애벌레 게임 프로그램 분석

본 보고서를 최종보고서로 제출합니다.

이　　름: 이지호

학　　번: 20163290

학　　과: 응용소프트웨어

1. 프로그램 분석

1.1 전역변수 분석

`#define ITEMNUM 10` : 게임내의 생성될 애벌레의 먹이 개수를 정하는 변수이다.
10개의 먹이를 생성한다.

`int board[22][22];` : 게임판의 크기를 정하는 2차원 배열이다.
22 x 22칸의 크기를 가지는 판이다.

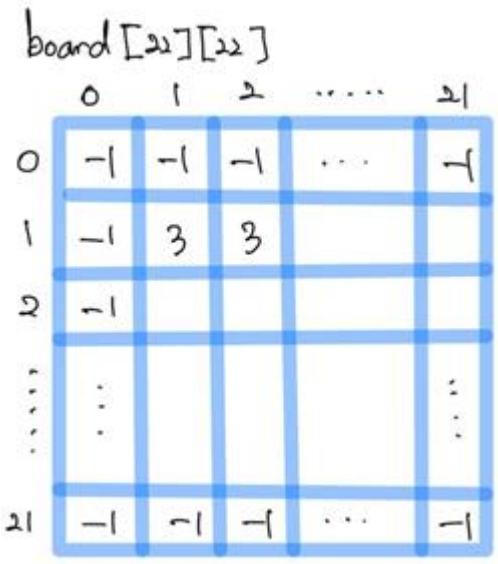
`int wormX[30], wormY[30];` : 애벌레의 길이는 최대 30이 된다.

`int xDirect, yDirect, len;`
xDirect, yDirect : UP, DOWN, LEFT, WRITE키가 눌렸을 때
x와 y의 방향에 대한 값을 저장하는 변수이다.
Len : 애벌레의 길이를 저장하는 변수이다.

1.2 함수 분석

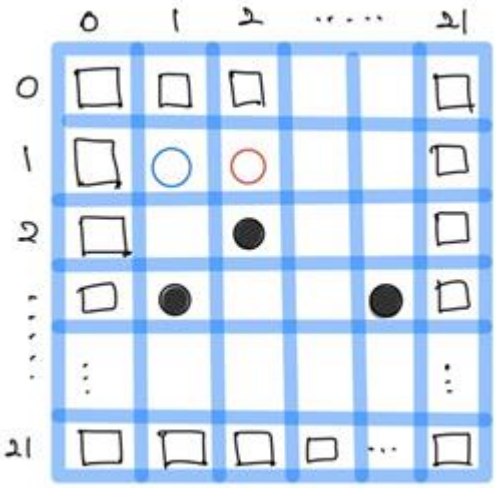
프로그램이 시작되고 GameInit()함수가 실행된다.

GameInit()
board 이차원 배열, 애벌레의 좌표를 저장하는 배열의 기본값을 초기화하는 함수이다.
Len = 2 초기 길이를 2로 지정
xDirect와 yDirect는 1과 0으로 저장하고 애벌레가 게임시작과 동시에 오른쪽으로 이동하게끔 해주었다.



이때 GameInit()함수에 들어있는 ItemGenerator()함수를 실행해줘야한다.

ItemGenrator()
ITEMNUM의 값인 10을 받아서 반복문을 10회 실행해준다.
이때 X와 Y의 값을 랜덤으로 생성해서 대입해준다.
X와 Y값은 최대 게임판의 공간 중 벽을 제외하고 20X20의 공간에 랜덤의 위치에 생성해준다.
i에 i-1의 값이 들어가는 경우 그 경우는 건너뛴다.
검은색 점이 랜덤으로 생성된 먹이이다.



