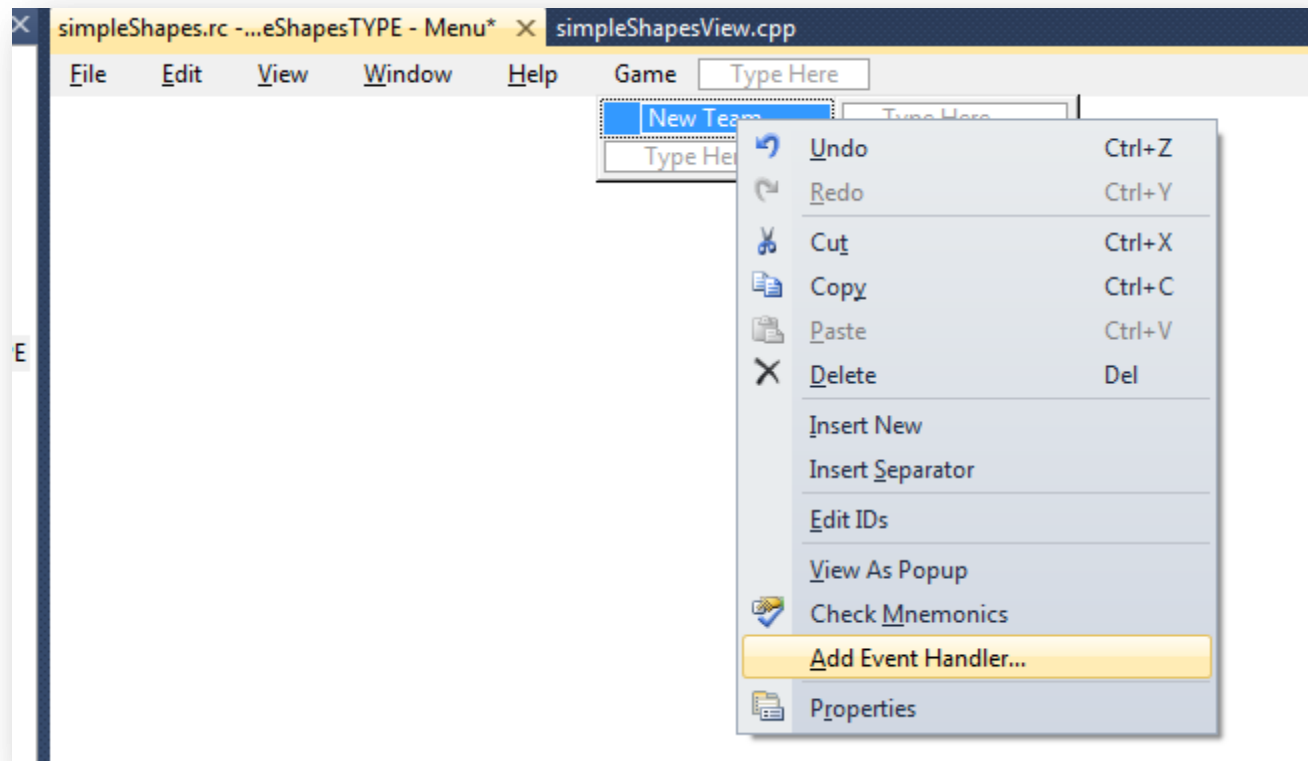
Screenshot!



Event Handling

- ▶ To make this menu function, we have to add an event-handler to it.
- ▶ For doing this, Right-Click the newly created menu item, i.e. “New Team”.
- ▶ Click on “Add Event Handler”.


Adding Event-Handler



Choosing Class for Events

- ▶ It's a good practice to have your events either in the Document or the View class.
- ▶ Here to choose View class for handling events, select the `CsimpleShapesView` from the class list on the Right Panel.
- ▶ The function handler name will be:
 - `OnGameNewteam`
- ▶ Let it be the same and click "Add and Edit".

