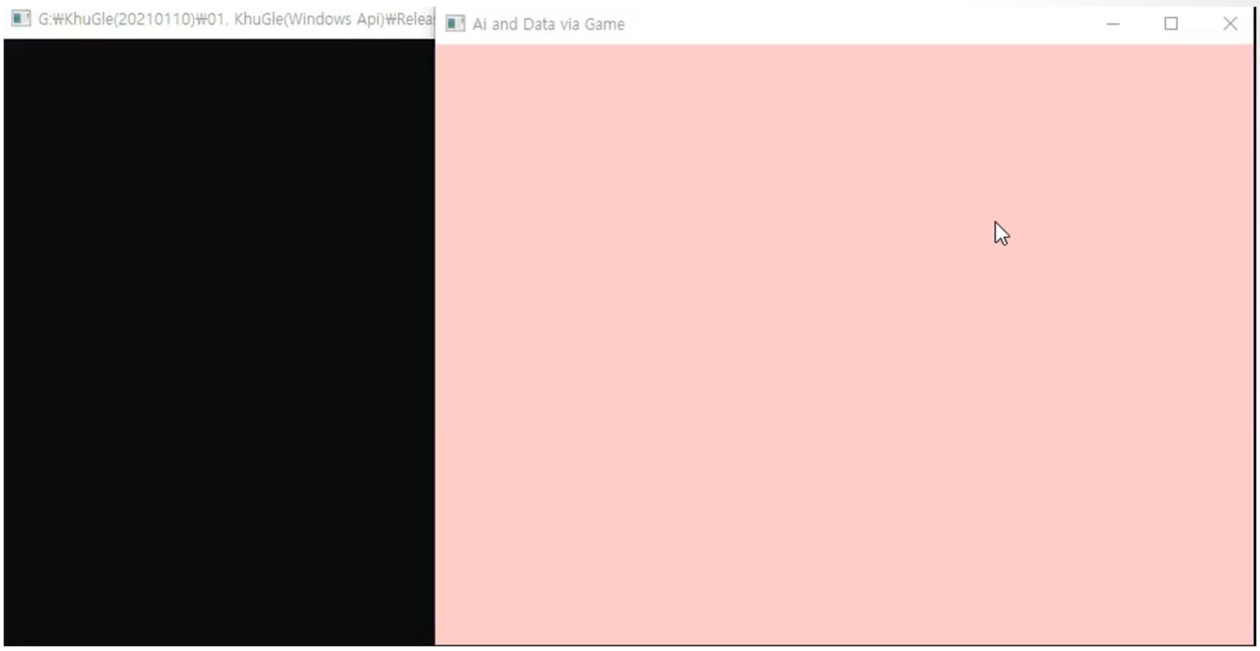
AI and Game Programming

Kyung Hee University
Daeho Lee

Schedule

- Windows API
- Game Layout
- Collision and Physics
- 3D Rendering
- Sound Processing
- Image Processing
- Correlation and Clustering
- Regression
- Performance Evaluation
- Perceptron
- MLP(DNN)
- CNN

1. Windows API



Project

- Win32 Console Application
- Setting
  - General
    - Project Default
      - Character Set: Use Multi-Byte Character Set
  - C/C++
    - General
      - SDL checks: No
  - Linker
    - System
      - SUBSYSTEM: WINDOWS
      - SUBSYSTEM : CONSOLE