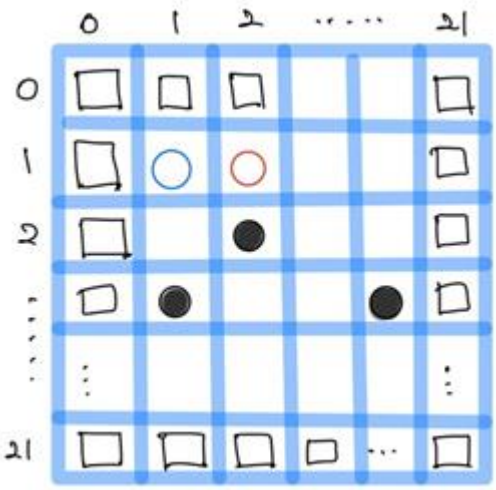
위의 함수를 마치고 GameInit함수의 len값과 xDirect값과 yDirect값을 저장해주고 함수를 벗어난다.

그러고 난 후 SetTimer함수를 실행한다.

SetTimer()함수는 일정시간 마다 WM_TIMER메시지를 발생시킨다.
SetTimer(hwnd, 1, 100, NULL) : 0.1초마다 메시지를 발생시키는 것이다.

이후 WM_PAINT로 가서
hdc로 페인트의 객체를 만들어 주고
DrawGameBoard()함수를 실행해준다.

DrawGameBoard()
생성된 페인트객체를 매개변수로 받아와 사용한다.
22 x 22 회를 반복하는 이중for문을 만들어주고
GameInit함수에서 외벽은 -1로 지정하고 애벌레의 먹이는 2로 지정한 것을 이용하여
Swich - case문을 이용하여 -1로 값이 들어간 배열의 공간에는 빈칸으로 구성된
가로 20 세로 20의 네모칸을 만들어준다.
먹이의 값으로 저장한 2의 값이 저장된 배열의 공간에는 검정색으로 가득차있는 지름이
20인 원을 그려준다.
그리고 페인트 객체의 색을 흰색으로 바꿔주면서 게임판에 대한 기본적인 설정을 한다.
애벌레의 좌표값을 받아서 머리는 빨간색 꼬리는 파란색 페인트를 선택하여 속이 빈 원을
그려주고 len의 값이 길어지는(애벌레가 먹이를 먹은경우) 상황에 맞게
for문을 설정해둔다.

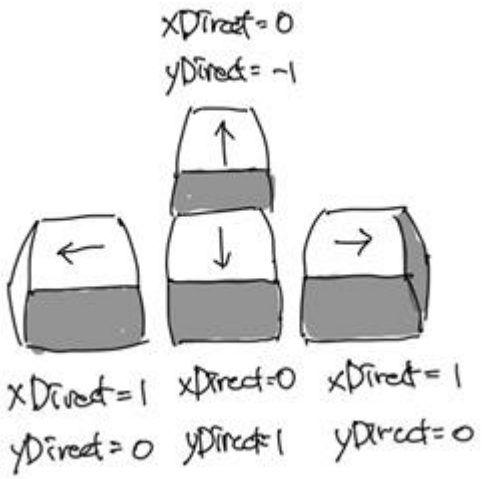


위 함수를 빠져나온뒤 페인트에 대한 것을 종료해주고 키값의 변화에 따른 처리를 해준다.

키를 누를때마다 값이 들어오는데 이 값이 들어올 때 **WM_KEYDOWN**을 실행해준다
이때 **DirectControl()**함수를 실행해준다.

DirectControl()
VK_~~~ 를 이용하여 들어오는 키 값을 처리해준다.
왼쪽키를 누른경우 **xDirect**가 **1**이면 케이스문을 빠져나온다.(이미 누르고 있다는 뜻)
xDirect가 **1**이 아닐 때 보드의 좌표값이 **-1**이 아니라면
(**180**도 회전에 대한경우를 막는것이다.)
xDirect의 값을 **-1**로 변경하고 **yDirect**값을 **0**으로 저장한다.
다른 키를 눌렀을 때 비슷한 방식으로 작동한다.

자세한 경우에 대한 설명으로는 **1-3**에서 정리하겠다.



위에서 **SetTimer**는 **WM_TIMER**의 메시지를 발생시킨다고 했다.
그래서 **SetTimer**함수에 의해 **WM_TIMER**는 프로그램 실행과 동시에 **0.1**초마다 계속해서 작동한다.
이때 **WM_TIMER**안의 **MovingWorm()**함수가 실행된다.
즉 **MovingWorm()**함수가 프로그램 시작과 동시에 **0.1**초마다 계속해서 실행되고 있는것이다.

