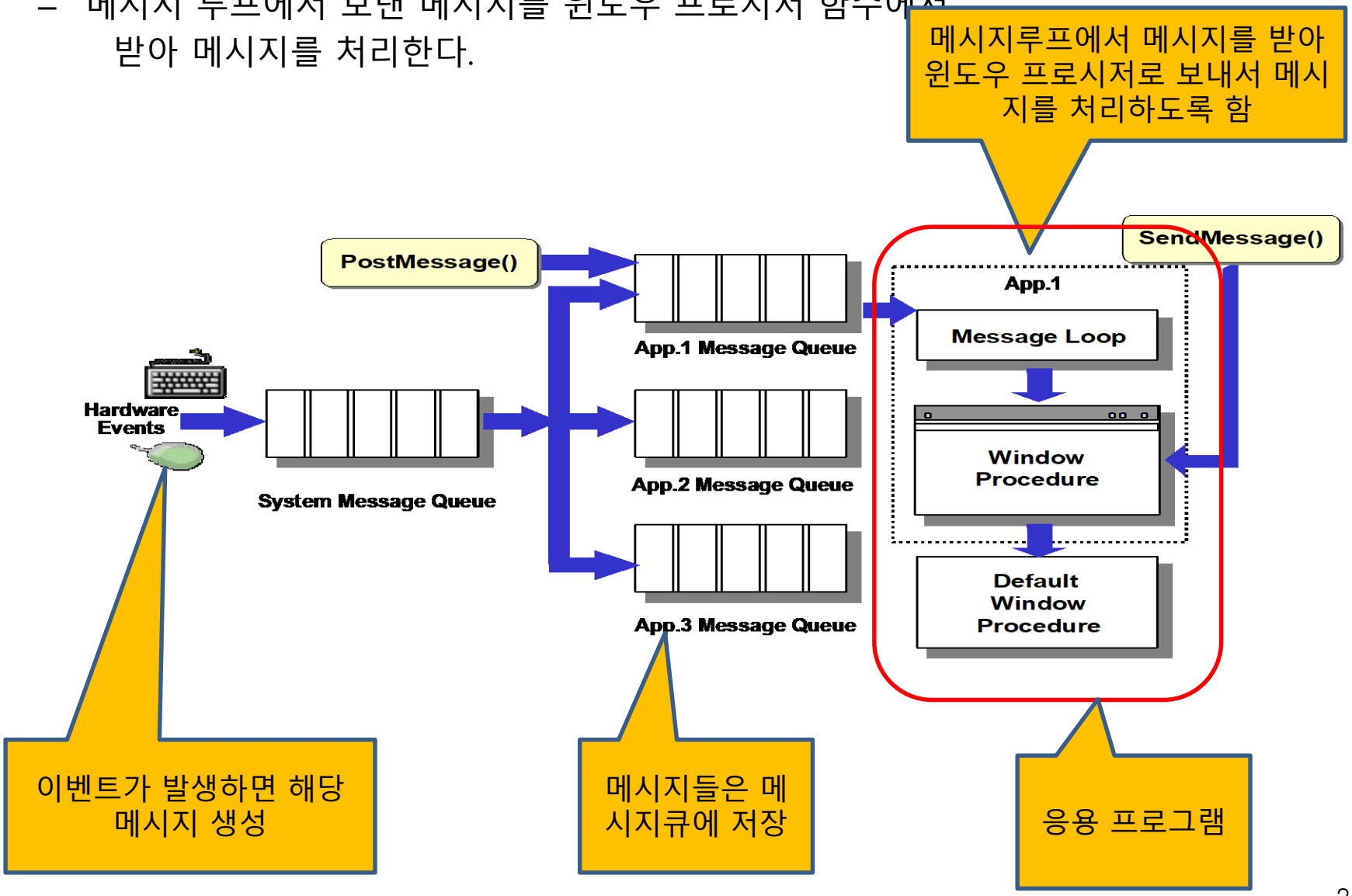
실습 리뷰

2020년 1학기 윈도우 프로그래밍

윈도우 프로그래밍은

- **메인 부분과 메시지 처리 부분으로 나뉜다**
  - 메시지 루프에서 보낸 메시지를 윈도우 프로시저 함수에서 받아 메시지를 처리한다.