WinMain

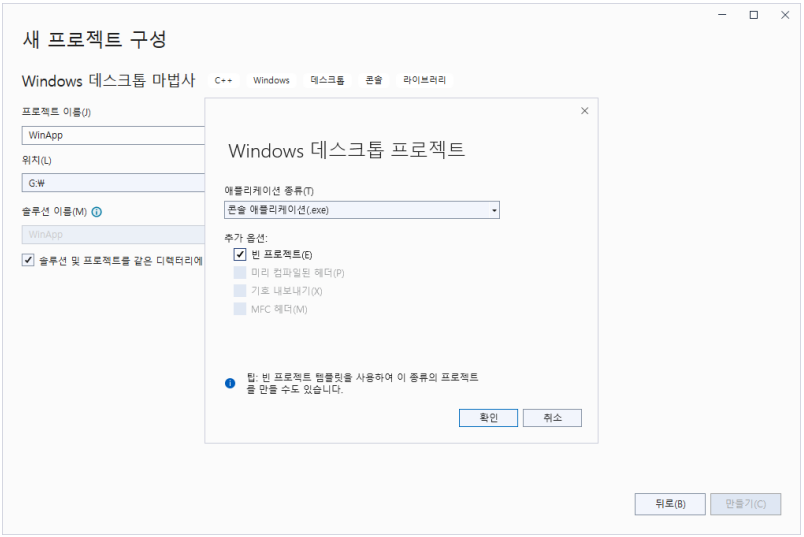
Windows API (1)

- API (application programming interface) that is used to create Windows applications
- Windows SDK (software development kit)

- int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow);

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    PWSTR szCmdLine, int CmdShow) {

    MessageBox(NULL, szCmdLine, "Title", MB_OK);

    return 0;
}
//MBCS (Multi-Byte Character Set) & Unicode Character Set
// "text" & L "text"
```

Windows API (2)



Windows API (3)

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  …
  switch (message) {
    case WM_CREATE: break;
    …
  }
  …
}
int APIENTRY WinMain(HINSTANCE hInstance,
  HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
  WNDCLASSEX windowClass;
  …
  windowClass.lpfnWndProc = WndProc;
  while(1) {
    if(PeekMessage(&msg, 0, 0, 0, PM_REMOVE)) {
        …
  }
}
```

WindowProc callback function
Message handling function

Windows API (4)

```
#include <windows.h>

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR pCmdLine,
int nCmdShow) {
    WNDCLASSEX windowClass;

    windowClass.cbSize = sizeof(WNDCLASSEX);
    windowClass.style = CS_HREDRAW | CS_VREDRAW;
    windowClass.lpfnWndProc = WindowProc;
    windowClass.cbClsExtra = 0;
    windowClass.cbWndExtra = 0;
    windowClass.hInstance = hInstance;
    windowClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    windowClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    windowClass.hbrBackground = NULL;
    windowClass.lpszMenuName = NULL;
    windowClass.lpszClassName = "WinApp Class";
    windowClass.hIconSm = LoadIcon(NULL, IDI_WINLOGO);

    if (!RegisterClassEx(&windowClass)) return 0;
```


WindowProc callback function
Message handling function

Windows API (5)

```
    HWND hwnd = CreateWindowEx(
        NULL,
        "WinApp Class",
        "Title of Program",
        WS_OVERLAPPEDWINDOW,
        // Size and position
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
        NULL,       // Parent window
        NULL,       // Menu
        hInstance,  // Instance handle
        NULL        // Additional application data
    );

    if (hwnd == NULL) return 0;

    ShowWindow(hwnd, SW_SHOW);// nCmdShow);

    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0) > 0) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}
```

WindowProc callback function
Message handling function

Windows API (6)

```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
    HBRUSH NewBrush = (HBRUSH)GetStockObject(GRAY_BRUSH);
    switch (uMsg) {
        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;

        case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hwnd, &ps);

            FillRect(hdc, &ps.rcPaint, NewBrush);
            Rectangle(hdc, 50, 50, 300, 200);
            RECT rt = { 0, 0, 500, 300 };
            DrawText(hdc, "WinApp", -1, &rt, DT_LEFT);

            EndPaint(hwnd, &ps);
        }
        return 0;
    }
    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}
```


Application class
– create window, message handling

KhuGleWin.h (1)

```
#pragma once
#include <windows.h>

class CKhuGleWin;

void KhuGleWinInit(CKhuGleWin *pApplication); // call WinMain (Global Function)

class CKhuGleWin {
public:
  HWND m_hWnd;
  int m_nW, m_nH;

  static CKhuGleWin *m_pWinApplication;

  int m_nDesOffsetX, m_nDesOffsetY;
  int m_nViewW, m_nViewH;
```

