Coursework summary

- ‘Using Microsoft Visual Studio’ document
- Game framework
  - Wraps up SDL (Simple DirectMedia Layer)
    - So you don’t need to learn SDL functions
  - Hides re-draw issues – to run relatively quickly
    - So you don’t need to spend hours debugging them
  - Very simple
    - You should be able to understand the code already
  - Avoids need for maths
    - Maths ability is not a pre-requisite for the course

Frameworks

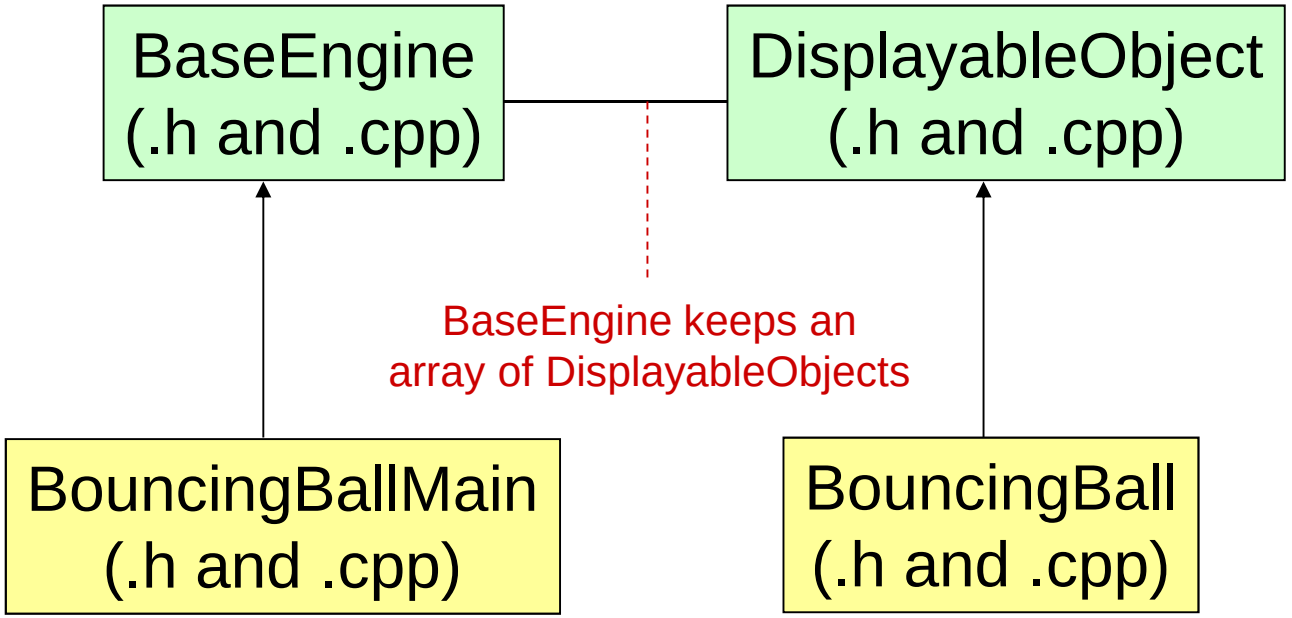
- C++ Frameworks often use inheritance
- Provide a generic framework to work within
- Allow specific functionality to be changed or added
- So does Java, e.g. with Java's **`Applet`**:
  - Override different functions to change the behaviour
    - Supplies methods for this purpose, e.g.:

    `init()`, **`paint()`**, **`start()`**, **`stop()`**, **`destroy()`**
  - Provide utility functions, called to do things, e.g.:

    `update()`, **`resize()`**, **`getParameter()`**, **`play()`**

C++ Coursework Framework

- Provides base classes which you sub-class to implement your own class



Rather than ‘custom’ functions, you use virtual functions to alter the behaviour.

How do you use the framework?