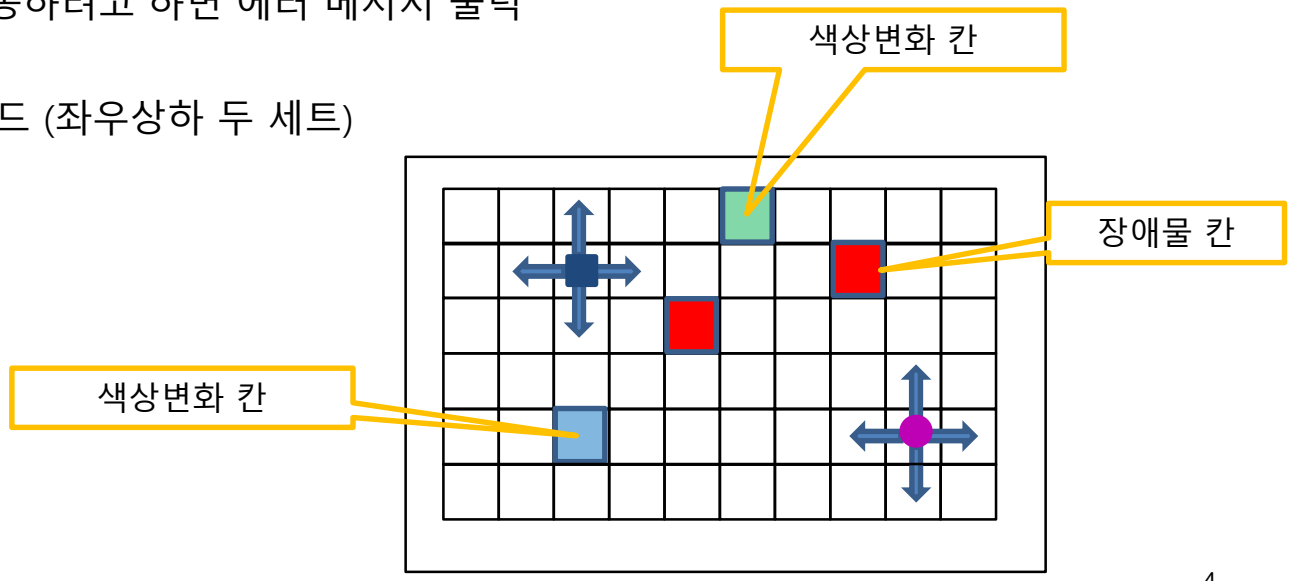
```
#include <windows.h>                          // 윈도우 헤더 파일
#include <tchar.h>

HINSTANCE g_hInst;
LPCTSTR lpszClass = L"Window Class Name";
LPCTSTR lpszWindowName = L"Window Programming Lab";


LRESULT CALLBACK WndProc (HWND hWnd, UINT iMessage,
                          WPARAM wParam, LPARAM lParam);

int WINAPI WinMain (HINSTANCE hInstance,HINSTANCE hPrevInstance,
    LPSTR lpszCmdParam,int nCmdShow)
{
    HWND hWnd;
    MSG Message;
    WNDCLASSEX WndClass;
    g_hInst=hInstance;

    WndClass.cbSize = sizeof(WndClass);
    WndClass.style=CS_HREDRAW | CS_VREDRAW;
    WndClass.lpfnWndProc=(WNDPROC)WndProc;
    WndClass.cbClsExtra=0;
    WndClass.cbWndExtra=0;
    WndClass.hInstance=hInstance;
    WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
    WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
    WndClass.hbrBackground=
            (HBRUSH)GetStockObject(BLACK_BRUSH);
    WndClass.lpszMenuName=NULL;
    WndClass.lpszClassName=lpszClass;
    WndClass.hIconSm = LoadIcon(NULL,IDI_APPLICATION);
    RegisterClassEx (&WndClass);

    hWnd = CreateWindow
                ( lpszClass, lpszWindowName,
                  WS_OVERLAPPEDWINDOW,
                  0, 0, 800, 600,
                  NULL ,(HMENU)NULL,
                  hInstance, NULL);

ShowWindow (hWnd,nCmdShow);
UpdateWindow (hWnd);

while( GetMessage (&Message,0,0,0)) {
        TranslateMessage (&Message);
        DispatchMessage (&Message);
    }
    return Message.wParam;
}

LRESULT CALLBACK WndProc (HWND hWnd,UINT iMessage,WPARAM
    wParam,LPARAM lParam)
{
   PAINTSTRUCT ps;
    HDC  hDC;
    TCHAR temp[] = TEXT("Hello world!");
    int x = 0, y = 0;

    switch(iMessage) {
    case WM_PAINT:
        hDC = BeginPaint(hWnd, &ps);
        TextOut(hDC, x, y, temp, lstrlen(temp));
        EndPaint(hWnd, &ps);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        return 0;
    }
    return (DefWindowProc (hWnd, iMessage, wParam, lParam));
}
```

윈도우 프로시저 함수: 처리될 수 있는 메시지들

- 메시지가 발생할 때마다 호출되어 처리됨
- Loop가 아니라 switch-case 문

실습 2-8 키보드 입력에 따라 도형 이동하기

- **돌 이동하기**
  - 화면에 10x10 칸을 그린다.
  - 칸의 중간 중간에 두개의 다른 색의 칸들이 있다. (최소 5개)
    - 빨간색 칸: 장애물 – 객체들이 통과할 수 없다.
    - 임의의 색 칸 (빨간색 외): 색상 변화 칸 – 돌이 이 칸을 지나면 그 칸의 색으로 바뀐다.
  - 임의의 위치에 두 개의 다른 색을 가진 돌을 그린다.
    - 다른 색으로 표시된 영역 이외의 부분에 돌이 생긴다.
  - 2인 플레이로 각각 다른 키보드 명령어를 이용하여 자신의 돌을 칸에 맞추어 좌우상하 이동한다.
    - 가장자리에 도착하면 그 방향으로 움직일 수 없다.
  - 플레이는 한 명씩 번갈아 하도록 한다.
    - 같은 돌을 두 번 이동하려고 하면 에러 메시지 출력
  - 키보드 명령
    - 2인의 돌 이동 키보드 (좌우상하 두 세트)
    - r/R: 새롭게 시작
    - q/Q: 프로그램 종료



실습 2-8 키보드 입력에 따라 도형 이동하기

- **프로그램 진행**
  - 보드 그리기
  - 키보드 입력받기
    - 턴제로 입력받기
  - 도형 이동하기
    - 좌우상하로 이동하기
    - 가장자리 처리하기
  - 충돌체크
    - 상대방 돌과의 충돌체크
    - 장애물과의 충돌체크
  - 전체 합쳐서 진행하기

실습 2-8 키보드 입력에 따라 도형 이동하기

- **구조체 및 변수 설정**
  - 상수값들

```
#define WINDOW_WIDTH 800
#define WINDOW_HEIGHT 800
#define boardSize 500
#define oneSize 50
#define obstacleNumber 5
```

  - 플레이어 구조체:

```
typedef struct {
        int xPos;
        int yPos;
        COLORREF pColor;
} PLAYER;
```

- 장애물 구조체:

```
typedef struct {
        int xPos;
        int yPos;
        int oStatus; //-- 종류: 벽/색상
        COLORREF oColor;
} OBSTACLE;
```

