- Contents -

Ⅰ . Abstract

IBM pc는 세계 pc 시장의 대부분을 차지하고 있다. 이러한 IBM pc의 가장 대표적인 cpu는 Intel의 80x86 계열의 cpu이다. assembly 프로그램을 통해서 이러한 cpu의 작동 원리와 구조에 대한 이해도를 높일 수 있다. 특히, 게임이란 분야는 한 컴퓨터가 가지고 있는 모든 기능을 수행해 볼 수 있는 좋은 도구이므로 본 텀 프로젝트의 주제로 삼았다. 따라서 지뢰찾기 프로그램을 구현함으로서 assembly를 익히고, assembly의 특성과 IBM PC의 구조를 이해하는데 목적이 있다.

우리가 흔히 볼 수 있는 windows 운영체제에 기본적으로 포함되어 있는 지뢰 찾기 게임을 MS-DOS 환경에서 assembly를 이용해서 만들어본다.

- 팀 구성원

정보통신대학 컴퓨터학과 2000160157 유 미 선
정보통신대학 컴퓨터학과 2000160184 이 진 무

팀원별 개발환경

유미선 [OS: Windows2000, CPU: Intel Celeron(400MHz), Memory: 192MB]
이진무 [OS: Windows2000, CPU: Intel Pentium3(400MHz), Memory: 320MB]

사용컴파일러: Macro Assembly Ver 6.11
모니터 : 평면 모니터가 아닌 일반 모니터

II. 본 프로젝트에 관련한 기본 지식

1. IBM PC

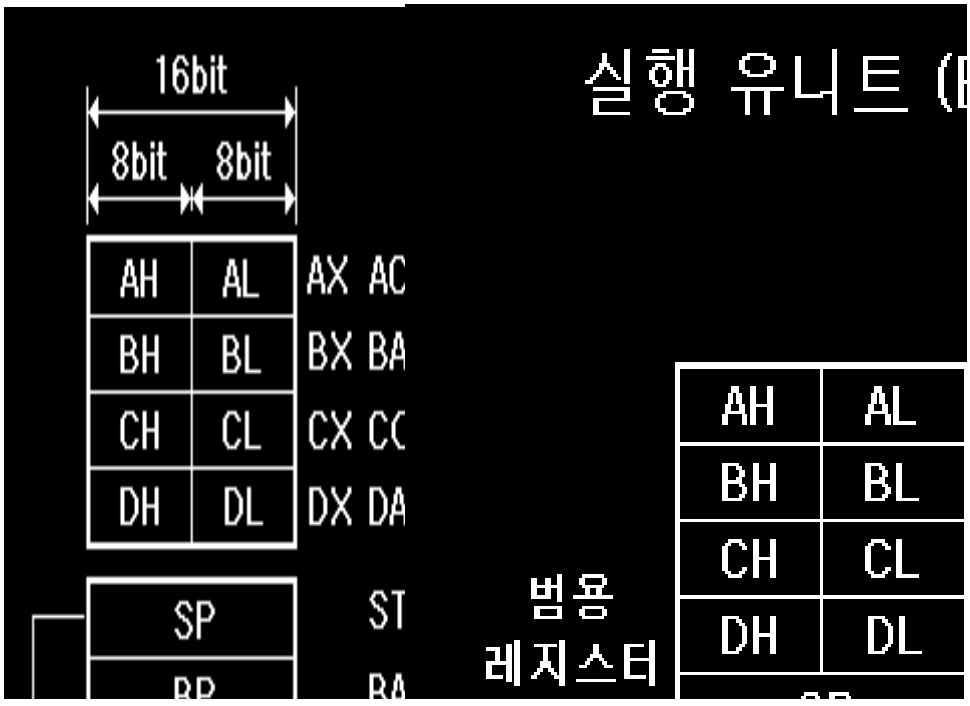
1.1 8086의 기본 구성

1.11 CPU의 기본 구성

8086의 CPU는 Execution Unit(EU) 과 Bus Interface Unit(BIU)으로 나눌 수 있다.
EU 는 산술연산과 논리 연산을 수행하는 ALU와 그것을 적절히 조정하는 Control Unit (CU) 가 있다. 그리고 내부적인 저장 공간인 레지스터가 존재한다.
BIU 는 EU 에서 처리할 명령어나 데이터를 EU 로 보내주며, bus를 통해 외부 입력 장치와 EU를 연결한다.

1.1.2 8086 의 내부 메모리에 대하여

8086에는 많은 레지스터가 있고, 각각16비트의 크기를 가지고 있다. 이들 레지스터는 크게 나누면 연산에 사용되는 범용 레지스터와 번지를 지정하기 위해 사용되는 세그먼트 레지스터(Segment register)와 인스트럭션 포인터(Instruction pointer), CPU의 상태 등을 지시, 제어하는 플레그 레지스터(flag register)등으로 구성된다. 범용 레지스터는 주로 연산에 사용되는 레지스터(AX, BX, CX, DX)와 주로 번지의 계산에 사용되는 레지스터(SP, BP, SI, DI) 등으로 나눌 수가 있다. 또한 AX, BX,CX, DX의 4개의 레지스터는, 상위 8비트와 하위 8비트로 나누어 각각 8비트 레지스터 AH, AL 등으로도 사용할 수가 있다.



8086 register CPU의 구조

1.1.3 레지스터의 종류와 각각의 특징

A. 데이타 레지스터

데이터 레지스터는 각종 데이터 처리를 대상으로 한다. 8086/8088 계열의 범용 레지스터는 4가지가 있다. 총 16비트로 이루어져있는데, 16비트 레지스터 및 8비트 레지스터 일부를 프로그래머가 명령 중에서 자유롭게 지정을 할 수 있는 범용 레지스터이다. 범용 레지스터로서의 역할 이외에도 아래에서 설명하는 특정한 역할도 가지고있다. 이 레지스터들은 사용자가 데이터를 조작하는데 자주 필요한 것들이다. 레지스터들의 기본적인 의미는 프로그래머가 임의로 무시할 수 있다.
- AX : 어큐레이터 레지스터라고도 하며 연산 레지스터로서 하며 연산의 결과나 중간 값 등을 저장하는데 쓰인다. AH는 AX의 상위 8비트를 AL은 AX의 하위 8비트를 가리킨다. 연산기능이 다른 것보다 조금 많은 레지스터이다.
- BX : 베이스 레지스터라고도 하며 베이스 어드레스 지정에 쓰인다. 간접 어드레스 지정시에 어드레스 레지스터, 트랜슬레이터 명령에 있어서 변환 테이블, 베이스 레지스터로서 사용한다.
- CX : 카운터 레지스터라고도 하며 반복 실행문의 반복 횟수를 지정할 때 쓰인다.
- DX : 데이터 레지스터라고도 하며 어큐레이터의 보조로 활용되거나 간접 어드레스에 의한 입출력 명령시 어드레스 지정에 사용된다. 곱셈, 나눗셈 작업을 할 때 사용된다.