Application class
– create window, message handling

KhuGleWin.h (2)

```
  _int64 m_TimeCountFreq, m_TimeCountStart, m_TimeCountEnd;
  double m_Fps, m_ElapsedTime;

  bool m_bKeyPressed[256];
  bool m_bMousePressed[3];
  int m_MousePosX, m_MousePosY;

  WINDOWPLACEMENT m_wpPrev;

  static LRESULT CALLBACK WndProc(HWND hwnd, UINT message,  WPARAM wParam,
    LPARAM lParam);
  LRESULT CALLBACK WndProcInstanceMember(HWND hwnd,
    UINT message, WPARAM wParam, LPARAM lParam);

  void Fullscreen(); // Full screen, toggle
```


Application class
– create window, message handling

KhuGleWin.h (3)

```
  void GetFps();
  virtual void Update();  // called by 'WinMain' loop
  void OnPaint();         // Client paint

  void ToggleFpsView();

  CKhuGleWin(int nW, int nH);
  virtual ~CKhuGleWin();
  bool m_bViewFps;
};
```

Debugging new

KhuGleWin.cpp (1)

```
#include "KhuGleWin.h"
#include <cmath>
#include <cstdio>
#include <iostream>

#pragma warning(disable:4996)      // function, class member,
                                   //variable, or
                                   // typedef that's marked deprecated
#define _CRTDBG_MAP_ALLOC // memory leak detection
#include <cstdlib>        // _CrtDumpMemoryLeaks() is called in WinMAIN
#include <crtdbg.h>

#ifdef _DEBUG
#ifndef DBG_NEW
#define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
#define new DBG_NEW
#endif
#endif  // _DEBUG
```


WinMain call
Application class

KhuGleWin.cpp (2)

```
CKhuGleWin *CKhuGleWin::m_pWinApplication = 0;
void KhuGleWinInit(CKhuGleWin *pApplication) {
  CKhuGleWin::m_pWinApplication = pApplication;
  WinMain(0, 0, 0, 0);
}

CKhuGleWin::CKhuGleWin(int nW, int nH){
  m_nW = nW;      m_nH = nH;
  m_bViewFps = false;

  for(int i = 0 ; i < 256 ; ++i)
     m_bKeyPressed[i] = false; // m_bKeyPressed['A'], m_bKeyPressed[VK_LEFT]

  for(int i = 0 ; i < 3 ; ++i) // left, wheel (middle), right
     m_bMousePressed[i] = false;
}

CKhuGleWin::~CKhuGleWin() {
}
```

Application class
WndProc

KhuGleWin.cpp (3)

```
LRESULT CALLBACK CKhuGleWin::WndProc(HWND hwnd, UINT message, WPARAM wParam,
  LPARAM lParam){ // static
  return m_pWinApplication->WndProcInstanceMember(hwnd,
         message, wParam, lParam);
}

LRESULT CALLBACK CKhuGleWin::WndProcInstanceMember(HWND hwnd,
  UINT message, WPARAM wParam, LPARAM lParam)
{
  int width, height;
  HDC hdc;                  // Dc, Brush, Pen, font, …
  HBRUSH hBrushGray;
  RECT rt;

  hBrushGray = (HBRUSH)GetStockObject(GRAY_BRUSH);
```


Application class
WndProc

KhuGleWin.cpp (4)

```
  double AspectOrg, AspectWin;

  switch (message)   {
         case WM_CREATE:
           break;

         case WM_PAINT:
           OnPaint();
           break;

         case WM_CLOSE:
           PostQuitMessage(0);
           break;
```

Application class
WndProc

KhuGleWin.cpp (5)

```
case WM_SIZE:
       height = HIWORD(lParam);width = LOWORD(lParam);
       AspectOrg = (double)m_nW/(double)m_nH;
       AspectWin = (double)width/(double)height;
       m_nDesOffsetX = 0;        m_nDesOffsetY = 0;
       m_nViewW = width;m_nViewH = height;

       if(AspectWin > AspectOrg){
         m_nDesOffsetX =
                (int)((AspectWin-AspectOrg)*height/2.);
         m_nViewW =
                (int)(height*AspectOrg);
       }
       else{
         m_nDesOffsetY =
                (int)((1./AspectWin-1./AspectOrg)*width/2.);
         m_nViewH = (int)(width/AspectOrg);
       }
       break;
```