MovingWorm()
애벌레의 머리가 가지는 위치에 대한 값을 tmpx와 tmpy에 저장한다.
보드 위의 애벌레의 머리값이 벽이 아니거나 머리를 가리키는 경우
변화되는 머리의 값을 머리의 값을 가지는 배열의 첫번째 공간에 저장해준다.
만약 머리의 부분이 먹이가 위치한 부분과 겹치게 되면 길이를 1늘이고 먹이가 있던 공간에 먹이가 없어짐을 표시한다.
즉 먹이를 먹은경우 길이를 늘여주고 애벌레 머리의 좌표값을 조정하여 길이가 늘어남을 나타내 주는 함수이다.

1.3 전체 코드 분석

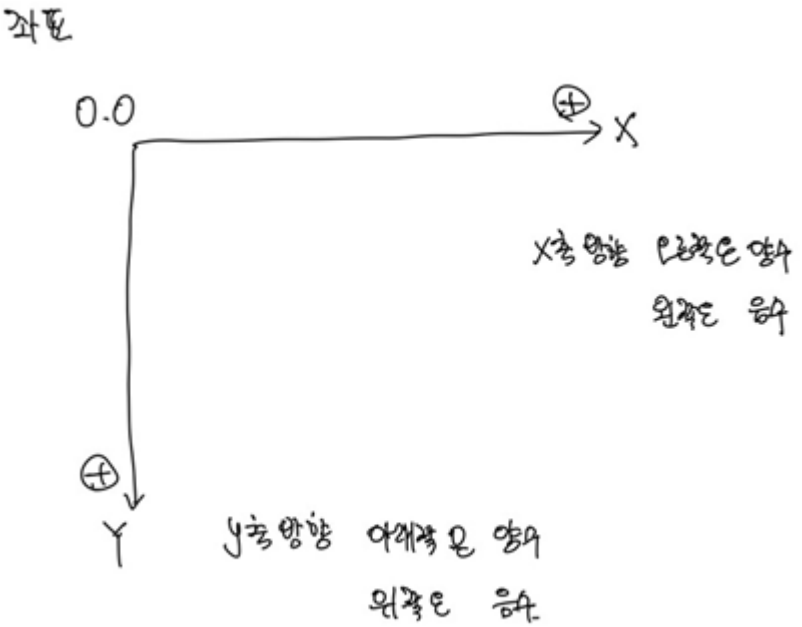
```
LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam);
void End_Game(HWND hwnd, int score, int* status);
void Start_Setting(HWND, int(*)[2], int(*)[2], int*, int*, int*, int*, int*,
int*, int*);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpszCmdLine, int nCmdShow)
{
	HWND hwnd;
	MSG msg;
	WNDCLASS WndClass;
	WndClass.style = CS_HREDRAW | CS_VREDRAW;	//출력 형태
	WndClass.lpfnWndProc = WndProc;	//메시지 처리를 위한 함수 이름 전달
	WndClass.cbClsExtra = 0;	//여분의 클래스메모리 (사용 X)
	WndClass.cbWndExtra = 0;	//여분의 윈도우메모리 (사용 X)
	WndClass.hInstance = hInstance;	//메모리상에서 프로그램의 위치 전달
	WndClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);	//아이콘모양
	WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);	//커서모양
	WndClass.hbrBackground = (HBRUSH)CreateSolidBrush(RGB(255, 255, 255));
	//배경색
	WndClass.lpszMenuName = NULL;	//메뉴
	WndClass.lpszClassName = _T("Windows Class Name");	//윈도우 클래스이름
	RegisterClass(&WndClass);	//커널에 등록
	hwnd = CreateWindow(
		_T("Windows Class Name"),	//윈도우 클래스이름
		_T("애벌레"),	//윈도우 타이틀이름
		WS_OVERLAPPED,	//윈도우스타일 (타이틀바의 최소화, 최대화, 닫기)
OR기호로 여러개 지정가능
		CW_USEDEFAULT,	//x좌표
		CW_USEDEFAULT,	//y좌표
		800,	//윈도우 가로크기
```

```
            500,          //윈도우 세로크기  CW_USEDEFAULT는 커널에 의해 기본값을
지정
            NULL,  // 부모 윈도우 핸들
            NULL,  // 메뉴 핸들
            hInstance,          //응용 프로그램 인스턴스
            NULL          //생성 윈도우 정보
      );
      ShowWindow(hwnd, nCmdShow);
//(나타낼 윈도우 핸들값, 윈도우를 화면에 나타내는 방법으로 상수값 제공
ex)SW_MAXIMIZE)
      UpdateWindow(hwnd); //윈도우 화면에 기본 출력하기
      while (GetMessage(&msg, NULL, 0, 0))    //메시지큐에서 MSG구조체변수에 저장
      {
            TranslateMessage(&msg);          //변환?
            DispatchMessage(&msg);          //메시지를 처리하는 함수에 메시지를
보냄
      }

      return (int)msg.wParam;
}


#define ITEMNUM 10
//애벌레 먹이의 개수

int board[22][22];
//게임판의 크기
int wormX[30], wormY[30];
//애벌레의 최대크기
int xDirect, yDirect, len;
//x축방향의 이동값, y축 방향의 이동값, 애벌레의 길이


void ItemGenerator()
//애벌레의 먹이를 만드는 함수이다.
{
      int i, x, y;

//정수형 변수 i, x, y 생성

      for (i = 0; i < ITEMNUM; i++)
            //전역변수로 입력한 ITEMNUM을 이용하여
            //10회를 반복하는 반복문을 실행해준다.
      {
            x = rand() % 20 + 1;
            y = rand() % 20 + 1;
```