B. 포인터 레지스터와 인덱스 레지스터

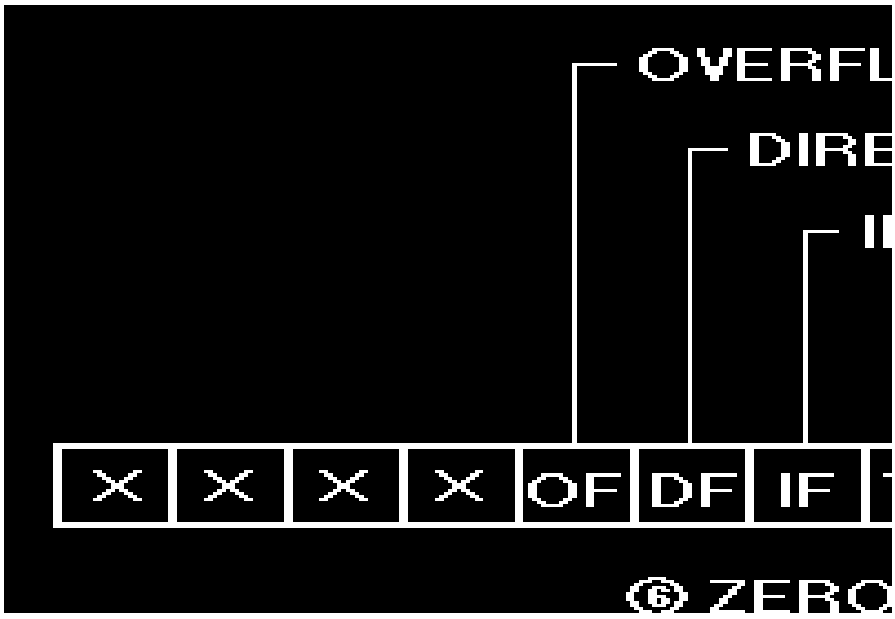
- BP : 베이스 포인터 레지스터이다. 기본적으로 스택 영역내 어드레스를 지시하지만 스택 세그먼트 SS 영역내에 배치한 데이타에 대한 베이스 어드레스의 위치 지정에 사용된다.
- SI : 소스 인덱스 레지스터이다, 오퍼랜드 소스, 소스 데이터를 어드레스 지정에 사용한다
- DI : 데스티네이션 인덱스 레지스터이다. 오퍼랜드의 처리 대상 또는 데스티네이션 데이터를 나타내는데 사용된다.
- SP : 스택포인터 레지스터라고 불린다. 메모리에서 스택의 위치를 펴시한다. 스택 맨 위의 주소를 담고 있다.

C. 세그먼트 레지스터

- CS : 코드 세그먼트 레지스터이다. 실행형 프로그램 코드가 들어있는 메모리 주소를 지정한다.
- DS : 데이타 세그먼트 레지스터이다. 프로그램 수행후 데이타가 들어있는 메모리 주소를 지정한다.
- SS : 스택 세그먼트 레지스터이다. 스택의 시작 주소를 담고 있다.
- ES : 확장 세그먼트 레지스터이다. 데이타 세그먼트의 확장으로 쓸수 있다. 확장 세그먼트의 시작 주소를 담고 있다.

D. 플래그 레지스터

- TF : 추적 플래그이다. 명령어가 1개씩 실행되기에 프로그램의 에러 추적등에 사용할수 있다. 프로세서를 프로 그램 디버거용의 싱글 스테프 모드 아래 놓는다.
- IF : 인터럽트 인에이블 플래그이다. 외부 인터럽트를 Inable 또는 Disable한다. 외부 인터럽트의 허락을 결정한다.
- DF : 디렉션 플래그이다. 스트링처리에서 방향을 결정하기에 자동 증가나 감소가 이루어진다. 스트링 명령에서의 실행방향 자동 인크리먼트 또는 디크리먼트 조작을 제어 한다.
- CF : 캐리플래그이다. 2바이트 이상의 가감산에 이용하고 최상위비트에서 자리올림 또는 그자리에서의 자리올림을 표시한다.
- PF : 패리티 플래그이다. 데이터 전송이나 연산시에 짝수 패리트를 검사해 준다.
- AF : 보조 캐리 플래그이다. 4비트에서 5비트로 자리올림을 표시해 준다.
- ZF : 제로 플래그이다. 연산에서 자주 쓰인다. 연산 결과가 0이면 1이라고 표시해 준다.
- SF : 부호 플래그이다. 부호를 표시해 준다. 오퍼레이션의 결과,최상위 비트가 1일때 세트된다. 0=정,1=부의 부호를 표시한다.
- OF : 오버플로우 플래그이다. 산술연산 결과의 오버플로우 상태를 표시한다.
- IP : 명령 포인터 레지스터이다. 현재 실행중인 명령의 위치를 가리킨다



2.1 세그먼트의 개념

2.1.1 IBM 의 번지 지정법

8086은 1M바이트까지의 메모리를 취급할 수가 있다. 이 1M 바이트의 메모리에는 0번지부터 순서대로 번지가 붙여져 있으므로 번지 값은 16진수로 0H~0FFFFFH까지의 값이 된다. 1M 바이트의 번지 데이터를 관리 하기위해서는 20 비트가 필요하게 된다. 사실 8086으로부터는 20개의 번지 신호선이 나온다. 그러나 8086의 레지스터는 모두 16비트의 크기 밖에는 없으므로 이것으로는 20비트의 번지를 나타낼 수 없다. 그 때문에 8086애서는 2개의 레

지스터를 조합 시켜 20비트의 번지를 나타내는 방법을 취하고 있다.
16비트 레지스터로 나타낼 수 있는 값은 16진수로 0~0FFFFH(4자리)까지의 값이다. 그래서 20비트 = 16진수 5자리의 값을 나타내기 위해서, 2개의 레지스터를 4비트 = 16진수 1자리분 만큼만 비켜서 덧셈하는 방법을 취한다.

```
예)   2000     H
     + 3456    H
     ------------
       23456    H
```

이때 상위 16진수로 4자리의 값을 지정하는 목적으로 사용되는 것이 세그먼트 레지스터이다. 하위 4자리는 IP(명령을 읽어 낼 때)나 BX, SI(데이터를 읽고 쓸 때)라는 식의 레지스터로 지정된다.