  - PLAYER playerList[2]; //--- 두 플레이어
  - OBSTACLE obstacleList[obstacleNumber]; //--- 장애물 위치 리스트
  - 그 외 정해야 할 변수들: 누구의 턴인지, 어떤 장애물인지 등을 구별할 값들

실습 2-8 키보드 입력에 따라 도형 이동하기

- **보드 그리기**

```
//=== boardCount: 보드의 칸의 개수
//===  xS: 보드의 x축 시작 값, yS: 보드의 y축 시작값
void DrawBoard(HDC hdc, int boardCount, int xS, int yS)
{
	int i;

	for (i = 0; i <= boardCount; i++) {
		MoveToEx (hdc, i * oneSize+xS, 0 + yS, NULL);
		LineTo (hdc, i * oneSize + xS, boardCount*oneSize + yS);
		MoveToEx (hdc, 0 + xS, i * oneSize + yS, NULL);
		LineTo (hdc, boardCount*oneSize + xS, i * oneSize + yS);
	}
}
```

실습 2-8 키보드 입력에 따라 도형 이동하기

- **장애물 그리기**

```
//=== xS: 보드의 x축 시작 값, yS: 보드의 y축 시작값
void DrawObstacle(HDC hdc, int xS, int yS)
{
	HBRUSH hBrush, oBrush;
	int i;

	for (i = 0; i < obstacleNumber; i++)
	{
		if (obstacleList[i].oStatus == 1)			//-- 벽 장애물
		{
			hBrush = CreateSolidBrush(obstacleList[i].oColor);
			oBrush = (HBRUSH)SelectObject(hdc, hBrush);
			Rectangle(hdc, obstacleList[i].xPos * oneSize + xS, obstacleList[i].yPos * oneSize + yS,
				obstacleList[i].xPos * oneSize + xS + oneSize, obstacleList[i].yPos * oneSize + yS + oneSize);
			SelectObject(hdc, oBrush);
			DeleteObject(hBrush);
		}
		else if (obstacleList[i].oStatus == 2)			//--- 색상 변경 장애물
		{
			hBrush = CreateSolidBrush(obstacleList[i].oColor);
			oBrush = (HBRUSH)SelectObject(hdc, hBrush);
			Rectangle(hdc, obstacleList[i].xPos * oneSize + xS, obstacleList[i].yPos * oneSize + yS,
				obstacleList[i].xPos * oneSize + xS + oneSize, obstacleList[i].yPos * oneSize + yS + oneSize);
			SelectObject(hdc, oBrush);
			DeleteObject(hBrush);
		}
	}

}
```

실습 2-8 키보드 입력에 따라 도형 이동하기

- **장애물 충돌 체크**

```
//=== return 값: 장애물 번호, 만약 장애물 개수보다 큰 숫자가 리턴되면 장애물이 없는 칸
//=== whichObstacle: 장애물 종류, 1-벽 장애물, 2-색상 장애물
//=== xPos, yPos: 플레이어의 x와 y 좌표 인덱스
int CollideCheck(int *whichObstacle, int xPos, int yPos)
{
	int i;

	for (i = 0; i < obstacleNumber; i++) {
		if ((obstacleList[i].xPos == xPos) && (obstacleList[i].yPos == yPos)) {
			if (obstacleList[i].oStatus == 1)
				*whichObstacle = 1;			//-- 벽 장애물
			else if (obstacleList[i].oStatus == 2)
				*whichObstacle = 2;			//--- 색상 장애물
			return i;
		}
	}
	return i;
}
```

실습 2-8 키보드 입력에 따라 도형 이동하기

- **윈도우 프로시저 함수**
  - WM_CREATE 메시지:
    - 보드의 칸 수 결정하기
    - 장애물 만들기: 벽 또는 색상 변동 장애물을 구분하여 만들어 저장
    - 플레이어 만들기: 플레이어의 위치와 색상 만들어 저장

```
case WM_CREATE:
    srand((unsigned int)time(NULL));
    xStart = 50;    yStart = 50;                    //--- 보드의 시작 위치
    bSize = boardSize / oneSize;                    //--- 보드의 칸의 개수 : x축과 y축을 같은 개수로 설정

    //--- 장애물 만들기
    for (i = 0; i < obstacleNumber; i++) {
        temp = rand() % (xCount*yCount);
        obstacleList[i].xPos = temp / 10;           //--- 장애물 x위치
        obstacleList[i].yPos = temp % 10;           //--- 장애물 y 위치
        if (temp % 2==0) {
            obstacleList[i].oStatus = 1;            //--- 벽 장애물
            obstacleList[i].oColor = RGB(100, 100, 100);
      }
      else {
          obstacleList[i].oStatus = 2;              //--- 색상 변동 장애물
          obstacleList[i].oColor = RGB(rand() % 255, rand() % 255, rand() % 255);
      }
    }

    //--- 플레이어 만들기: 하드 코딩하고 장애물과의 위치 비교 안했음
    playerList[0].xPos = 0;
    playerList[0].yPos = 0;
    playerList[0].pColor = RGB(255, 255, 0);

    playerList[1].xPos = 9;
    playerList[1].yPos = 9;
    playerList[1].pColor = RGB(255, 0, 255);

break;
```