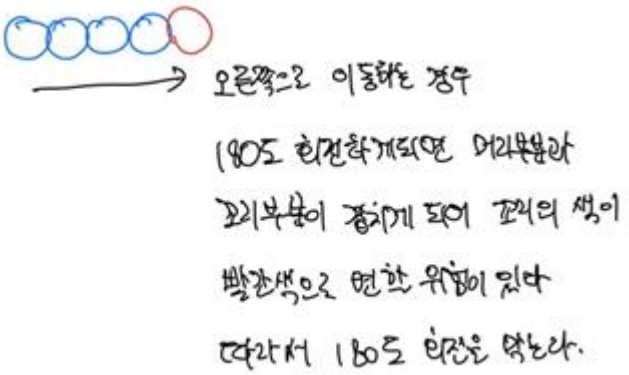
```
			//벽을 제외하고 남는 20개의 공간중에서
			//먹이의 좌표를 랜덤으로 생성한다
			//+1이 들어가는 이유는 먹의의 좌표에
			//0값이 들어가면 벽과 겹치기 때문이다.
			if (board[y][x] == 0)
				board[y][x] = 2;
			//이렇게 생긴 먹의의 좌표값에 2를 넣어준다.
			else
			{
				i = i - 1;
				//먹이가 있는 위치에 먹이를 또 생성한경우
				//i값을 -1해주고 해당 반복문을 건너뛴다.
				continue;
			}
		}
		return;
}


void DrawGameBoard(HDC hdc)
//실질적으로 게임판에 대한 그림을 그리는 함수이다.
{
		int i, x, y;
		for (y = 0; y < 22; y++) {
			for (x = 0; x < 22; x++)
		//22*22의 공간에 대한 필요한 부분에
		//그림을 그리기 위해 이중반복문을 실행해준다.
			{
				switch (board[y][x])
				//board[y][x]에 들어간 값이
				{
				case -1:
				//-1이라면
					Rectangle(hdc, x * 20, y * 20, (x + 1) * 20, (y + 1)
* 20);
				//속이 빈 가로 20 세로 20의 정사각형을 만들어준다.
					break;
				case 2:
				//2라면
					SelectObject(hdc, GetStockObject(BLACK_BRUSH));
					//검정색을 들고
					Ellipse(hdc, x * 20, y * 20, (x + 1) * 20, (y + 1) *
20);
					//지름이 20인 원을 그려준다.
					SelectObject(hdc, GetStockObject(WHITE_BRUSH));
					//펜을 원래대로 돌려준다.
					break;
				}
```

```
			}
		}

		SelectObject(hdc, CreatePen(PS_SOLID, 2, RGB(255, 0, 0)));
		//빨간색을 들고
		Ellipse(hdc, wormX[0] * 20, wormY[0] * 20, (wormX[0] + 1) * 20, (wormY[0]
+ 1) * 20);
		//머리부분에 해당하는곳 wormX[0]와 wormY[0]에
		//지름이 20인 원을 빨간색으로 그린다.
		SelectObject(hdc, CreatePen(PS_SOLID, 2, RGB(0, 0, 255)));
		//파란색을 들고

		for (i = 1; i < len; i++)
			Ellipse(hdc, wormX[i] * 20, wormY[i] * 20, (wormX[i] + 1) * 20,
(wormY[i] + 1) * 20);
		//꼬리부분에 해당하는곳 wormX[i]와 wormY[i]에
		//지름이 20인 원을 파란색으로 그린다.
}


void GameInit()
//board배열에 대한 초기화를 하는 함수이다.
{
		int i;
		for (i = 0; i < 22; i++)
		{
			board[i][0] = -1;
			board[i][21] = -1;
			board[0][i] = -1;
			board[21][i] = -1;

			//벽에 해당하는 부분에 대한 초기화를 해주는
			//반복문을 실행한다. -1로 초기화
		}
		wormX[0] = 2; wormY[0] = 1;
		wormX[1] = 1; wormY[1] = 1;
		//애벌레의 머리와 꼬리를 정하는 좌표를 초기화해주고
		board[wormY[0]][wormX[0]] = 3;
		board[wormY[1]][wormX[1]] = 3;
		//board상에 머리와 꼬리의 부분에 대한 값을
		//3으로 초기화 해준다.

		ItemGenerator();
		//먹이의 값을 설정하는 함수를 실행해준다.

		len = 2;
		//초기의 길이를 2로 잡고
		xDirect = 1; yDirect = 0;
		//프로그램이 실행하면 무조건 오른쪽으로 가게끔 설정해준다.
```

```
}