Event Handler Wizard - simpleShapes



Welcome to the Event Handler Wizard

Command name:

Message type:

Function handler name:

Handler description:

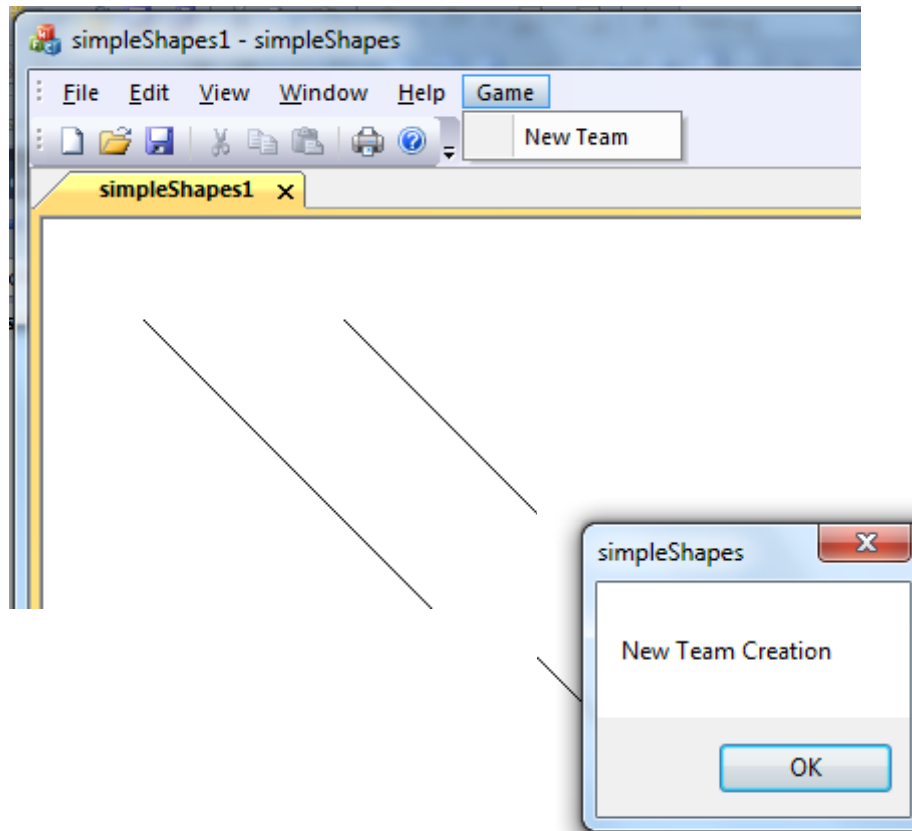
Class list:

Some Coding

- ▶ After clicking on “Add and Edit”, the code editor with the event–handler function will open.
- ▶ We can place our code in this function, which looks something like:

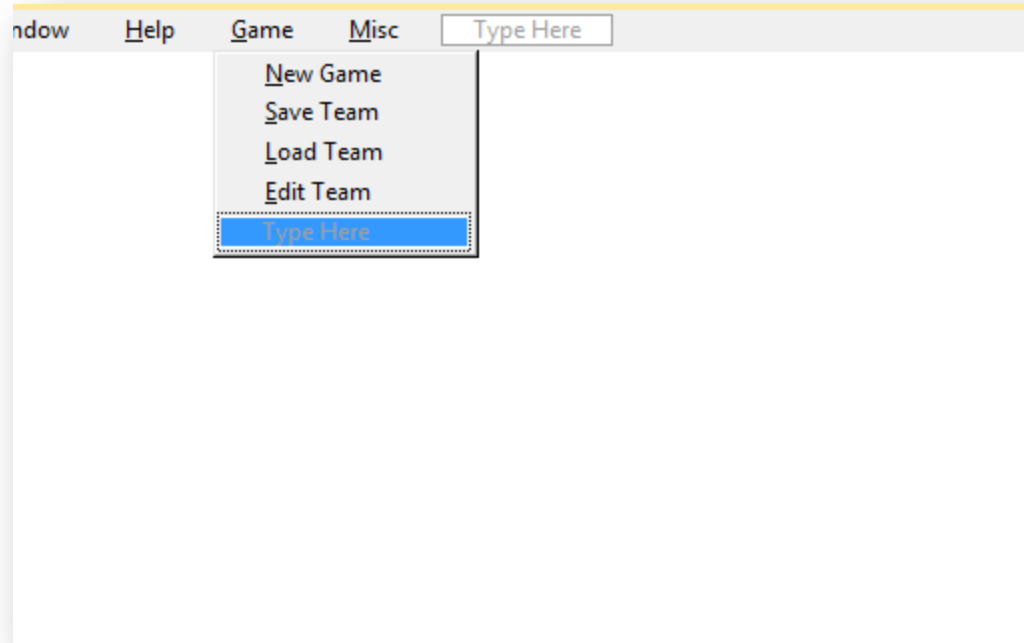
```
void CsimpleShapesView::OnGameNewteam()
{
    MessageBox(_T(“New Team Creation!”));
}
```