실습 2-8 키보드 입력에 따라 도형 이동하기

- WM_PAINT 메시지:
  - 보드 그리기, 장애물 그리기, 플레이어 그리기

```
case WM_PAINT:
	hdc = BeginPaint(hwnd, &ps);

	DrawBoard(hdc, bSize, xStart, yStart);		//--- 보드 그리기
	DrawObstacle(hdc, xStart, yStart);		//--- 장애물 그리기

	//--- 플레이어 그리기
	hBrush[0] = CreateSolidBrush(playerList[0].pColor);
	oBrush = (HBRUSH)SelectObject(hdc, hBrush[0]);
	Ellipse(hdc, playerList[0].xPos * oneSize + xStart, playerList[0].yPos * oneSize + yStart,
			(playerList[0].xPos + 1) * oneSize + xStart, (playerList[0].yPos + 1) * oneSize + yStart);

	hBrush[1] = CreateSolidBrush(playerList[1].pColor);
	oBrush = (HBRUSH)SelectObject(hdc, hBrush[1]);
	Ellipse(hdc, playerList[1].xPos * oneSize + xStart, playerList[1].yPos * oneSize + yStart,
			(playerList[1].xPos + 1) * oneSize + xStart, (playerList[1].yPos + 1) * oneSize + yStart);

	SelectObject(hdc, oBrush);
	DeleteObject(hBrush[0]);
	DeleteObject(hBrush[1]);

	EndPaint(hwnd, &ps);
break;
```

실습 2-8 키보드 입력에 따라 도형 이동하기

– 키보드 입력: 0번 플레이어 (좌우상하 키보드를 이용해서 좌우상하 이동함)

```
case WM_KEYDOWN:
    if (playerTurn == 0) {
        switch (wParam) {
        case VK_LEFT:
            if (playerList[0].xPos > 0) {
                IsObstacle = CollideCheck(&whichObstacle, playerList[0].xPos - 1, playerList[0].yPos);
                if (IsObstacle > 4)                          //--- 장애물이 없어서 움직일 수 있음
                    playerList[0].xPos--;
                else {                                       //--- 장애물이 있는 경우
                    if (whichObstacle == 2) {                //--- 색상 장애물
                        playerList[0].xPos--;
                        playerList[0].pColor = obstacleList[IsObstacle].oColor;
                    }
                }
                playerTurn = 1;
            }
        break;
        case VK_UP:
            if (playerList[0].yPos > 0) {
                IsObstacle = CollideCheck(&whichObstacle, playerList[0].xPos, playerList[0].yPos - 1);
                if (IsObstacle > 4)                          //--- 장애물이 없어서 움직일 수 있음
                    playerList[0].yPos--;
                else {                                       //--- 장애물이 있는 경우
                    if (whichObstacle == 2) {                //--- 색상 장애물
                        playerList[0].yPos--;
                        playerList[0].pColor = obstacleList[IsObstacle].oColor;
                    }
                }
                playerTurn = 1;
            }
        case VK_RIGHT:                          //--- xPos+1 위치를 비교
            ...
        case VK_DOWN:                           //--- yPos+1 위치를 비교
            ...
    }
    InvalidateRect (hwnd, NULL, true);
break;
```

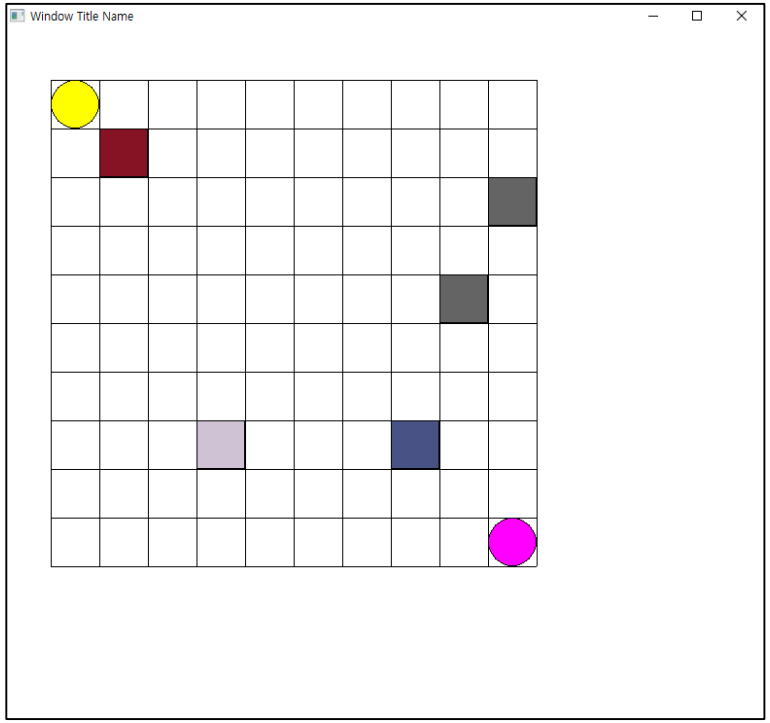
실습 2-8 키보드 입력에 따라 도형 이동하기

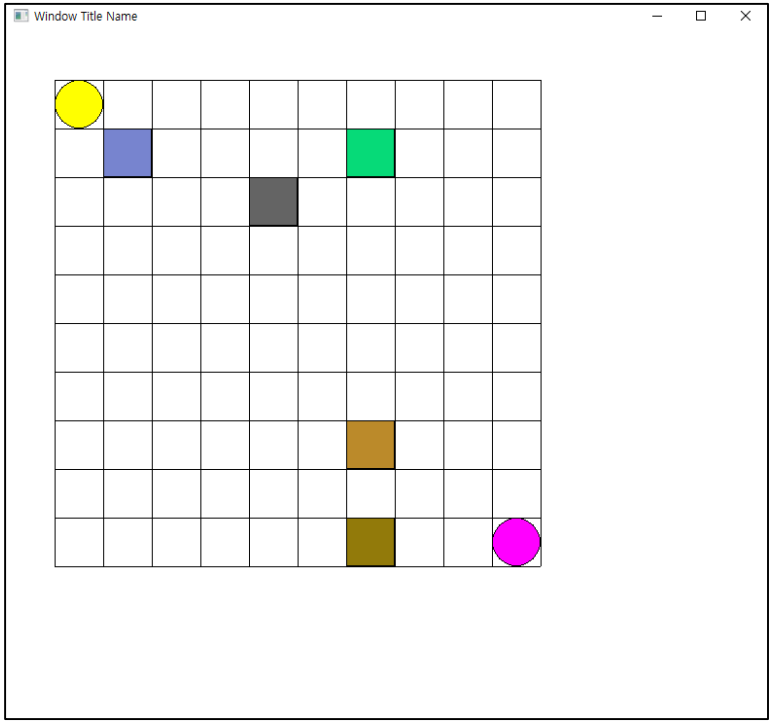
- 키보드 입력: 1번 플레이어 (wasd 키보드를 이용해서 좌우상하 이동함)

```
case WM_CHAR:
    if (playerTurn == 1) {
        switch (wParam) {
        case 'a': ...      break;
        case 'w':...       break;

        case 'd':...       break;

        case 's':...       break;

    }
    InvalidateRect (hwnd, NULL, true);
break;
```