void DirectControl(int DirectKey)
```




```
//방향키로 누르는 값을 매개변수로 받아 처리해주는 함수이다.
{
	switch (DirectKey) {
	case VK_LEFT:
		//왼쪽키를 입력했을때
		if (xDirect == 1)
			//오른쪽 키값이 입력되어있는경우
	//이 경우를 설정하는 이유 : 애벌레는 180도 회전을 바로하지 못한다.
```




```
			break;
		//아래의 case문을 실행할수 없으므로 빠져나간다.
		if (board[wormY[0]][wormX[0] - 1] != -1)
			//board위의 애벌레의 값이 벽과 같지 않다면
		{
			xDirect = -1;
			yDirect = 0;
```

```
                //x축으로 -1의 방향으로 이동
                //y축으로의 이동이 없음을 표시한다.
        }
        break;
case VK_RIGHT:
        //오른쪽 키를 입력했을때
        if (xDirect == -1)
                break;
        //왼쪽키가 입력된 상태라면
        //아래의 case문을 실행할수 없으므로 빠져나온다.
        if (board[wormY[0]][wormX[0] + 1] != -1)
                //오른쪽으로 이동하는 애벌레의 머리의 좌표값이
                //벽이 아니라면
        {
                xDirect = 1;
                yDirect = 0;
                //x축으로 +1의 방향으로이동
                //y축으로의 이동이 없음을 알려준다.
        }
        break;
case VK_UP:
        //위의 방향키를 입력했을때
        if (yDirect == 1)
                break;
        //아래의 방향으로 이동하고 있다면
        //아래의 case문을 실행할 수 없으므로 빠져나온다.
        if (board[wormY[0] - 1][wormX[0]] != -1)
                //위로 이동하는 애벌레의 머리가 벽이 아니라면
        {
                xDirect = 0;
                yDirect = -1;
                //x축 방향으로의 이동은 없고
                //y축 방향으로 -1의 방향으로 이동하고있다.
        }
        break;
case VK_DOWN:
        //아래의 방향키를 눌렀을 경우
        if (yDirect == -1)
                break;
        //위쪽의 방향으로 이동하고 있다면
        //아래의 case문을 실행할 수 없으므로 case문을 빠져나온다.
        if (board[wormY[0] + 1][wormX[0]] != -1)
                //아래로 이동하는 애벌레의 머리가 벽이 아니라면
        {
                xDirect = 0;
                yDirect = 1;
                //x축 방향으로 이동은 없고
```

```
                        //y축 방향으로 1의 방향으로 이동한다.
                }
                break;
        }
}

//방향키로 누르는 값을 매개변수로 받아 처리해주는 함수이다.
{
        switch (DirectKey) {
        case VK_LEFT:
                //왼쪽키를 입력했을때
                if (xDirect == 1)
                        //오른쪽 키값이 입력되어있는경우
        //이 경우를 설정하는 이유 : 애벌레는 180도 회전을 바로하지 못한다.
                        break;