Application class
WndProc

KhuGleWin.cpp (6)

```
case WM_LBUTTONDOWN:
 m_MousePosX = (LOWORD(lParam)- m_nDesOffsetX)*m_nW/m_nViewW;
 m_MousePosY = (HIWORD(lParam)-m_nDesOffsetY)*m_nH/m_nViewH;
 m_bMousePressed[0] = true;
 break;

case WM_LBUTTONUP:
 m_MousePosX = (LOWORD(lParam)-m_nDesOffsetX)*m_nW/m_nViewW;
 m_MousePosY = (HIWORD(lParam)-m_nDesOffsetY)*m_nH/m_nViewH;
 m_bMousePressed[0] = false;
 break;
```

KhuGleWin.cpp (7)

```
case WM_MBUTTONDOWN:
 m_MousePosX = (LOWORD(lParam)-m_nDesOffsetX)*m_nW/m_nViewW;
 m_MousePosY = (HIWORD(lParam)-m_nDesOffsetY)*m_nH/m_nViewH;
 m_bMousePressed[1] = true;
 break;

case WM_MBUTTONUP:
 m_MousePosX = (LOWORD(lParam)-m_nDesOffsetX)*m_nW/m_nViewW;
 m_MousePosY = (HIWORD(lParam)-m_nDesOffsetY)*m_nH/m_nViewH;
 m_bMousePressed[1] = false;
 break;
```


KhuGleWin.cpp (8)

```
case WM_RBUTTONDOWN:
 m_MousePosX = (LOWORD(lParam)-m_nDesOffsetX)*m_nW/m_nViewW;
 m_MousePosY = (HIWORD(lParam)-m_nDesOffsetY)*m_nH/m_nViewH;
 m_bMousePressed[2] = true;
 break;

case WM_RBUTTONUP:
 m_MousePosX = (LOWORD(lParam)-m_nDesOffsetX)*m_nW/m_nViewW;
 m_MousePosY = (HIWORD(lParam)-m_nDesOffsetY)*m_nH/m_nViewH;
 m_bMousePressed[2] = false;
 break;
```

Application class
WndProc

KhuGleWin.cpp (9)

```
case WM_MOUSEMOVE:
 m_MousePosX = (LOWORD(lParam)-m_nDesOffsetX)*m_nW/m_nViewW;
 m_MousePosY = (HIWORD(lParam)-m_nDesOffsetY)*m_nH/m_nViewH;
 break;
case WM_KEYDOWN:
 switch (wParam){
       case VK_F11:
         Fullscreen();
         break;
       case VK_F12:
         ToggleFpsView();
         break;
       case VK_LEFT:
         break;
 }
 if(wParam >= 0 && wParam < 256)
       m_bKeyPressed[wParam] = true;
 break;
```


Application class
WndProc

KhuGleWin.cpp (10)

```
case WM_KEYUP:
 if(wParam >= 0 && wParam < 256)
       m_bKeyPressed[wParam] = false;
       break;

case WM_CHAR:
 switch (wParam) {
       case 'a':
        break;
 }
 break;
```

Application class
WndProc

KhuGleWin.cpp (11)

```
  case WM_ERASEBKGND:
   hdc = (HDC) wParam;
   GetClientRect(hwnd, &rt);
   SetMapMode(hdc, MM_ANISOTROPIC);
   SetWindowExtEx(hdc, 100, 100, NULL);
   SetViewportExtEx(hdc, rt.right, rt.bottom, NULL);
   FillRect(hdc, &rt, hBrushGray);
   break;

  default:
   break;
  }
  return (DefWindowProc(hwnd, message, wParam, lParam));
}
```