Result! 😊

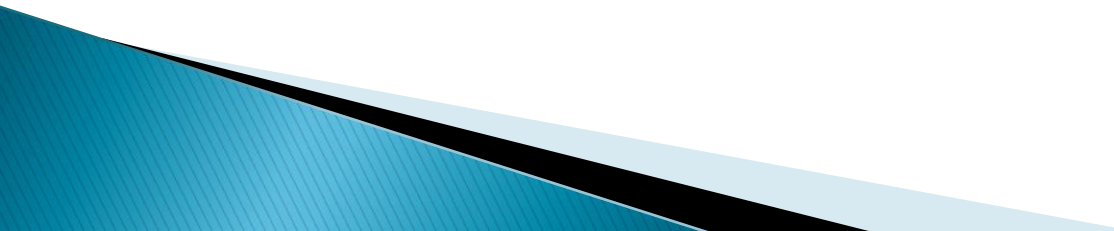


- ▶ **Exercise:** Try to invoke dialogs for team editor and player editor from the menu itself.

Our Goal 😐



Mouse Events

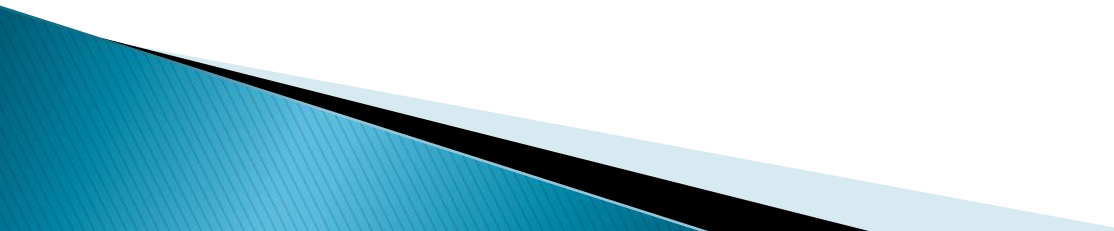
- ▶ In the following slides we'll learn how to respond to various mouse events like button-up, button-down and drag.
 - ▶ Here we'll use manual coding over widgets for handling mouse events.
 - ▶ Apply these techniques for controlling your game map and players.
- 

The default right-click

- ▶ Try to run your MFC application and right-click on the application window.
- ▶ You see a context-menu, this is a default mouse event provided to you by MFC.
- ▶ You can edit this mouse event by exploring “OnRButtonUp” function in your “view” file.

```
void CsimpleShapesView::OnRButtonUp(UINT /* nFlags */, CPoint point)
{
    ClientToScreen(&point);
    OnContextMenu(this, point);
}
```


The default right-click

- ▶ Try to place your own code in place of the highlighted code.
 - ▶ You'll see that once you run your application and perform a right-click in the window, your code will get executed.
 - ▶ To create a new handler for mouse-events, we need to understand what all events we can trigger from our mouse.
- 

List of Mouse Events

- ▶ These are the most commonly used mouse events:
 - Left-Button Up
 - Left-Button Down
 - Right-Button Up
 - Right-Button Down
 - On Mouse Move
- ▶ For now, let's us understand how to handle the Left-Button Up event.