이 때 세그먼트 레지스터는 2000H라는 값을 가지는데, 실제로는 물리번지로는 20000H를 가리키고 있다. 이 번지를 세그먼트 베이스라고 부르고, 다른 레지스터에 나타낸 3456H라는 값을 오프셋번지라고 말한다.

오프셋이란 어떤 기준번지(여기서는 세그먼트 베이스)로 부터의 변위를 나타낸다.오프셋번지는 BX나 SI, IP등의 레지스터에 의해 나타내는 것 외에 직접 수치로 지정될 수도 있다.

그러나 한 개의 번지를 나타내는데 매번 두개의 레지스터를 지정해야 하는 것에도 문제는 있다. 그래서 8086에서는 세그먼트 레지스터를 지정하지 않아도 오프셋 번지만을 지정하면, 자동적으로 세그먼트 레지스터의 값을 더하여 번지가 계산되도록 만들어져 있다.

따라서 세그먼트 레지스터의 값을 한번 설정해 놓으면, 세그먼트 베이스로부터 64KB이내의 번지는 오프셋 번지를 지정하는 것만으로 표시할 수 있다.

이와 같이 세그먼트 레지스터에 의해 표현되는 세그먼트 베이스의 값을 기준으로 한, 64KB의 범위를 세그먼트라고 한다. 세그먼트 베이스의 값이 하위 1바이트가 0이면 어떤 곳에도 설정할 수 있다. 또한 필요하다면 세그먼트 레지스터의 값을 변경하는 것도 가능하므로, 세그먼트 레지스터와 오프셋 번지를 변환함으로써 1MB의 메모리를 자유롭게 취급할 수가 있다.

2. 어셈블리언어

- 이번 프로젝트에서는 어셈블리 언어를 사용해서 프로그래밍을 하므로 어셈블리 언어에 대한 이해가 필요하다. 어셈블리 언어에 대한 개괄적인 정보를 수집해 보았다.

2.1 어셈블리 언어란....

◆ 어셈블리 언어(Assembly Language)

① 기계어와 1대 1로 대응한 명령을 기술하는 언어이다.
② CPU 개발사에 따른 독자적인 기능을 갖는다.
③ 동일한 CPU에 대해서도 다른 어셈블리가 있을 수도 있다.

--> 기계어의 수치에 따른 표현 대신에 인간에게 알기 쉬운 표기법을 사용하여 덧셈 명령은 ADD, 전송 명령은 MOV등을 표현하도록 한 것이다.
--> 하나하나의 명령의 표현이 기계어 코드와 1대 1로 대응이 된다.

◆ 매크로 어셈블리 언어(Macro Assembly Language)
① 기계어로 번역되지 않는 의사명령을 사용한다.
② 매크로 기능을 가진다.

◆ 어셈블러(assembler)
어셈블리 언어로 쓰여진 소스(source)프로그램을 번역하여 기계어 프로그램을 작성해주는 프로그램이다. 대표적으로 MASM, CASM, NASM 등이 있다.

◆ 어셈블(assemble)
어셈블리 언어로 쓰여진 프로그램을 어셈블러를 사용하여 기계어로 번역하는 작업

2.2 Assembly의 필요성

◆ 보다 깊게 컴퓨터를 이해하고, 보다 잘 컴퓨터를 익히 사용하려고 할 때에 부딪치게 되는 것이 어셈블리 언어이다.

◆ 어셈블리 언어의 지식은 고속연산 루틴을 작성하기 위해서만 필요한 것이 아니라 마이크로 컴퓨터의 시스템을 잘 알기 위하여 필요한 것이다.

◆ 메모리상의 데이터나 I/O 기기를 직접 액세스 하는 등, 고급언어에서는 할수 없는 조작을 할수 있다라는 것이다.

◆ MS-DOS에서는 고급 언어와 어셈블러와의 링크가 가능하므로, 고급 언어와 어셈블러의 링크가 가능하므로, 고급 언어와 어셈블러 각각의 장점을 살린 프로그램의 작성이 가능하다.
◆ CPU가 이해할 수 있는 것은 수치로 기술된 명령밖에 이해할 수가 없다.

2.3 MASM에 의한 Assembly 개발법

■ 매크로 어셈블러를 사용하여 기계어 프로그램을 작성
① 에디터(editor)를 사용하여, 어셈블리 언어로 기술된 소스 프로그램을 작성한다.
② 어셈블러를 사용하여, 그것을 오브젝트 파일(object file)이라는 중간 파일로 변환한다.
③ 링커(linker)를 사용하여 그것을 실행 파일(execute file)이라는 실행가능한 파일로 만든다.

■ MASM에 의한 어셈블러 개발법
- 아스키 문자열로써 저장하는 형태의 에디터라면 무엇이든 사용할 수 있다.

- MASM은 모듈별 개발이 용이하므로 축적된 소프트웨어를 유효하게 활용할 수 있다.

■ 링커(Linker)의 역할
- 몇가지의 화일을 합쳐서 하나의 프로그램으로 만드는 것이 링커의 역할이고 링커에 입력이 되는 화일이 중간 화일이 오브젝트 화일이다.
- MS-DOS상에서는 어셈블러에 한하지 않고, 여러 종류의 컴파일러도 동일한 형식의 오브젝트 파일을 작성하고 있다. 따라서 서로 다른 언어로 작성된 파일을 결합하는 것도 가능하다.

■ 라이브러리(Library)의 사용법
- 모듈별로 개발된 오브젝트 화일들을 합쳐서 하나의 라이브러리 화일 이라는 것을 생성해 둔다. 링크시에 라이브러리를 지정하는 것만으로도 그 중에서 필요한 오브젝트 화일만을 자동으로 꺼내어져 결합하는 것이 가능하다.
- 라이브러리 매니저(Library Manager) LIB을 사용함으로써, 다수의 오브젝트 파일을 하나의 라이브러리로 정리해 둘 수 있다.
- 오브젝트 파일의 추가, 발췌, 삭제가 가능하다.
- FORTRAN, C등에서 컴파일러 유틸리티 속에 포함되어 있는 라이브러리는 입출력 함수 등을 합친 것이다

2.4 분할 어셈블의 개념

MASM에서는 분할 어셈블을 하는 것이 가능하다. 분할 어셈블은 프로그래밍의 효율을 크게 향상시킨다.또한 모듈별 프로그래밍으로도 연결된다. 프로그램의 모듈 구조는 고급언어의 세계에서는 상식이라고 할 수 있는 개념이지만, MASM에 있어서도 역시 중요한 개념이다.

2.4.1 분할 어셈블이란?