Application class
Full screen

KhuGleWin.cpp (12)

```
void CKhuGleWin::Fullscreen() {
  DWORD dwStyle = GetWindowLong(m_hWnd, GWL_STYLE);
  if(dwStyle & WS_OVERLAPPEDWINDOW) {
    m_wpPrev.length = sizeof(WINDOWPLACEMENT);
    MONITORINFO mi = {sizeof(MONITORINFO)};
    if(GetWindowPlacement(m_hWnd, &m_wpPrev) &&
      GetMonitorInfo(MonitorFromWindow(m_hWnd, MONITOR_DEFAULTTOPRIMARY), &mi))
    {
      SetWindowLong(m_hWnd, GWL_STYLE,
        dwStyle & ~WS_OVERLAPPEDWINDOW);
      SetWindowPos(m_hWnd, HWND_TOP,
        mi.rcMonitor.left, mi.rcMonitor.top,
        mi.rcMonitor.right - mi.rcMonitor.left,
        mi.rcMonitor.bottom - mi.rcMonitor.top,
        SWP_NOOWNERZORDER | SWP_FRAMECHANGED);
    }
  }
```

Application class
Full screen

KhuGleWin.cpp (13)

```
  else {
    SetWindowLong(m_hWnd, GWL_STYLE, dwStyle | WS_OVERLAPPEDWINDOW);
    SetWindowPlacement(m_hWnd, &m_wpPrev);
    SetWindowPos(m_hWnd, NULL, 0, 0, 0, 0,
      SWP_NOMOVE | SWP_NOSIZE | SWP_NOZORDER |
      SWP_NOOWNERZORDER | SWP_FRAMECHANGED);
  }
}
```


Application class
GetFps
Update

KhuGleWin.cpp (14)

```
void CKhuGleWin::GetFps() {
  QueryPerformanceCounter((LARGE_INTEGER*)&m_TimeCountEnd);
  m_ElapsedTime = (double)(m_TimeCountEnd -
        m_TimeCountStart)/(double)m_TimeCountFreq;
  m_TimeCountStart = m_TimeCountEnd;
  m_Fps = 1./m_ElapsedTime;
}
void CKhuGleWin::Update() { // called in WinMain
  RECT Rect;
  GetClientRect(m_hWnd, &Rect);
  InvalidateRect(m_pWinApplication->m_hWnd, &Rect, false);

  char strFps[200];
  sprintf(strFps, "FPS: %7.3lf, Elapsed time: %lf", m_Fps,  m_ElapsedTime);
  if(m_bViewFps){
    std::cout << strFps << std::endl;
  }
}
```

Application class
OnPaint

KhuGleWin.cpp (15)

```
void CKhuGleWin::OnPaint() {
  RECT Rect;
  GetClientRect(m_hWnd, &Rect);
  int nW = Rect.right-Rect.left;
  int nH = Rect.bottom-Rect.top;

  if(nW <= 0 || nH <= 0) return;

  PAINTSTRUCT ps;
  HDC hdc = BeginPaint(m_hWnd, &ps);
  HDC hDC, hCompDC;
  hDC = GetDC(m_hWnd);
  hCompDC = CreateCompatibleDC(hDC);

  HBITMAP hBitmap;
  hBitmap = CreateCompatibleBitmap(hDC, nW, nH);
  SelectObject(hCompDC, hBitmap);
```


Application class
OnPaint

KhuGleWin.cpp (16)

```
  BITMAPINFOHEADER bmiHeader;

  bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
  bmiHeader.biWidth = m_nW;
  bmiHeader.biHeight = m_nH;
  bmiHeader.biPlanes = 1;
  bmiHeader.biBitCount = 24;
  bmiHeader.biCompression = BI_RGB;
  bmiHeader.biSizeImage = (m_nW*3+3)/4*4   * m_nH;
  bmiHeader.biXPelsPerMeter = 2000;
  bmiHeader.biYPelsPerMeter = 2000;
  bmiHeader.biClrUsed = 0;
  bmiHeader.biClrImportant = 0;
```

Application class
OnPaint

KhuGleWin.cpp (17)

```
  unsigned char *Image2D24
     = new unsigned char [bmiHeader.biSizeImage];
  int x, y, Offset;

  for(y = 0 ; y < m_nH ; y++)
  {
    Offset = (m_nW*3+3)/4*4 * (m_nH-y-1);
         // RGB, BGR, 4byte aligned (each row), bottom-up
    for(x = 0 ; x < m_nW ; x++) {
         int Offset2 = Offset+x*3;

         Image2D24[Offset2++] = 200;        // B
         Image2D24[Offset2++] = 200;        // G
         Image2D24[Offset2] = 255;          // R
    }
  }
```