Creating a new Event

- ▶ To handle our left-button up event, we need to create a handler for it first.
- ▶ For this follow these steps:
 - In the View file, go to the Message Map block.
 - Here we can see a handler for the right button-up.

```
BEGIN_MESSAGE_MAP(CsimpleShapesView, CView)
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, &CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, &CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, &CsimpleShapesView::OnFilePrintPreview)
    ON_WM_CONTEXTMENU()
    ON_WM_RBUTTONDOWN()
    ON_COMMAND(ID_GAME_NEWGAME, &CsimpleShapesView::OnGameNewteam)
END_MESSAGE_MAP()
```

Message Map

- ▶ Add the following statement for our event, within the Message Map:
 - ON_WM_LBUTTONDOWN()

```
IMPLEMENT_DYNCREATE(CsimpleShapesView, CView)

BEGIN_MESSAGE_MAP(CsimpleShapesView, CView)
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, &CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, &CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, &CsimpleShapesView::OnFilePrintPreview)
    ON_WM_CONTEXTMENU()
    ON_WM_RBUTTONDOWN()
    ON_WM_LBUTTONDOWN()
    ON_COMMAND(ID_GAME_NEWGAME, &CsimpleShapesView::OnGameNewteam)
END_MESSAGE_MAP()
```

Handler Function declaration

- ▶ Now we have to add a function declaration for this event in our View class header file (simpleShapesView.h)
- ▶ In the **protected** scope Add the following statement as a declaration:
 - ▶ `afx_msg void OnLButtonUp(UINT nFlags, CPoint point);`
- ▶ Now define this function in the View implementation file (simpleShapesView.cpp)

Handler Function Definition

- ▶ Let's invoke a message box for now. To do this use the following code:

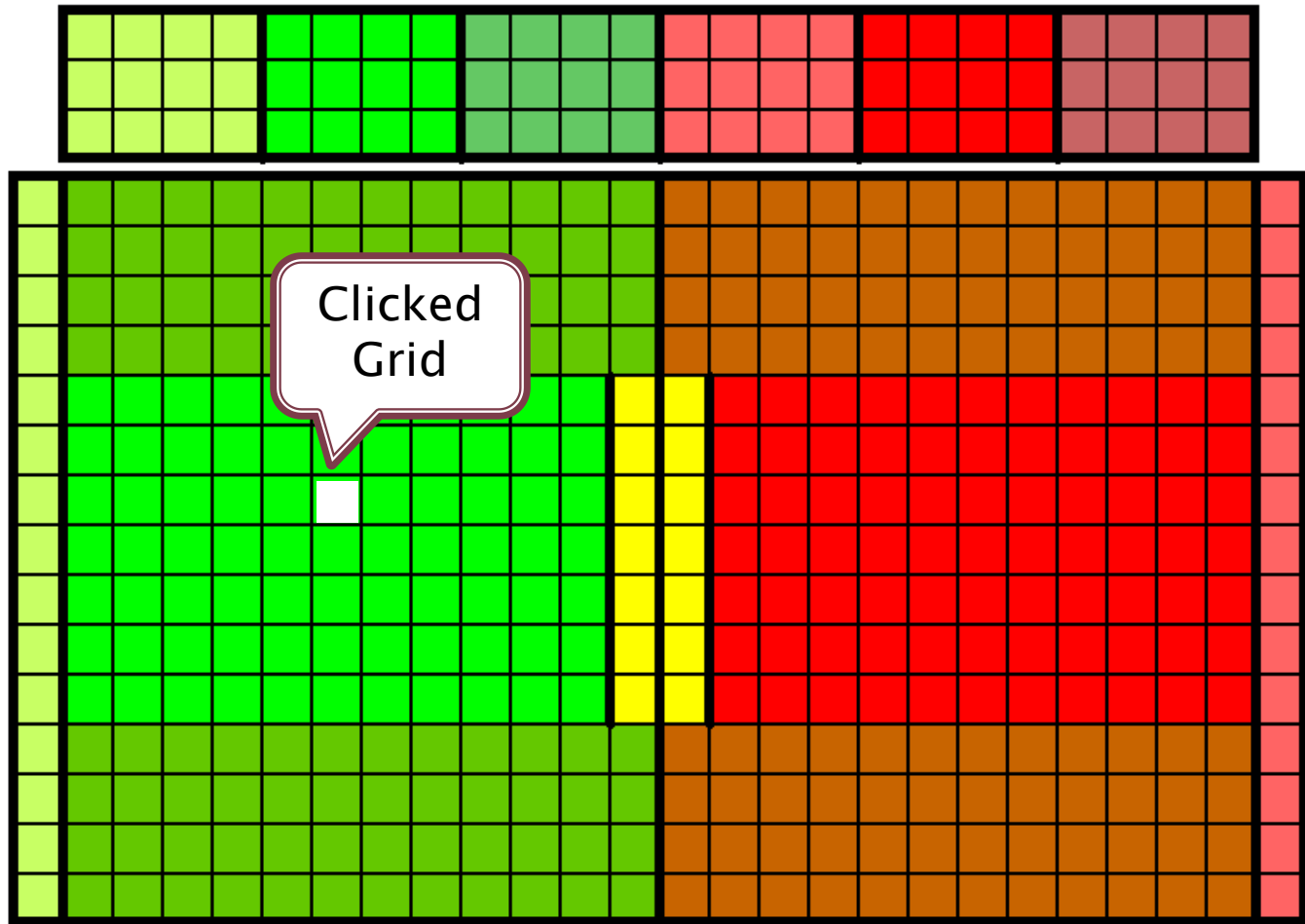
```
void CsimpleShapesView::OnLButtonUp(UINT /* nFlags */, CPoint point)
{
    MessageBox(_T("You clicked the Left button.));
}
```