실습 2-8 키보드 입력에 따라 도형 이동하기

- **결과 화면**

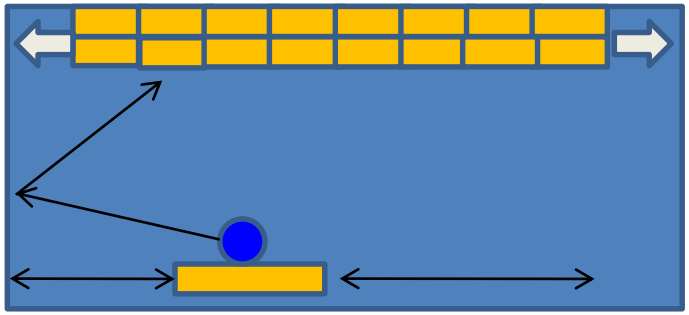


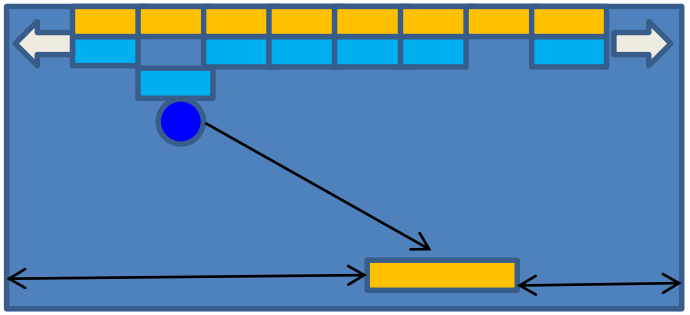


- 앞의 코드에서:
  - 플레이어끼리의 충돌체크 X
  - 장애물의 중복된 자리 체크 X

실습 3-2 벽돌 깨기

- **벽돌 깨기 게임 만들기**
  - **화면의 상단에 2*10 개의 벽돌이** 있다. 벽돌들은 화면에 꽉 차지않고 양쪽에 공간이 있다.
    - 벽돌들은 시간에 따라 좌우로 왔다갔다한다.
  - **화면의 하단에 bar(바)**가 있고 마우스를 이용하여 바를 움직인다.
    - 바닥의 bar(바)를 마우스로 선택하고 드래그하여 이동한다. (좌우로만 이동)
  - 바 위에 공이 있고 시작 명령어를 누르면 공이 튀기기 시작한다. **공이 한 벽돌에 한 번 부딪치면 그 줄의 모든 벽돌의 색이 바뀌며, 부딪친 벽돌은 한 칸 내려온다.**
    - 임의의 색을 설정하여 바꾼다.
  - **공이 튀기면서 벽돌에 두 번 닿으면 벽돌이 없어진다.**
  - **잠시 멈추기 명령어를 누르면 (p 명령어)** 색이 변한 벽돌의 개수와 없어진 개수를 화면에 출력한다.
  - **키보드 명령어**
    - **s/S**: 공 튀기기 시작
    - **p/P:** 움직임이 잠시 멈춤/다시 시작
    - **+/-** 입력: 공의 이동 속도가 늘어난다.
    - **n/N:** 게임 리셋
    - **q/Q:** 프로그램 종료





실습 3-2 벽돌 깨기

- **프로그램 진행**
  - 공 튕기기
    - 공의 X와 y 좌표값이 바뀌어야 한다.
    - 공이 벽이나 벽돌을 만나면 방향 전환
      - 공의 방향 바꾸기: 공이 좌측벽 또는 우측벽에 닿으면 볼의 x축 이동 방향이 반대로 바뀐다.
      - 공의 방향 바꾸기: 공이 위쪽벽 또는 아래쪽벽에 닿으면 볼의 y축 이동 방향이 반대로 바뀐다.
  - 벽돌 이동
    - 벽돌은 좌우로 이동하고 있어야 한다.
      - 특정 범위 안에서 벽돌이 좌 ↔ 우 이동해야 한다.
  - 공과 벽돌 충돌체크
    - 어떤 벽돌인지 벽돌의 위치 (좌표값 또는 번호) 확인
      - 벽돌의 위치: 벽돌을 한 칸 아래로 바꾸기, 또는 벽돌을 삭제하기
      - 공의 방향 바꾸기

실습 3-2 벽돌 깨기

- **각 메시지에서 해야할 일들**
  - WM_CREATE 메시지:
    - 벽돌, 공, 하단의 바 초기화
    - 타이머 설정
  - WM_PAINT 메시지:
    - 벽돌 그리기
    - 하단의 바 그리기
    - 공 그리기
  - WM_LBUTTONDOWN 메시지:
    - 왼쪽 마우스의 위치값 저장
    - 왼쪽 마우스 버튼의 상태 플래그 설정
      - 왼쪽 마우스 버튼이 바를 누르고 있다면 눌려졌다고 설정
  - WM_MOUSEMOVE 메시지:
    - 왼쪽 마우스 버튼이 눌려졌다면
      - 마우스의 위치에 따라 바의 위치값을 변경하여 저장
  - WM_LBUTTONUP 메시지:
    - 왼쪽 마우스 버튼의 상태 플래그 해제
  - WM_TIMER 메시지
    - 공의 좌표값 이동
    - 블록의 좌표값 이동
    - 공과 블록간의 충돌체크
    - WM_PAINT 메시지 발생시키기