- Game framework links everything together
  - You just ‘tweak’ the behaviour to implement a game
- Override the functions that you want to change the behaviour of
- Base classes also provide various utility functions that your overrides can call to do work

Game initialisation

Initialise SDL

Create window

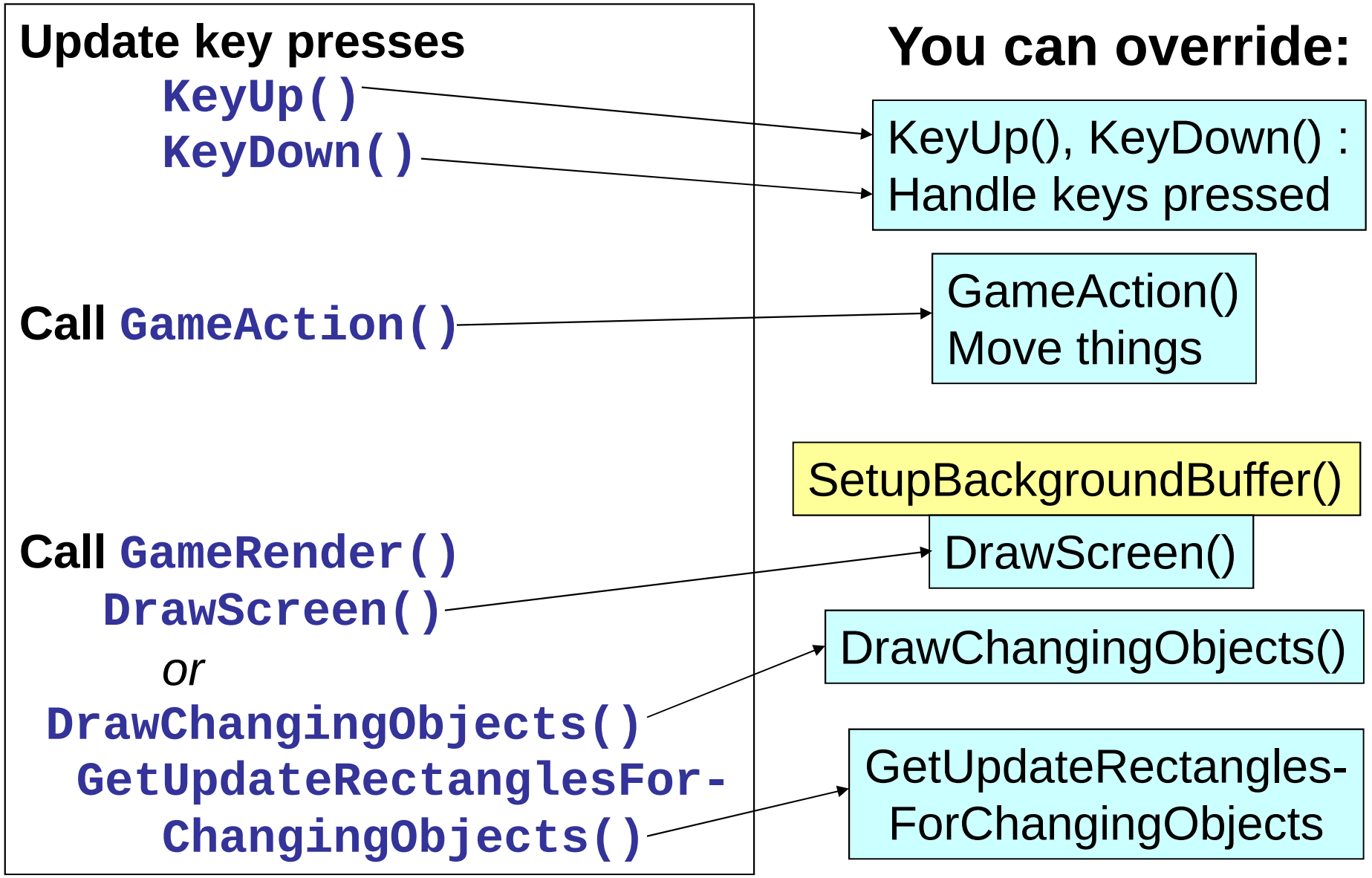
Create framework objects

Call **`GameInit()`**

You can override:

`GameInit()`
Do your own initialisation

Game main loop



Drawing

Drawing the whole window

Drawing only the moving objects

Useful functions

Drawing to the whole window

`GameInit()` calls `SetupBackgroundBuffer()`

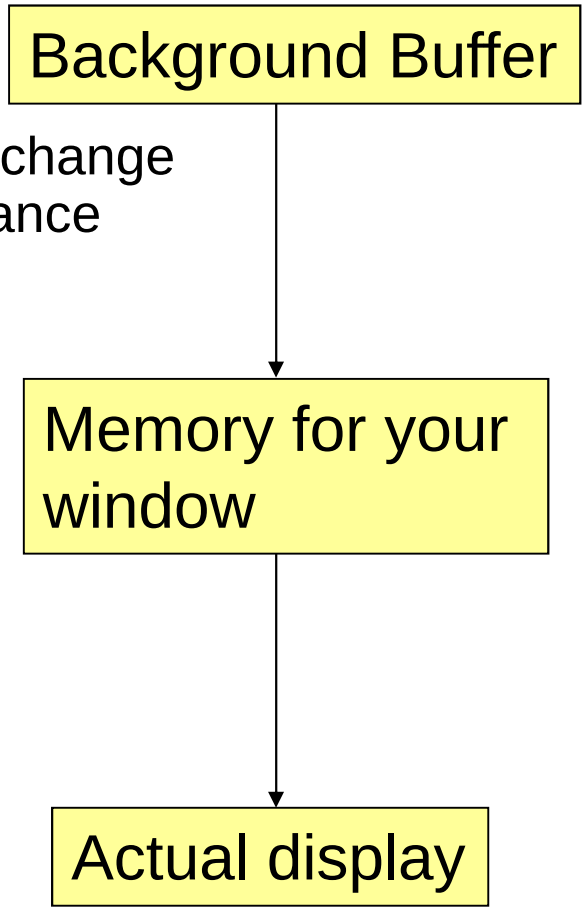
- You draw to the background buffer within SetupBackgroundBuffer()
- You call SetupBackgroundBuffer() after any big change (e.g. loading a new level), to update the appearance

`GameRender()` calls `DrawScreen()`

- Within **`DrawScreen()`** you should:
  - Call **`CopyAllBackgroundBuffer()`**
  - Optionally, draw any strings on top
  - And ask moving objects to draw themselves

`GameRender()` calls `SDL_UpdateRect()`

- To update entire screen



Re-drawing moving objects

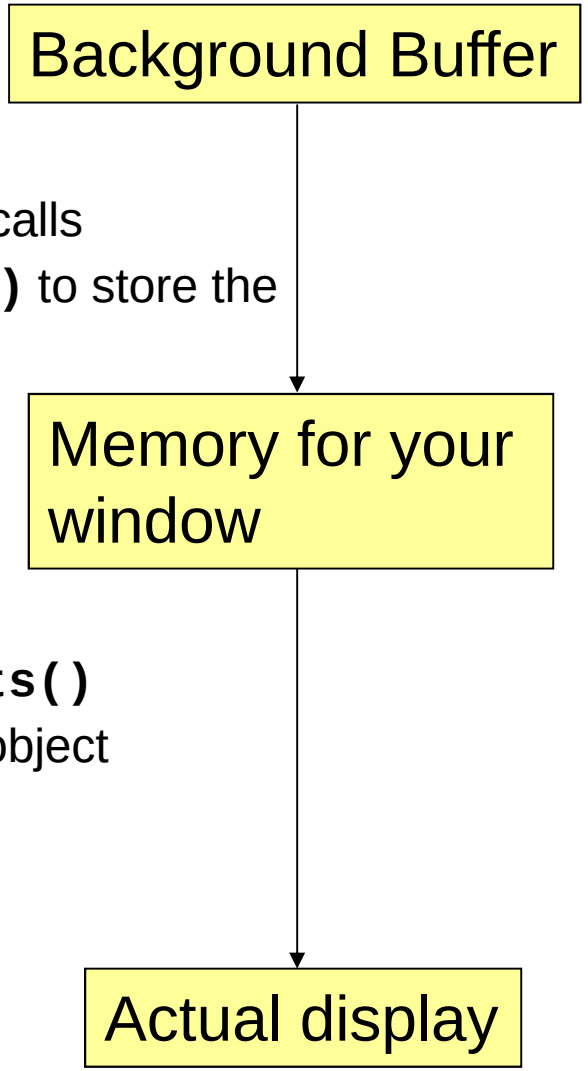
`GameRender()` calls `DrawChangingObjects()`