- ▶ Now we are done with a basic left-button up event.

Understanding Mouse Coordinates

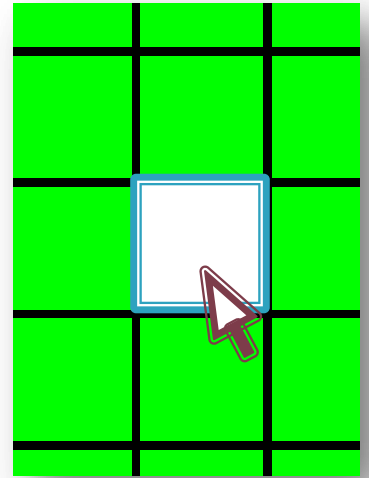
- ▶ In the handler function of our mouse, we can see an argument “point” of type CPoint.
- ▶ Why do we need this?
 - To know the exact location of our mouse at the time of this event.
- ▶ How can we use it?
 - In your game map, each object (grid and player) has it’s own coordinates/location.
 - To control them from mouse events, we can perform some kind of calculations based on this “point” argument to figure out if a particular object has been clicked or not.

Our Goal ☹️

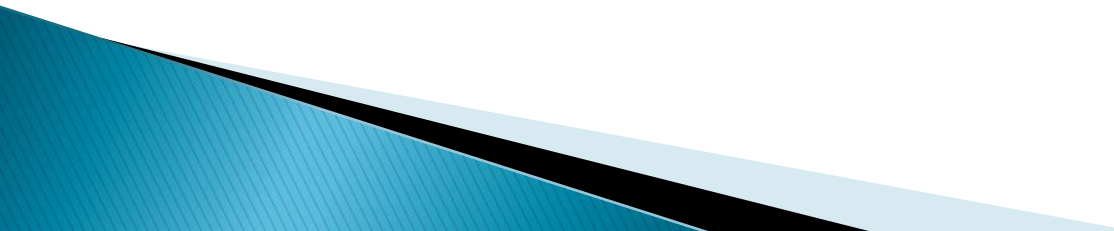


Some tips!

- ▶ To see if the mouse has clicked on the grid, some simple math can be used.
- ▶ As we know that we can get the mouse coordinates from our mouse handler.
- ▶ If the Top-Left corner of the grid has coordinates $(X1, Y1)$ and the edge-length is E .
- ▶ Lets say the our mouse has coordinates $(X2, Y2)$.
- ▶ Condition to check if the mouse has clicked within the grid:
 - If($x1 < x2 \ \&\& \ x1 > (x2 - E)$)
Clicked!