실습 3-2 벽돌 깨기

- **구조체 및 변수 설정**

– 블록 구조체 설정

```
typedef struct {
      RECT blockRect;  //-- 블록 위치
      int status;          //-- 블록 상태
      //--- 0: 안 그리기, 1: 기존 색상, 2: 변경색상
} BLOCK;
```

- 공 구조체 설정

```
typedef struct {
      RECT bRect;        //-- 공 위치
      int xStep;         //-- 공 x축 이동값
      int yStep;         //-- 공 y축 이동값
} BALL;
```

– 상수값

```
#define WIDTH 800
#define HEIGHT 500
#define blockXnum 7
#define blockYnum 3
#define blockWidth 75
#define blockHeight 28
#define barXSize 100
#define barYSize 20
```

- 변수들

```
BLOCK block[blockYnum][blockXnum]; //-- 블록들
BALL s_ellipse;                                  //-- 공
Int blocksLeft, blocksRight;      //-- 움직이는 블록
                                  //---의 좌측, 우측 값
int blocksSpeed = 5;               //-- 블록이동속도
Int blocksDirection = 1;           //-- 블록 이동방향
```

실습 3-2 벽돌 깨기

- **WM_CREATE 메시지**
  - 필요한 값 설정하기

```
case WM_CREATE:
    GetClientRect(hwnd, &rt);

    //--- 바 만들기
    barRect.left = 0;barRect.top = 500;barRect.right = barRect.left + barXSize;barRect.bottom = barRect.top + barYSize;

    //--- 공 만들기
    s_ellipse.bRect.left = 200;
    s_ellipse.bRect.top = 100;
    s_ellipse.bRect.right = s_ellipse.bRect.left + 40;
    s_ellipse.bRect.bottom = s_ellipse.bRect.top + 40;
    s_ellipse.xStep = 10;
    s_ellipse.yStep = 10;

    //--- 블록 만들기:블록의 시작 위치 정하기
    blocksLeft = 0;
    blocksRight = blocksLeft + (blockXnum * 80);
    for (mx = 0; mx < blockXnum; mx++)
        for (my = 0; my < blockYnum; my++) {
        block[my][mx].status = 1;
        block[my][mx].blockRect.left = (mx * 80) + blocksLeft;
        block[my][mx].blockRect.top = my * 30;
        block[my][mx].blockRect.right = block[my][mx].blockRect.left + blockWidth;
        block[my][mx].blockRect.bottom = my * 30 + blockHeight;
        }
    for (mx = 0; mx < blockXnum; mx++)          //--- 세번째 줄은 안 그리기
        block[2][mx].status = 0;

    Selection = false;
    SetTimer(hwnd, 1, 50, NULL);
    Break;
```

실습 3-2 벽돌 깨기

- **WM_PAINT 메시지**
  - 블록, 공, 바 그리기

```
case WM_PAINT:
        hdc = BeginPaint(hwnd, &ps);

        //--- 더블 버퍼링
        //Rectangle(mdc, 0, 0, rt.right, rt.bottom);
        //DrawBlock(mdc, blockXnum, blockYnum);
        //Rectangle(mdc, barRect.left, barRect.top, barRect.right, barRect.bottom);
        //
        //Ellipse(mdc, s_ellipse.bRect.left, s_ellipse.bRect.top, s_ellipse.bRect.right, s_ellipse.bRect.bottom);

        //BitBlt(hdc, 0, 0, rt.right, rt.bottom, mdc, 0, 0, SRCCOPY);

        //--- 더블 버퍼링 x
        Rectangle(hdc, 0, 0, rt.right, rt.bottom);
        DrawBlock(hdc, blockXnum, blockYnum);          //--- 블록 그리기

        Rectangle(hdc, barRect.left, barRect.top, barRect.right, barRect.bottom);    //-- 바 그리기

        Ellipse(hdc, s_ellipse.bRect.left, s_ellipse.bRect.top, s_ellipse.bRect.right, s_ellipse.bRect.bottom);        //-- 공 그리기
        EndPaint(hwnd, &ps);
break;
```

```
static HDC mdc;              // 더블버퍼링을 위해 추가될 부분
static HBITMAP hBitmap;  // 코드부분은create 메시지에 넣어도 된다.

hdc = GetDC(hwnd);
mdc = CreateCompatibleDC(hdc);
hBitmap = CreateCompatibleBitmap(hdc, rt.right, rt.bottom);
SelectObject(mdc, (HBITMAP)hBitmap);
```

실습 3-2 벽돌 깨기

- **마우스 이벤트 처리하기**
  - 마우스가 바를 클릭하고 움직이면 바를 움직인다.

```
case WM_LBUTTONDOWN:
        mx = LOWORD(lParam);
        my = HIWORD(lParam);
        pt.x = mx;              pt.y = my;
        if (PtInRect(&barRect, pt))          //--- 마우스가 바를 클릭하고 있다면
                Selection = true;
        break;

case WM_MOUSEMOVE:
        hdc = GetDC(hwnd);
        if (Selection) {
                mx = LOWORD(lParam);
                my = HIWORD(lParam);
                barRect.left = mx;
                barRect.right = mx + barXSize;
                InvalidateRect(hwnd, NULL, true);

        }
        ReleaseDC(hwnd, hdc);
break;

case WM_LBUTTONUP:
        Selection = false;
break;
```