Application class
OnPaint

KhuGleWin.cpp (18)

```
  SetStretchBltMode(hCompDC, HALFTONE);

  StretchDIBits(hDC, m_nDesOffsetX, m_nDesOffsetY,
    m_nViewW, m_nViewH,0,0,
    bmiHeader.biWidth,bmiHeader.biHeight,
    Image2D24, (LPBITMAPINFO)&bmiHeader,
    DIB_RGB_COLORS,SRCCOPY);

  delete [] Image2D24;

  DeleteObject(hBitmap);

  DeleteDC(hCompDC);
  ReleaseDC(m_hWnd, hDC);

  EndPaint(m_hWnd, &ps);
}
```

KhuGleWin.cpp (19)

```
int APIENTRY WinMain(HINSTANCE hInstance,
  HINSTANCE hPrevInstance, LPSTR lpCmdLine,
  int nCmdShow)
{
  if(!CKhuGleWin::m_pWinApplication) return -1;

  WNDCLASSEX windowClass;
  MSG msg;
  DWORD    dwExStyle;
  DWORD    dwStyle;
  RECT     windowRect;

  int width = CKhuGleWin::m_pWinApplication->m_nW;
  int height = CKhuGleWin::m_pWinApplication->m_nH;
```


KhuGleWin.cpp (20)

```
  windowRect.left = (long)0;
  windowRect.right = (long)width;
  windowRect.top = (long)0;
  windowRect.bottom = (long)height;

  windowClass.cbSize = sizeof(WNDCLASSEX);
  windowClass.style = CS_HREDRAW | CS_VREDRAW;
  windowClass.lpfnWndProc
    = CKhuGleWin::m_pWinApplication->WndProc;
  windowClass.cbClsExtra = 0;
  windowClass.cbWndExtra = 0;
  windowClass.hInstance = hInstance;
  windowClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
  windowClass.hCursor = LoadCursor(NULL, IDC_ARROW);
  windowClass.hbrBackground = NULL;
  windowClass.lpszMenuName = NULL;
  windowClass.lpszClassName = "WinClass";
  windowClass.hIconSm = LoadIcon(NULL, IDI_WINLOGO);
```

KhuGleWin.cpp (21)

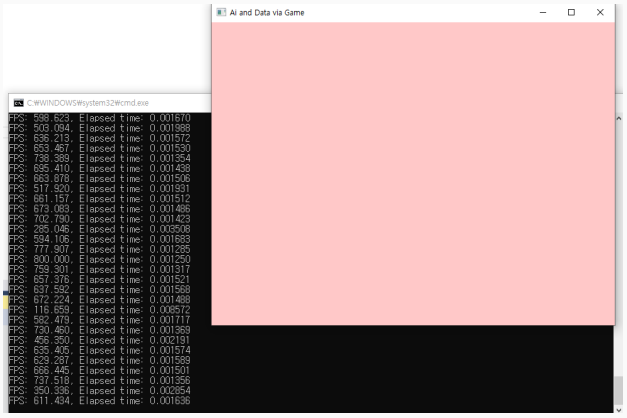
```
  if(!RegisterClassEx(&windowClass))return 0;
  dwExStyle = WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;
  dwStyle = WS_OVERLAPPEDWINDOW;
  AdjustWindowRectEx(&windowRect, dwStyle, FALSE, dwExStyle);
  CKhuGleWin::m_pWinApplication->m_hWnd
    = CreateWindowEx(NULL, "WinClass",
     "Ai and Data via Game",
     dwStyle | WS_CLIPCHILDREN | WS_CLIPSIBLINGS, 0, 0,
     windowRect.right - windowRect.left,
     windowRect.bottom - windowRect.top,
     NULL, NULL, hInstance, NULL);
  if(!CKhuGleWin::m_pWinApplication->m_hWnd) return 0;

  ShowWindow(CKhuGleWin::m_pWinApplication->m_hWnd, SW_SHOW);
  UpdateWindow(CKhuGleWin::m_pWinApplication->m_hWnd);
  QueryPerformanceFrequency(
         (LARGE_INTEGER*)&CKhuGleWin::m_pWinApplication->
         m_TimeCountFreq);
  QueryPerformanceCounter(
         (LARGE_INTEGER*)&CKhuGleWin::m_pWinApplication->
         m_TimeCountStart);
```