Serialization

- ▶ “Serialization is the process of converting our objects into sequence of bits, so that they can be stored in a file or memory buffer”. Wikipedia
 - ▶ Although we can store the information in files as text by using fstream or in Databases but this makes it more difficult to extract information directly in the form of objects.
 - ▶ We need some mechanism that can store the current state of objects and not the individual members of our object.
- 

Serialization in MFC

- ▶ Serialization can be done in MFC applications with ease.
- ▶ For this we need to follow these steps:
 - Make our Class inherit from the CObject class.
 - Add the DECLARE_SERIAL() macro to the class definition.
 - Declare Serialize() as a member function of your class.
 - Add the IMPLEMENT_SERIAL() macro to the class implementation file.
 - Implement the Serialize() function in your program.

Inheritance from CObject

- ▶ Lets consider an example of our “Team” class, which contains the members for storing the number of wins, loses, draws, etc.
- ▶ To serialize it, inherit it from CObject:
 - Class Team: public CObject
 - {
 -
 -
 - }

Declare Macro

- ▶ Within the class definition, add the following statement to the protected scope:
 - `DECLARE_SERIAL(Team);`
- ▶ Declare the function `Serialize()` in this class as follows:
 - `void Serialize(CArchive&);`
- ▶ Now we have the basic setup for serialization.

Example

- Class Team: public CObject
{

 protected:

 DECLARE_SERIAL(Team);
 public:

 void Serialize(CArchive&);
}

Class Implementation

- ▶ In the class implementation file, we have to declare the macro `IMPLEMENT_SERIAL()`
- ▶ This should be declared immediately after the includes i.e. it should look like this:

```
#include "Team.h"  
IMPLEMENT_SERIAL(Team,CObject,1);
```

```
....
```

```
....
```



Serialize() function definition

- ▶ Now we have to define the Serialize() function that we declared in the class definition.
- ▶ This function will specify, which members of the class do we need to serialize and store.
- ▶ We can include all or some of the class members that we need to store.

- ▶ The function should look like:
 - `void Team::Serialize(CArchive& archive)`
 - `{ //Your code goes here }`

Serialize() function detailed

```
void Team::Serialize( CArchive& archive )
{
    CObject::Serialize( archive );
    if( archive.IsStoring() )
        archive <<wins<<looses<<draws<<family<<totalPlayers;
    else
        archive >>wins>>looses>>draws>>family>>totalPlayers;
}
```

More details.

- ▶ `archive.IsStoring()` is used to know whether we want to store or retrieve the object.
- ▶ `archive<<`
 - This serializes the members for storing
- ▶ `archive>>`
 - This de-serializes the members for retrieving

To Use Serialize()

- ▶ In our program, we can invoke this function whenever we need this.
- ▶ Example invocation statement:
 - `teamA.Serialize();` //If teamA is not a pointer.
 - `teamA->Serialize();` //If teamA is a pointer.

Storing in Files

- ▶ The following syntax is an example for storing the teamA object in mySerial1.bbf file

```
CFile theFile;  
theFile.Open(_T("mySerial1.bbf"), CFile::modeWrite);  
CArchive archive(&theFile, CArchive::store);  
teamA->Serialize(archive);
```

Reading from the file

- ▶ The following syntax is an example for retrieving the `teamA` object from `mySerial1.bbf` file

```
CFile theFile;  
theFile.Open(_T("mySerial1.bbf"), CFile::modeRead);  
CArchive archive(&theFile, CArchive::load);  
teamA->Serialize(archive);  
teamA->displayDetails();
```

End Notes

- ▶ To know more about MFC, go through the MSDN tutorials at:
<http://msdn.microsoft.com/en-ca/visualc/bb496952.aspx>
- ▶ Try to use the right programming approach. If this is right, you will find it helpful in designing your User Interface.
- ▶ You can always ask your questions by mail at: rishinder2007@yahoo.com