실습 3-2 벽돌 깨기

- **타이머 메시지**
  - 공의 위치를 바꾼다.
    - 공과 가장자리와의 충돌체크: 공의 방향을 바꾼다.

```
case WM_TIMER:
    //--- 공 움직임(좌표변화)
    s_ellipse.bRect.left += s_ellipse.xStep;
    s_ellipse.bRect.right += s_ellipse.xStep;
    s_ellipse.bRect.top += s_ellipse.yStep;
    s_ellipse.bRect.bottom += s_ellipse.yStep;

    //--- 공과 윈도우 가장자리와의 충돌체크: 방향을 바꾼다.
    If (s_ellipse.bRect.right > WIDTH)
            s_ellipse.xStep *= -1;
    if (s_ellipse.bRect.bottom > HEIGHT)
            s_ellipse.yStep *= -1;

    if (s_ellipse.bRect.left < 0)
            s_ellipse.xStep *= -1;
    if (s_ellipse.bRect.top < 0)
            s_ellipse.yStep *= -1;
```

실습 3-2 벽돌 깨기

- 블록의 위치를 바꾼다.
  - 블록의 속도에 따라 블록을 좌우로 이동한다.

```
//---블록 움직임(좌표변화)
if (blocksDirection > 0 ) {
	blocksLeft += blocksSpeed;
	blocksRight = blocksLeft + (blockXnum * 80);

	if (blocksRight > WIDTH) {		//--- 블록의 오른쪽 끝이 오른쪽 가장자리에 닿으면 방향을 바꾼다.
		blocksDirection *= -1;
		blocksLeft = WIDTH – (blockXnum * 80);
		blocksRight = blocksLeft + (blockXnum * 80);
	}
}
else {
	blocksLeft -= blocksSpeed;
	blocksRight = blocksLeft + (blockXnum * 80);

	if (blocksLeft < 0) {  //--- 블록의 왼쪽 끝이 왼쪽 가장자리에 닿으면 방향을 바꾼다.
		blocksDirection *= -1;
		blocksLeft = 0;
		blocksRight = blocksLeft + (blockXnum * 80);
	}
}
for (my = 0; my < blockYnum; my++)    //--- 블록의 위치 바꾸기
	for (mx = 0; mx < blockXnum; mx++) {
		block[my][mx].blockRect.left = (mx * 80) + blocksLeft;
		block[my][mx].blockRect.top = my * 30;
		block[my][mx].blockRect.right = block[my][mx].blockRect.left + blockWidth;
		block[my][mx].blockRect.bottom = my * 30 + blockHeight;
	}
```

실습 3-2 벽돌 깨기

– 공과 블록간의 충돌체크

```
//---공과 블록간의 충돌체크
if (s_ellipse.yStep < 0) {
    for (i = 0; i < blockYnum; i++) {
        for (j = 0; j < blockXnum; j++) {
            if (block[i][j].status > 0) {
                if (IsCollision (s_ellipse.bRect.left + 20, s_ellipse.bRect.top,
                                 block[i][j].blockRect.left, block[i][j].blockRect.top)) {
                    if (block[i][j].status == 1) {
                        block[i+1][j].status = 2;
                        block[i][j].status = 0;
                    }
                    else if (block[i][j].status == 2)
                        block[i][j].status = 0;
                    s_ellipse.yStep *= -1;                    //---공의 진행방향 전환
                }                                             //--- if (IsCollision()) 끝
            }                                                 //---I f(block[i][j]) 끝
        }
        if (s_ellipse.yStep > 0)
            break;
    }
}   //--- if (yStep<0) 끝

InvalidateRect (hwnd, NULL, true);
break;
```

실습 3-2 벽돌 깨기

- **블록 그리기 함수**
  - WM_PAINT 메시지에서 호출하는 블록 그리기 함수

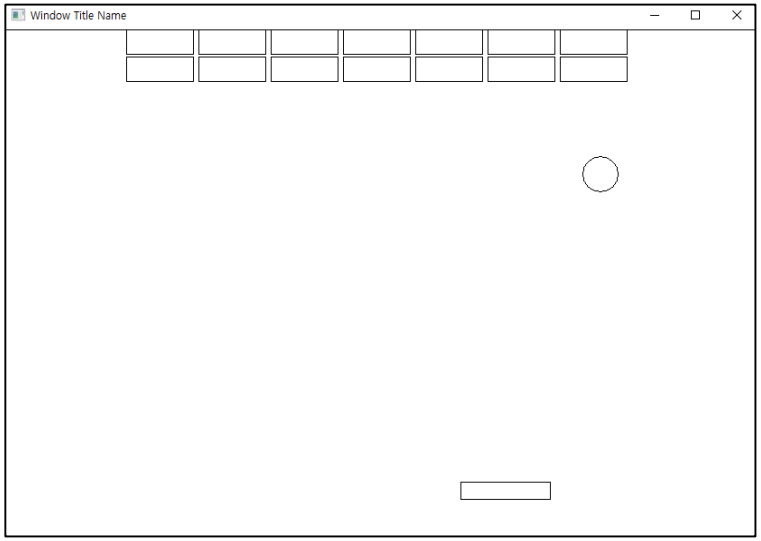
```
//=== xCount, yCount: 블록의 개수
void DrawBlock (HDC hdc, int xCount, int yCount)
{
	int i, j;
	HBRUSH hBrush, oldBrush;		//--- 블록면을 색칠하기 위한 브러쉬 생성


	for (i = 0; i < yCount; i++)
		for (j = 0; j < xCount; j++) {

			if (block[i][j].status == 1) {				//--- 블록 상태가 1: 기존 색상으로 그리기
				Rectangle(hdc, block[i][j].blockRect.left, block[i][j].blockRect.top,
					block[i][j].blockRect.right, block[i][j].blockRect.bottom);
			}
			else if (block[i][j].status == 2) {		//--- 블록 상태가 2 (한칸 내려온 경우): 변경된 색상으로 그리기
				hBrush = CreateSolidBrush(RGB(255, 0, 0));
				oldBrush = (HBRUSH)SelectObject(hdc, hBrush);

				Rectangle(hdc, block[i][j].blockRect.left, block[i][j].blockRect.top,
					block[i][j].blockRect.right, block[i][j].blockRect.bottom);

				SelectObject(hdc, oldBrush);
				DeleteObject(hBrush);
			}
		}

}
```