KhuGleWin.cpp (22)

```
  while(1) {
   if(PeekMessage(&msg, 0, 0, 0, PM_REMOVE)) {
     if (msg.message == WM_QUIT) break;
     TranslateMessage(&msg);          DispatchMessage(&msg);
   }
   else {
    CKhuGleWin::m_pWinApplication->GetFps();
    CKhuGleWin::m_pWinApplication->Update();
    }
  }
  delete CKhuGleWin::m_pWinApplication;
  _CrtDumpMemoryLeaks();
  UnregisterClass("WinClass", windowClass.hInstance);
  return msg.wParam;
}
void CKhuGleWin::ToggleFpsView() {
  m_bViewFps = !m_bViewFps;
}
```

Main.cpp

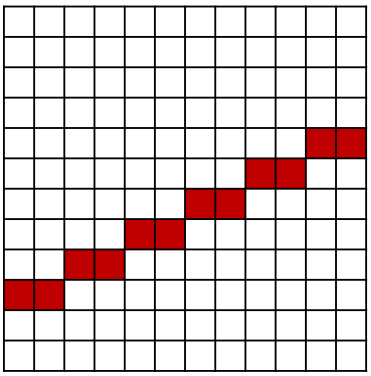
```
#include "KhuGleWin.h"
#include <iostream>

int main() {
  CKhuGleWin *pKhuGleSample = new CKhuGleWin(640, 480);
  KhuGleWinInit(pKhuGleSample);

  return 0;
}
```

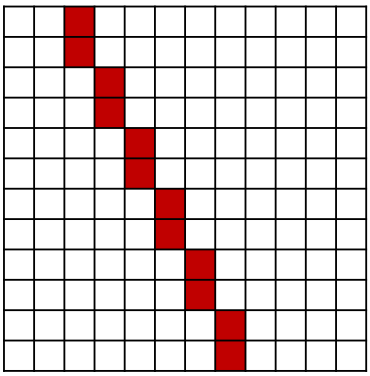


Exercise I (1)

```
class CKhuGleWin {
public:
  int m_nLButtonStatus;
  int m_LButtonStartX, m_LButtonStartY, m_LButtonEndX, m_LButtonEndY;
  CKhuGleWin(int nW, int nH) {
    m_nLButtonStatus = 0;
    …
  }
  void Update() {
    if(m_bMousePressed[0]) {
      if(m_nLButtonStatus == 0){
        m_LButtonStartX = m_MousePosX;        m_LButtonStartY = m_MousePosY;
      }
      m_LButtonEndX = m_MousePosX;      m_LButtonEndY = m_MousePosY;
      m_nLButtonStatus = 1;
    }
    else {
      if(m_nLButtonStatus == 1){
      // Save m_LButtonStartX,m_LButtonStartY,m_LButtonEndX,m_LButtonEndY
        m_nLButtonStatus = 0;
      }
    }
    …
  }
  void OnPaint() { /* Draw line */}
};
```

Exercise I (2)





Advanced Courses

- Timer
  - WM_TIMER
  - SetTimer
    - UINT_PTR SetTimer(HWND hWnd, UINT_PTR nIDEvent, UINT uElapse, TIMERPROC lpTimerFunc);
- Thread
  - std::thread // <thread>
  - std::thread::join() // pauses until the thread finishes
  - Mutex
    - Synchronization primitive that can be used to protect shared data from being simultaneously accessed by multiple threads
    - std::mutex // <mutex>
    - std::mutex::lock() // locks the mutex, blocks if the mutex is not available
    - std::mutex::unlock() // unlocks the mutex