                //아래의 case문을 실행할수 없으므로 빠져나간다.
                if (board[wormY[0]][wormX[0] - 1] != -1)
                        //board위의 애벌레의 값이 벽과 같지 않다면
                {
                        xDirect = -1;
                        yDirect = 0;
                        //x축으로 -1의 방향으로 이동
                        //y축으로의 이동이 없음을 표시한다.
                }
                break;
        case VK_RIGHT:
                //오른쪽 키를 입력했을때
                if (xDirect == -1)
                        break;
                //왼쪽키가 입력된 상태라면
                //아래의 case문을 실행할수 없으므로 빠져나온다.
                if (board[wormY[0]][wormX[0] + 1] != -1)
                        //오른쪽으로 이동하는 애벌레의 머리의 좌표값이
                        //벽이 아니라면
                {
                        xDirect = 1;
                        yDirect = 0;
                        //x축으로 +1의 방향으로이동
                        //y축으로의 이동이 없음을 알려준다.
                }
                break;
        case VK_UP:
                //위의 방향키를 입력했을때
                if (yDirect == 1)
                        break;
                //아래의 방향으로 이동하고 있다면
                //아래의 case문을 실행할 수 없으므로 빠져나온다.
```

```
			if (board[wormY[0] - 1][wormX[0]] != -1)
				//위로 이동하는 애벌레의 머리가 벽이 아니라면
			{
				xDirect = 0;
				yDirect = -1;
				//x축 방향으로의 이동은 없고
				//y축 방향으로 -1의 방향으로 이동하고있다.
			}
			break;
		case VK_DOWN:
			//아래의 방향키를 눌렀을 경우
			if (yDirect == -1)
				break;
			//위쪽의 방향으로 이동하고 있다면
			//아래의 case문을 실행할 수 없으므로 case문을 빠져나온다.
			if (board[wormY[0] + 1][wormX[0]] != -1)
				//아래로 이동하는 애벌레의 머리가 벽이 아니라면
			{
				xDirect = 0;
				yDirect = 1;
				//x축 방향으로 이동은 없고
				//y축 방향으로 1의 방향으로 이동한다.
			}
			break;
		}
}


void MovingWorm()
{
		int tmpx, tmpy, i;

		tmpx = wormX[0];
		tmpy = wormY[0];
		//애벌레의 머리를 임시로 tmpx와 tmpy에 저장해둔다.
		//나중에 변화할수 있기때문에 임시로 저장.
		//값에 직접 저장해두면 먼저 들어있던 값이 없어진다.
		wormX[0] += xDirect; wormY[0] += yDirect;
		//키 다운으로 입력된 값만큼 애벌레의 머리값에 더해준다.

		if (board[wormY[0]][wormX[0]] == -1 || board[wormY[0]][wormX[0]] == 3)
		//board의 애벌레의 좌표값이 벽이 아니거나 3으로 저장된 경우
		{
			wormX[0] = tmpx;
			wormY[0] = tmpy;
			//머리의 값이 임시로 저장되어있던 tmpx와 tmpy의 값을 원래대로 돌려준다.