분할 어셈블이란, 소스프로그램을 여러 개의 소스 파일로 나누어, 각각을 독립적으로 어셈블하는 것이다.어셈블의 결과 출력되는 오브젝트 파일도 여러 개로 되지만, 그것들을 서로 연결 시킴에 따라 하나의 실행 파일이 완성된다. 언뜻 보면 수고가 드는 방법이라고 생각되지만, 분할 어셈블이 가능함에 의해서 생기는 이점은 그 수고를 훨씬 웃돈다.
일반적으로는, 프로시저(procedure)부분을 꺼내어 독립 소스 파일로 만듦으로써 소스 프로그램을 분할한다. 그러면 하나는 프로시저의 호출은 있어도 프로시저의 본체가 없는 소스 파일(주된 모듈), 또 하나는 프로시저의 본체는 있어도 프로시저의 호출이 없는 소스 파일로 된다.
각각의 소스 파일은, 그 자신 어셈블 가능이어야만 한다. 즉, 세그먼트의 정의(SEGMENT ~ ENDS)나 세그먼트 레지스터의 할당(ASSUME) 등, 어셈블에 필요한 지시를 의사명령에 의해 올바르게 지정해 줄 필요가 있다. 단, 주된 모듈 이외의 소스 파일에서는 END의사 명령으로 실행 개시번지를 지정하지 않음에 주의한다.
이제, 하나의 소스 파일을 어셈블하면, 하나의 오브젝트 파일이 생성된다. 그리고 그것을 서

로 연결시켜 하나의 실행 파일을 작성하는 것이 링크의 조작이다. 링크 조작의 역할은, 다른 오브젝트 파일에 분할되어 있는, 프로시저의 본체와 프로시저의 호출을 잘 연결시킴에 있다.

2.4.2 분할 어셈블의 이점

(1) 어셈블 시간의 단축
분할 어셈블의 첫째 이점은, 프로그램의 개발시간을 단축할 수 있는 점이다. 프로그램을 작성하기 위해서는, 어셈블 및 링크를 몇 번이나 해야만 한다. 그 이유는 프로그램을 실행해 보아 생각한 대로 작동해 주지 않는 부분을 수정하거나, 마음에 들지 않는 부분을 개량하거나 하는 일을 몇번이나 반복하기 때문이다. 소스 파일을 분할함에 때라, 이 사이클에 필요한 시간을 단축할 수 있다. 여기서 사이클이란 편집, 어셈블 및 링크 작업을 하는 하나의 되풀이 되는 과정을 일컫는다.
프로그램이 커져도 변경을 가하거나 추가해 가는 부분은 전체로 보면 정말 일부에 지나지 않는다. 분할한 소스 파일의 하나 이외에 변경이 가해져 있지 않으면, 그 파일만을 어셈블하면 다른 소스 파일을 어셈블할 필요는 없다. 소스 파일의 하나에 모든 프로그램이 포함되는 경우 보다도, 어셈블에 걸리는 시간은 짧아 진다.

(2) 프로그램의 부품화와 공유
프로그램을 몇 개나 작성하는 동안에, 똑 같은 프로시저가 종종 필요한 경우가 있다. 이것과 똑 같은 예는 상수 정의를 생각해 보자. 상수 정의의 경우는 정의 부분을 해더 파일로서 분할하고 INCLUDE의사 명령에 의해서 소스 파일에 포함시킴으로써, 여러 개의 프로그램에서 똑 같은 정의를 공유할 수 있다. 또 매크로 명령에 관해서도 똑 같은 방법이 유효하다. 그런데 프로시저의 경우에는, 이 방법이 그다지 유효하지 않다. 상수나 매크로는 정의 부분이 반드시 필요한데 대해, 프로시저는 그 본체가 똑 같은 소스 파일 내에 있을 필요가 없기 때문이다.
소스 파일을 분할하고 각각을 독립적으로 어셈블함에 따라, 효율 좋게 프로시저를 공유할 수 있다. 이 방법은 프로그램의 부품화라고 한다. 프로시저라는 부품을 이요하여 프로그램을 조립해 간다는 사고방식이다.
분할 어셈블이 가능하면 여러 가지 프로그램에 똑 같은 듯한 부분이 중복해서 존재하고, 프로그램 작성의 노력도 어셈블의 시간도 헛수고가 되어 버린다. 분할 어셈블이 가능하면, 부품은 한번 어셈블할 뿐이고 그 다음에는 오브젝트 파일을 링크하면 이용할 수 있다.

(3) 모 듈
프로그램의 부품화와 공유의 사고방식을 더욱 확장한 것이 모듈화의 사고방식이다. 모듈화란 부품의 독립성을 높임으로써 프로그램의 개발효율과 신뢰성을 높이도록 하는 사고방식이다.
하나의 모듈은 밀접하게 관련하는 기능을 가진 프로시저의 집합으로 이루어 진다. 그러면서도 전체로서는 다른 모듈에 의존하거나 영향을 주는 일이 적어야만 한다. 이와 같은 모듈을 부품으로 이용하면, 신뢰성이 높은 프로그램을 효율 좋게 작성할 수 있다. 각각의 기능이 확실하기 때문에 부품으로서 이용하기 쉽고, 부품끼리 서로 영향을 끼침으로써 발생하는 버

그가 적게 끝나기 때문이다.
이용가치가 높은 모듈을 만들어 내면 그것은 하나의 재산이 된다. 새롭게 프로그램을 만들려고 할 때, 이미 완성되어 있는 모듈로부터 필요한 것을 선출해서 이용할 수 있기 때문에, 프로그램 개발을 대단히 효율 좋게 할 수 있기 때문이다.

(4)라이브러리
모듈의 수가 늘어남에 따라 , 파일의 관리나 링크의 조작이 귀찮아 진다. 그것을 간단히 하기 위하여 라이브러리 파일을 작성할 수 있다. 라이브러리 파일은 여러 개의 오브젝트 파일을 하나의 파일로 통합 정리한 것이다