- Call **`UndrawChangingObjects()`**
  Draw the background over old draw position
- Call **`DrawChangingObjects()`**
  Calls **`Draw()`** on each object, which draws it and calls **`StoreLastScreenPositionAndUpdateRect()`** to store the position

Call **`DrawStrings()`**
Draw any information strings

`GameRender()` asks for areas which changed

- Calls **`GetUpdateRectanglesForChangingObjects()`** which calls **`GetRedrawRect()`** on each displayable object which uses values stored by **`StoreLastScreenPositionAndUpdateRect()`**
- **`GameRender()`** calls **`SDL_UpdateRects()`**
  - Update the areas that have changed



Useful Game Engine functions

- Is a key currently pressed?
  - **`IsKeyPressed(int iKeyCode)`**
    - Key-code is an SDL constant
- The screen has changed, redraw needed
  - **`Redraw(true)`** : redraw the whole screen
  - **`Redraw(false)`** : redraw just the moving objects
  - This is how **`GameRender()`** decides what to do
  - If you do not call **`Redraw()`** nothing will change
- **`SetScreenPixel( iX, iY, uiColour )`**
  - Draw a pixel on the screen

Setting pixel colour

- **`SetScreenPixel( iX, iY, uiColour )`**
  - Draw a pixel on the screen
  - **`SetScreenPixel( 4, 6, 0xFF0000 )`**

    Set pixel (4,6) to Red
  - **`SetScreenPixel( 134, 23, 0x808080 )`**

    Set pixel (134,23) to grey
- **`SafeSetScreenPixel(iX, iY, uiColour)`**
  - As **`SetScreenPixel`** but will verify that **`(iX,iY)`** is within the screen area
  - Slower, but writing outside of screen may corrupt your data or crash your program!

Colours

- Colours are specified by Red-Green-Blue (RGB) value, as in Java
- Colours are specified in 4 bytes of an unsigned int:
  - Highest Byte: Set it to zero
  - Next byte: Red element, 0-255
  - Next byte: Green element, 0-255
  - Lowest byte: Blue element, 0-255
- Hexadecimal no. has 2 digits per byte
  - In hex, colour is easy to express: 0xRRGGBB

DisplayableObject

DisplayableObject

- You will probably just have to implement three functions:
- **Constructor** : Initialise data
  - Initialise the drawing position variables
- **Draw()** : Draw the object, store the position at which it was drawn, calculate redraw region
  - You may not need to change this much
  - Use **SetScreenPixel()** to set the pixel colour
  - Also use **StoreLastScreenPositionAndUpdateRect()**
- **DoUpdate()** : Determine new values for the current position – i.e. implement the moves, handle player input, etc

DisplayableObject: member data

- Member data:
  - **m_iCurrentScreenX, m_iCurrentScreenY**
    - Position on the screen at which to draw
  - **m_iPreviousScreenX, m_iPreviousScreenY**
    - Previous position at which it was drawn, to undraw it later
  - **m_iStartDrawPosX, m_iStartDrawPosY**
    - Offset at which to actually draw, relative to top-left of area
  - **m_iDrawWidth, m_iDrawHeight**
    - Size of the thing being drawn, from the start draw position