		}
		else
```

```
		//위의 경우가 아닌경우
	{
		if (board[wormY[0]][wormX[0]] == 2)
			//애벌레의 머리가 먹이의 부분과 겹치게 된다면
		{
			len = len + 1;
			board[wormY[0]][wormX[0]] = 0;
			//길이를 1 늘여주고 그 위치의 먹이를 없애준다.
		}
		else
			board[wormY[len - 1]][wormX[len - 1]] = 0;
		//먹이의 부분을 지나지 않는 경우
		//애벌레가 지나간 부분의 값들을 0으로 초기화 해준다.

		for (i = len - 1; i > 1; i--)
		{
			wormX[i] = wormX[i - 1];
			wormY[i] = wormY[i - 1];
			//애벌레의 이동을 나타내준다.
		}
		wormX[1] = tmpx;
		wormY[1] = tmpy;
		//원래 들어있던 값을 worm[1]에 넣어주고
		board[wormY[0]][wormX[0]] = 3;
		//애벌레의 머리에 대한 좌표를 3으로 저장해준다.
		//그래야지 머리는 빨간색 꼬리는 파란색의
		//길이가 늘어난 애벌레를 표현할 수 있다.
	}
}


LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
	HDC hdc;
	PAINTSTRUCT ps;
	switch (iMsg) {
	case WM_CREATE:
		GameInit(); //보드의 값의 초기화를 해주는 함수.
		SetTimer(hwnd, 1, 100, NULL);
		//timer를 설정하는 함수.
		//WM_TIMER의 메세지를 0.1초 간격으로 보낸다.
		return 0;
	case WM_PAINT:
		hdc = BeginPaint(hwnd, &ps);
		//hdc를 통해 paict객체를 생성해주고
		DrawGameBoard(hdc);
		//생성된 객체를 DrawGameBoard함수에 매개변수로 전달
		//게임보드에 대한 시각화를 하는 함수이다.
```

```
            EndPaint(hwnd, &ps);
            //페인트 객체를 생성하면 항상 돌려줘야 하기 때문에
            //EndPaint를 실행한다.
            break;
        case WM_KEYDOWN:
            DirectControl((int)wParam);
            //방향키가 눌릴때 마다 값을 저장하여 반환해주는 함수이다.
            break;
        case WM_TIMER:
            MovingWorm();
            //SetTimer에 대해 작동되는 함수이며 애벌레의 움직임을
            //표현해주는 함수이다.
            InvalidateRgn(hwnd, NULL, TRUE);
            //그리기 영역을 유효화하는 함수이다.
            break;
        case WM_DESTROY:
            KillTimer(hwnd, 1);
            //타이머를 꺼주는 함수이다.
            PostQuitMessage(0);
            return 0;
        }
        return(DefWindowProc(hwnd, iMsg, wParam, lParam));
}
```

2. 결론 및 소감

게임으로만 플레이 해보던 애벌레 게임의 구조에 대해서 알 수 있어서 좋았습니다.

Key down에 대한 것을 int형 매개변수를 이용하여 컨트롤 할 수 있다는 것이 새로웠습니다.

API를 여러가지 함수를 이용해서 구현하는 방법에 대해 배울 수 있었고,

2차원 배열을 통해서 보드를 구현할 수 있고 움직이는 좌표에 대해 x좌표와 y좌표를

이용하여 2차원 배열상에서 구현하는 방법에 대해서도 알게 되었습니다.

타이머를 계속 호출하면서 변화되는 값이 반응할 수 있도록 하는 방법도 배웠습니다.

앞으로의 API개발에 큰 도움이 될 것 같은 좋은 과제인 것 같아서 좋았고,

제대로 분석할 수 있다는 것에 보람도 느꼈습니다.

감사합니다.