2.5 용어 정리

1. 프로세서(Processor) : 컴퓨터의 기본 연산을 수행하는 메인 시스템 보드(main system board)위에 있는 칩(chip)
2. 명령어 집합(Instruction set) : 프로세서가 수행할 수 있는 모든 연산의 집합
3. 기계 언어(Machine language) : 프로세서가 수행할 수 있는 기본 연산의 집합과 각 연산의 사용 규칙
4. 컴퓨터 언어(Computer language) : 프로그램을 작성할 때 사용되는 명령어의 집합
5. 고급 언어(High level language) : 기계 언어와는 거리가 먼 인간이 사용하는 언어와 유사한 형태로 되어 있는 명령어의 집합
6. 저급 언어(Low level language) : 기계 언어와 유사한 형태로 사람이 프로그램하기 어려운 형태의 낮은 수준의 명령어 집합
7. 어셈블리 언어(Assembly language) : 저급 언어의 한 종류로 기계가 수행할 수 있는 연산과 메모리 주소를 니모닉(nmemonic) 약어로 표현한 언어
8. 컴파일러(Compiler) : 고급언어로 쓰여진 프로그램을 실행 가능한 기계 언어로 변환하여 주는 프로그램
9. 인터프리터(Interpreter) : 고급언어로 쓰여진 프로그램을 한 줄씩 기계 언어로 변환하고 실행하는 프로그램
10. 소스 프로그램(Source program) : 어셈블리 언어로 작성된 프로그램으로 나중에 어셈블러에 의해 기계 언어로 변환
11. 오브젝트 모듈(Object module) : 소스 프로그램이 기계 언어로만 변환된 상태, 저장된 파일의 이름은 OBJ라는 확장자를 갖는다.
12. 로드 모듈(Load module) : 오브젝트 모듈의 실행 가능한 버전, 이는 실행 모듈(run module)이라고도 불린다. 파일에 저장할 때 파일명은 EXE라는 확장자를 갖는다.

<자료>
http://dove.kornu.ac.kr/~yhoh/lecture/assembly/assemly.html

3. DOS와 GUI

- 기존의 지뢰찾기 게임은 Windows상에서 실행될 수 있는 프로그램이다. 하지만 우리가 구현하려는 platform은 DOS이다. 그러므로 Windows 프로그래밍을 할 때와는 또 다른 접근 방법이 필요하다. 그래서 DOS라는 운영체제에 대해서 알아보고 여기서 어떻데 유저인터페이스를 구성할 것인지를 생각해 보았다.

3.1 DOS (Disk Operating System) ; 도스

도스는 PC의 운영체계로서 가장 광범위하게 사용된 최초의 운영체계였다. 최초의 개인용 컴퓨터용 도스는 PC-DOS라고 불렀는데, 빌 게이츠와 그의 신생 회사인 마이크로소프트가 개발하여 IBM에 납품하였다. 이때 그는 MS-DOS라고 불리는 마이크로소프트 버전을 직접 시장에 팔 권리를 유보해 두었었다. 그러나 PC-DOS와 MS-DOS는 내용면에서 거의 같은 것이기 때문에 대부분의 사용자들은 이 둘을 모두 그냥 "도스"라고 불렀다.
도스는 (아직까지도 그렇지만) 그래픽을 지원치 않고 단순히 명령어에 기반을 둔 운영체계였으며, 인터페이스가 상대적으로 단순한 만큼 사용이 편리한 것은 아니었다. 사용자의 명령을 기다리는 도스의 프롬프트는 그저 다음의 모양처럼 생겼다.
C:₩>
마이크로소프트 윈도우 운영체계의 초기 버전은, 실제로는 MS-DOS 운영체계 상에서 돌아가는 일종의 응용프로그램이었으며, 윈도우 운영체계는 아직도 어떤 특별한 목적을 위해 계속 사용되는 도스를 위해 지원을 계속하고 있다.

3.2 MS-DOS (Microsoft Disk Operating System)

MS-DOS[엠에스 도스]는 마이크로소프트사에서 만들어 개인용컴퓨터들에 폭넓게 장착되었던 운영체계이다. 이것은 IBM PC에 장착하기 위해 PC-DOS (Personal Computer DOS)라는 이름으로 빌 게이츠의 회사가 개발했던 운영체계와 기본적으로는 같은 것이었다. 이런 각각의 DOS 시스템을 사용하는 대부분의 사용자들은 단순히 그들의 시스템을 DOS라고 불렀다. PC-DOS처럼 MS-DOS도 여전히 문자형식이며, 명령어에 의해 줄 단위로 움직이는 운영체계로서, 비교적 단순하지만 사용자 측면에서 그리 친숙한 것은 아니었다.
사용자들은 MS-DOS 운영체계 하의 PC를 사용하기 위해서는 MS-DOS에서 지원되는 명령어들을 모두 외우고, 세부적인 사용방법까지를 배워야만 했다. MS-DOS는 미리 정해진 명령어에서 단 한글자라도 틀리면, 가치없이 에러 메시지를 내는 사용하기 매우 불편하며, 익히기 힘든 운영체계였지만, 오랜 기간동안 IBM PC 호환기종의 운영체계로 군림하였고, 버전도 1.0 에서 출발하여 마지막에는 6.22 까지 지속적으로 발전했다. 좌측의 그림은 MS-DOS에서 파일을 복사하는 경우의 예시화면이며, 사용자의 명령을 기다리는 프롬프트와 커서의 모양도 함께 나타나 있다.
현재 대부분의 사용자들이 운영체계를 윈도우95나 윈도우98로 옮겨간 상태이지만, 윈도우 운영체계 조차 처음에는 MS-DOS를 기반으로 동작하는 하나의 응용프로그램에 불과하였으며, 마이크로소프트는 지금까지도 특수한 용도의 활용을 위해 DOS(또는 DOS 지향의 사용자 환경)의 지원을 계속하고 있다.

IBM은 원래 개인용 컴퓨터가 만들어졌던 1970년대 이전에는, 소규모기업용 컴퓨터들에 DOS와는 다르며 전혀 관련이 없는 VSE 라는 독자적인 운영체계를 가지고 있었지만 MS-DOS로 대체되었다 .

3.3 interface ; 인터페이스

명사로서 사용되는 인터페이스라는 용어는 다음 중 하나를 의미한다.
다이얼이나, 조이스틱, 컴퓨터나 프로그램에 의해 제공되는 운영체계의 명령어, 그래픽 표현 형식 기타 다른 장치들과 같이, 사용자가 컴퓨터나 프로그램과 의사소통을 하고 사용할 수 있도록 해주는 사용자 인터페이스
사용자에게 그림을 이용한 의사소통 방법을 제공하는 그래픽 사용자 인터페이스(GUI), GUI는 보통 인간 환경공학적으로 보다 만족스럽고, 사용자 편의를 더 강조한 인터페이스이다.
일련의 명령어나 함수, 옵션, 그리고 프로그램 언어에 의해 제공되는 명령어나 데이터를 표현하기 위한 다른 방법들로 구성되는 프로그래밍 인터페이스
어떤 장치를 커넥터나 다른 장치에 부착할 수 있도록 지원하는 물리적이거나 논리적인 설비

동사로서 사용될 때, 인터페이스 한다는 것은 다른 사람이나 객체와 의사소통 한다는 것을 의미하는데, 특히 하드웨어 장비에 있어 인터페이스 한다는 것은, 두 장비가 효과적으로 교신하거나 함께 일할 수 있도록 적절한 물리적인 연결을 확립하는 것을 의미한다.