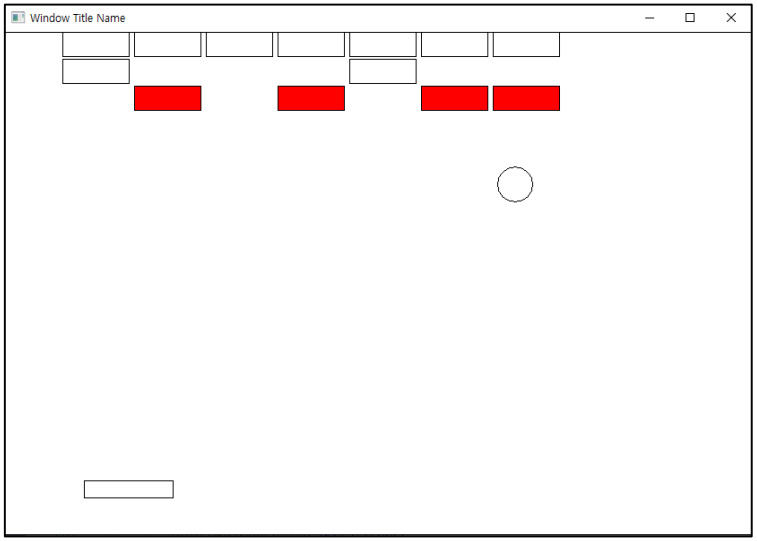
실습 3-2 벽돌 깨기

- **충돌 체크 함수**
  - 점과 영역간의 충돌체크

```
//=== x, y: 점으로 사용할 원의 좌표값
//=== xBlock, yBlock: 영역으로 사용할 좌상 좌표값
bool IsCollision(int x, int y, int xBlock, int yBlock)
{
	POINT pt;
	RECT rect;

	pt.x = x;
	pt.y = y;
	rect.left = xBlock;
	rect.top = yBlock;
	rect.right = rect.left + blockWidth;
	rect.bottom = rect.top + blockHeight;

	if (PtInRect (&rect, pt))
		return true;
	else
		return false;
}
```

- **결과 화면**





실습 3-4 자동차 움직이기

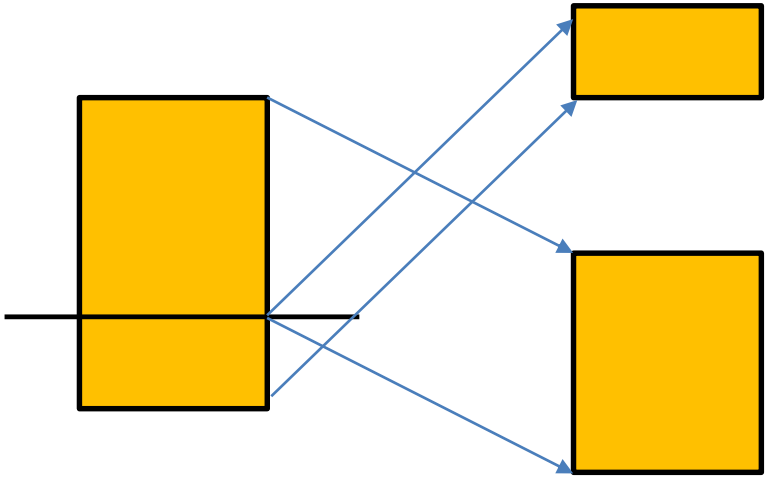
- **자동차 구조체**
  - 자동차의 위치: RECT 활용
  - 자동차의 상태: 달리는 상태/멈춘 상태를 표시할 bool 값
- **자동차를 저장할 배열**
  - 사거리에서 멈춘 순서대로 달리기 위해서 배열에 저장하고, 멈춘 자동차는 교차로 영역에 자동차가 없으면 배열에 저장된 순서대로 꺼내져 다시 달리기 시작한다.
- **자동차 일부분씩 그리기**
  - 자동차가 가장자리로 빠져나가고 반대편 가장자리로 일부분씩 나타나게 하려면
    - 한 개의 자동차를 2개로 나눠서 그린다.
      - 상하로 이동하던 자동차인 경우:

      Rectangle (hdc, left, top, right, bottom);

      ➔

      Rectangle (hdc, left, top+step, right, bottom);

      Rectangle (hdc, left, 0, right, step);
    - 타이머 메시지에서
      - step += 10; //--- 한번에 이동하는 크기에 따라



- **타이머 메시지에서는**
  - 자동차마다 타이머를 설정한다.
  - 각각의 자동차 타이머에서 자동차의 위치를 바꾼다.
    - 가장자리와 교차로의 좌표값을 비교하며 자동차를 나눠서 출력하거나 타이머를 멈춘다.

이번 시간에는

- **이번 시간에는**
  - 실습을 리뷰
    - 키보드 다루기, 마우스 다루기
    - 타이머 다루기
  - 이번시간에는 실습은 없습니다~~
- **애니메이션**
  - 비트맵 사용한 애니메이션
  - 더블 버퍼링
    - 숙제에서 화면이 깜빡거리는 경우에 적용하기
- **다음시간에 만나요. 안녕~**