Movement object

BouncingBall1

```
class BouncingBall1 : public BouncingBall
{
public:
   BouncingBall1(BouncingBallMain* pEngine, int iID, int
       iDrawType, int iSize, int iColour, char* szLabel,
       int iXLabelOffset, int iYLabelOffset, TileManager*
       pTileManager );

   void SetMovement(
       int iStartTime, int iEndTime, int iCurrentTime,
       int iStartX, int iStartY, int iEndX, int iEndY );

   void DoUpdate( int iCurrentTime );

protected:
   /* Movement position calculator */
   MovementPosition m_oMovement;

   // Pointer to the tile manager
   TileManager* m_pTileManager;
};
```

Using the Movement object

- Allows a caller to specify where the object will move from and to and when.
- **Setup()** sets up a new movement:
  - Start position (x and y), End position (x and y), Start time, End time
- **Calculate()** sets up an internal x and y member according to the time
- **GetX()** and **GetY()** retrieve the calculated time
- **HasMovementFinished(iCurrentTime)** returns true if move completed
- **Reverse()** reverses the x and y coordinates, and updates times to reverse the move

```
void BouncingBall1::SetMovement( int iStartTime, int
   iEndTime, int iCurrentTime,
    int iStartX, int iStartY, int iEndX, int iEndY )
{
   m_oMovement.Setup( iStartX, iStartY, iEndX, iEndY,
       iStartTime, iEndTime );
   m_oMovement.Calculate( iCurrentTime );
   m_iCurrentScreenX = m_oMovement.GetX();
   m_iCurrentScreenY = m_oMovement.GetY();
}
```

The tile-based approach

Tiles

Tile based games assume a rectangular map consisting of a grid of tiles

- Each tile has a type
- Type determines how it is drawn and whether it blocks movement
- e.g.
  ‘X’ = wall,
  ‘ ’ = passage
  ‘-’ = pellet to eat

x coordinate

y coordinate

x=0 y=0	x=1 y=0	x=2 y=0	x=3 y=0	x=4 y=0
x=0 y=1	x=1 y=1	x=2 y=1	x=3 y=1	x=4 y=1
x=0 y=2	x=1 y=2	x=2 y=2	x=3 y=2	x=4 y=2
x=0 y=3	x=1 y=3	x=2 y=3	x=3 y=3	x=4 y=3

BouncingBallMain.h

```
class BouncingBallMain :
public BaseEngine
{
protected:
  …

// A member object. Object is created when
  the BouncingBallMain is created
TileManager m;
```

BouncingBallMain.cpp

- Specify how many tiles wide and high

```
m.SetSize( 20, 20 );
```

- Specify the screen x,y of top left corner

```
m.SetBaseTilesPositionOnScreen( 250, 100 );
```

- Tell it to draw tiles
  from x1,y1 (i.e. 2,0) to x2,y2 (i.e. 17.19) in tile array,
  to the background of this screen

```
m.DrawAllTiles( this /*Engine*/,
  this->GetBackground() /*Or foreground*/,
  2, 0, 17, 19 );
```

BouncingBall – update tiles

- **Find the X value of the tile**

```
int iTileX = m_pTileManager->
    GetTileXForPositionOnScreen(m_iCurrentScreenX);
```

- **Find the Y value of the tile**

```
int iTileY = m_pTileManager->
    GetTileYForPositionOnScreen(m_iCurrentScreenY);
```

- **Get the value of that tile**

```
int iCurrentTile = m_pTileManager->
    GetValue( iTileX, iTileY );
```

- **Change the value of that tile and redraw it**

```
m_pTileManager->UpdateTile( GetEngine(), iTileX,
    iTileY, iCurrentTile+1 );
```