<자료>
http://www.terms.co.kr

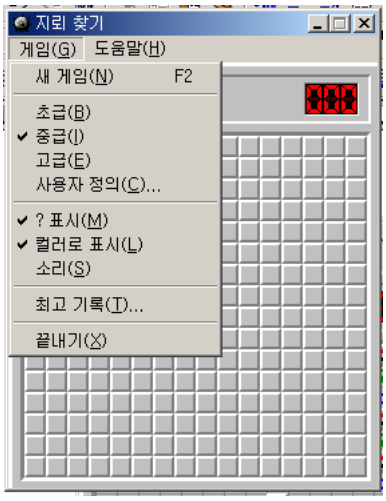
4. 지뢰찾기

- 이번 프로젝트는 Microsoft(R)사의 OS Windows에 기본적으로 설치되어 있는 Microsoft(R) 지뢰 찾기 (버전 5.0 by Robert Donner and Curt Johnson)을 모델로 하기 때문에, 기존의 지뢰 찾기 게임은 어떠한지 살펴보았다.

4.1 개요

지뢰 찾기의 목표는 모든 지뢰를 하나도 터뜨리지 않고 빨리 찾는 것이다. 지뢰를 터뜨리면 게임이 끝난다.

일단 프로그램을 수행시키면, 다음과 같은 화면이 나타난다.



메뉴바를 선택하면 왼쪽의 그림과 같이,

-새로 게임을 시작할 수 있는 바,
-게임의 난이도를 높고 낮출 수 있는 바(전체 지뢰밭의 크기에 따라),
-기타의 옵션 바,
-기존의 최고기록을 보여주는 바,
-프로그램을 종료시키는 바

가 나타난다.

4.2 게임의 규칙

◦상자를 클릭하면 상자가 열린다. 지뢰가 있는 상자를 열면 지뢰가 터지고 게임이 끝난다.
◦상자의 숫자는 그 상자를 둘러싼 상자 여덟 개의 상자에 들어 있는 지뢰의 총 개수를 나타낸다.
◦지뢰가 들어 있다고 생각되는 상자에 표시해 두려면 해당 상자를 마우스 오른쪽 단추로 클릭한다.
◦지뢰 찾기 게임은 게임 영역, 지뢰 카운터 및 타이머로 이루어져 있다.

위는 실제 게임을 하는 장면을 캡쳐한 것이다.

4.3 분석

-기본적으로 기존의 지뢰찾기 프로그램은 윈도우 상에서 돌아가는 위도우 프로그램이기 때문에 창을 만드는 것도 용이하게 graphic user interface를 구현할 수 있음을 알 수 있다. 하지만 우리가 수행하려는 목적은 DOS상에서 assembly 언어를 이용하여서 지뢰찾기를 구현하려는 것이기 때문에 어떻게 하면 DOS상에서 사용자가 용이하게 사용할 수 있는 graphic-like interface를 만들 수 있을 지에 관한 연구가 필요할 것이다.

-지뢰의 개수를 알아보았다.

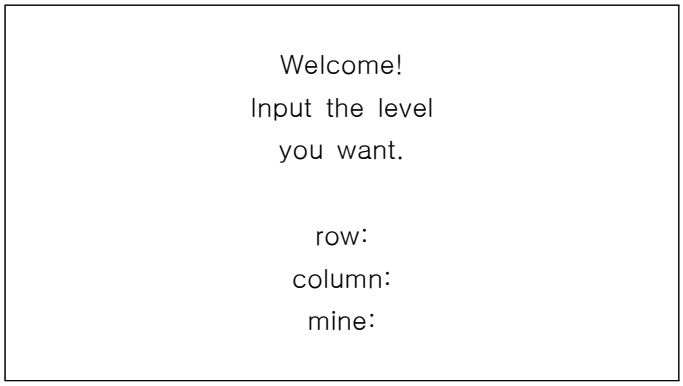
초급 9*9 => 10　　칸8.1당 지뢰1개 12%
중급 15*15 => 40　　칸5.6당 지뢰1개 17%
고급 16*30 => 99　　칸4.8당 지뢰1개 20%

밭의 넓이에 비해 지뢰수가 너무 많거나 너무 적으면 게임의 흥미도가 떨어지게 된다. 그러므로 지뢰의 개수는 기존의 게임의 분포와 비슷하게 12%-20%정도로 발생시키는 것이 좋을 것이다. 게임을 만들 때에 난이도를 기본적으로 상, 중, 하급으로 사용자정의 필드를 하나 만들 수도 있는데 이때에 사용자가 넣을 수 있는 지뢰 수의 제한을 두는 것이 바람직할 것으로 보인다.

Ⅲ. 구현 사항

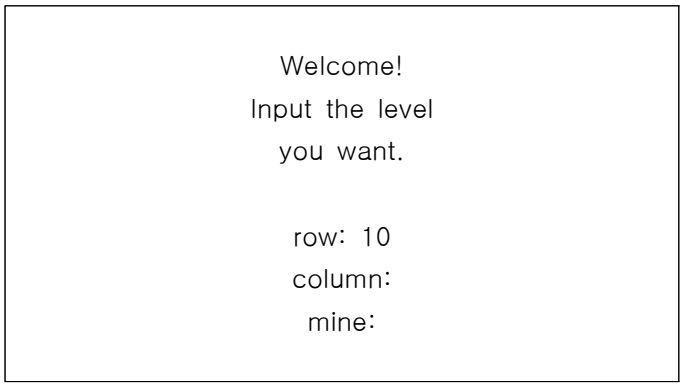
1. 요구사항 분석 및 구현할 기능

1. 게임의 시작



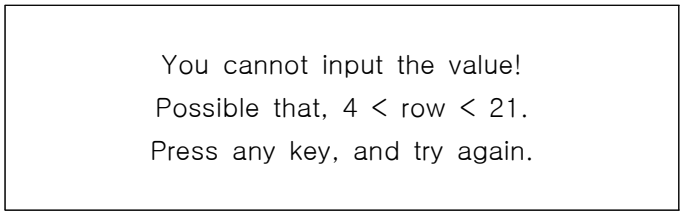
게임을 시작하는 경우는 프로그램을 새로 시작하는 경우와 기존 게임을 중지하고 다시 시작하는 경우가 있다.
게임을 시작하면, 레지스터 및 변수들이 초기화되고, 위와 같은 인터페이스가 출력된다.
게임을 시작하기에 앞서 난이도를 결정하는데, 게임의 난이도는 row, column, 그리고 mine의 수를 입력받아 결정된다. 이들의 값이 클수록 난이도는 높아진다.
프로그램이 초기화되면 row값을 입력받는 위치에 커서가 초기화되어 있게 되며, 숫자를 입력하고 엔터를 누르면 column을 입력받도록 커서가 옮겨지며, 값을 입력하고 또 엔터를 치면 mine을 입력받도록 커서가 옮겨진다. 마찬가지로 값을 입력하고 다시 엔터를 치면 게임이 시작된다.
이 때,
- row의 값이나 column의 값이 5보다 작거나, 또는
- 화면에 출력할 수 없을 정도의 큰 값(최대한계영역을 row는 20칸으로, column은 20칸으로 정한다.)으로 얻어지는 경우, 그리고
- mine의 값이 row*column/2의 값보다 큰 경우에

각각의 단계별로 오류 메시지를 띄우고 해당 부분부터 다시 입력받도록 한다.
그림은 다음과 같다.

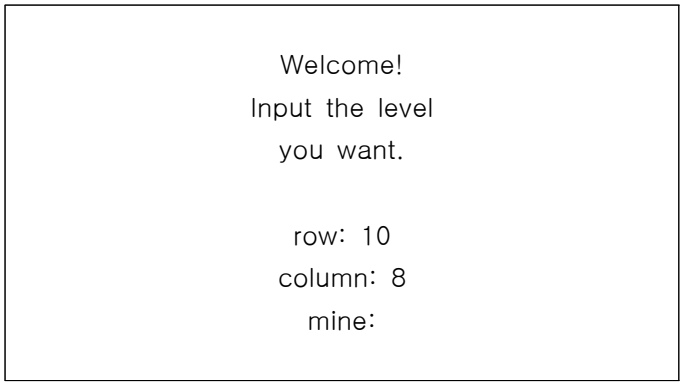


만약 이 때 숫자가 아니거나 5미만의 숫자 또는 20보다 큰 숫자인 경우 다음과 같은 메시지를

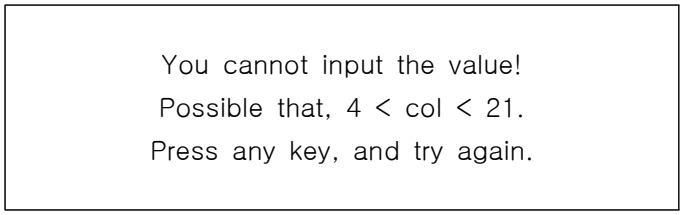
보여주고 아무키나 누르면 다시 row입력 상태로 돌아간다.



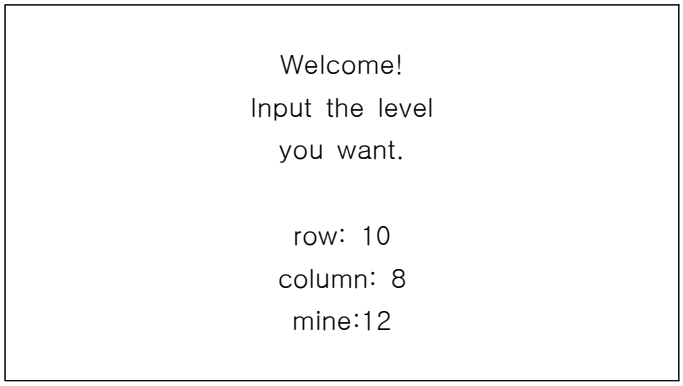
입력 가능한 숫자가 입력되면 column입력 상태로 간다.



만약 이 때 숫자가 아니거나 5미만의 숫자 또는 20보다 큰 숫자인 경우 다음과 같은 메시지를 보여주고 아무키나 누르면 다시 column입력 상태로 돌아간다.



입력 가능한 숫자가 입력되면 mine입력 상태로 간다.



만약 이 때 숫자가 아니거나 0보다 작거나 또는 (row*column)/2보다 큰 숫자인 경우 다음과 같은 메시지를 보여주고 아무키나 누르면 다시 mine입력 상태로 돌아간다.

You cannot input the value!
Possible, 0 < mine < row*col/2
Press any key, and try again.

입력 가능한 숫자가 입력되면 게임이 셋팅된다. 화면은 스크롤 되며 그래픽모드로 전환된다.

2. 게임의 진행

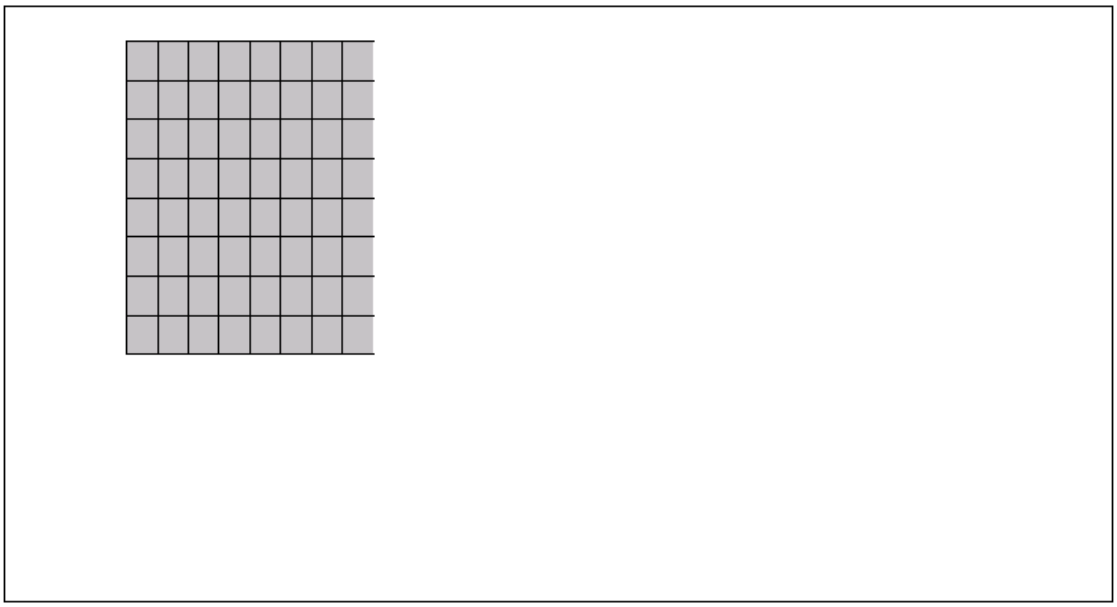
2.1 화면구성

게임은 N(row)*M(column)의 Cell로 구성되며 화면에 나타나는 초기의 각 Cell에는 아무런 표시가 되어있지 않다. (5≤N≤20, 5≤M≤20)

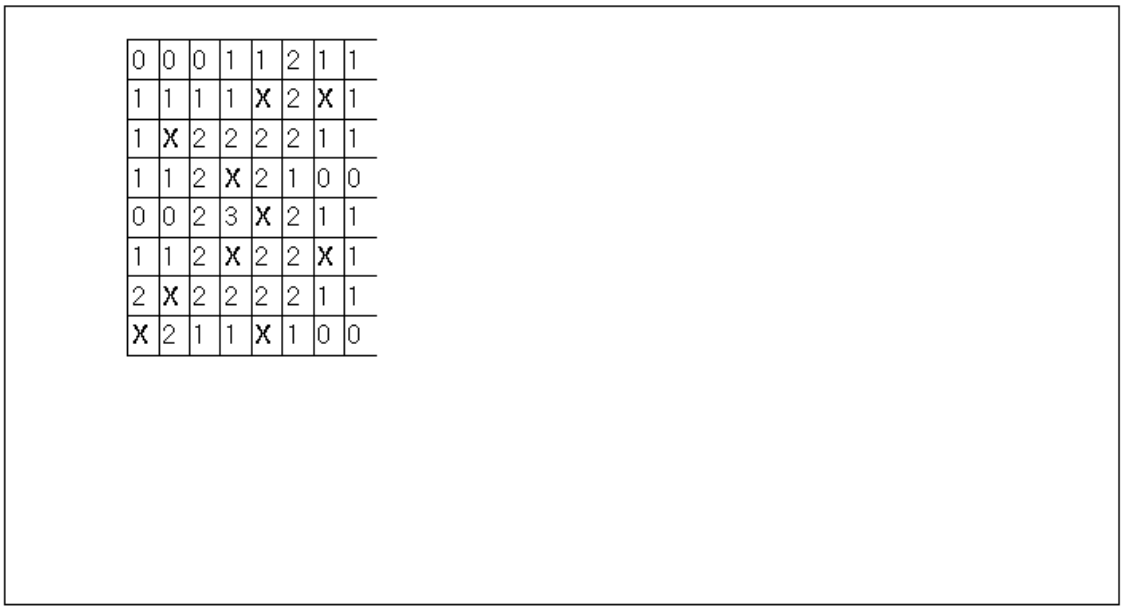
Cell은 지뢰를 가지고 있는 Cell과, 지뢰를 가지고 있지 않은 Cell로 구성되며, 전자의 Cell은 난수를 발생하여 배치되어지며, 후자의 Cell은 인접 Cell에 들어있는 지뢰의 개수에 해당하는 값을 갖고 있고, 그 값은 0~8이다.
Cell의 구현은 2차원 배열을 구현하도록 함으로써 한다.

다음의 예는 각각 게임을 시작한 상태와 보여지지 않는 Cell들의 상태이다.

Example) N=8, M=8, mine=10 인 경우



< 게임이 시작된 후 초기화면 >



< 숨겨진 지뢰와 숫자들 >

* 여기에서 하나의 Cell의 상대적인 크기는 실제 구현할 프로그램과 다를 수 있다. 또 Cell들이 차지하는 화면의 비율은 Cell의 개수에 비례한다. Cell 하나의 크기는 20pixel * 20pixel 로 한다.

2.2 게임 진행 방법

게임이 시작되면, 첫 번째 Cell이 위치하게 된다. 이 상태에서 사용자는 화살표 키를 움직임으로써 다른 Cell로 커서를 이동시킬 수 있다. 커서를 움직임에 있어 벗어난 영역 쪽으로 이동하려고 하면 경고음이 나면서 커서는 이동하지 않고 제자리에 머문다.
사용자는 지뢰가 있다고 예상되는 Cell을 선택하기 위해서 “m"을 선택하여 표시하며, 이 때, 해당 Cell은 팀원의 얼굴 그림으로 바뀐다.
또 사용자는 “c"를 선택하여 이미 표시된 지뢰표시를 제거할 수 있고, 이 때, 해당 Cell은 다시 아무런 표시가 되지 않은 Cell로 바뀐다.
현재 커서가 머물러 있는 곳의 Cell을 열기 위해서는 “space bar"를 선택한다. 이 때, 해당 Cell이 만약 지뢰가 아니라면 자신이 가지고 있는 숫자를 표시하여 주고 2초 후에 자동으로 원래의 상태로 돌아간다. 또 지뢰를 가지고 있다면, 다른 모든 지뢰를 가지고 있는 Cell이 지뢰를 표시하면서 게임은 중지된다.
게임이 중지될 때까지 위의 과정은 반복된다.

2.3 게임의 중지 또는 종료

게임은 임의의 시간에 중지 또는 종료될 수 있다.
게임이 중지되는 경우는 다음과 같다.
1) 사용자가 게임에서 지뢰가 있는 모든 Cell에 표시하여 승리하였을 경우,
2) 사용자가 지뢰가 있는 Cell을 열어 게임에 실패하였을 경우.

1)과 2)의 경우 각각 아래와 같은 메시지를 출력되며 사용자의 응답을 기다린다. 이 때, 사용자가 "Y" 또는 "y"를 누르면 새로 게임이 시작되고, "N" 또는 "n"을 누르면 게임이 종료된다.

Congratulations, U won! U wanna play again?(y/n)

< 1)의 경우 >

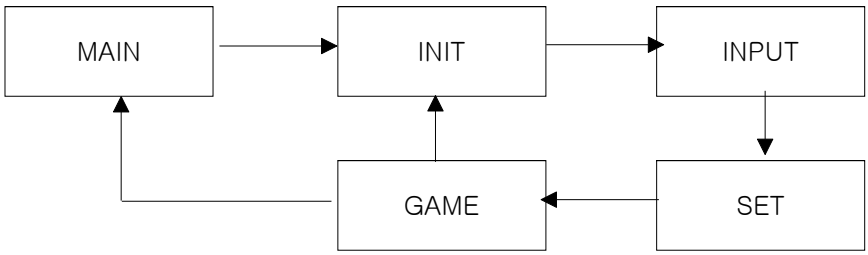
Oops! Sorry, you lost. U wanna play again?(y/n)

< 2)의 경우>

Ⅳ. 프로젝트 설계

1. 전체 프로젝트 설계

[A10MAIN]
  ↓
[B10INIT]
  ↓
[C10INPUT]← ←
  ↓ ↑
<input proper?> no
  ↓ yes
[D10SET]
  ↓
[E10GAME]
  ↓
<what is the input?>
1. 'space bar'
  → <mine?> yes (lost the game) → [E20AGAIN] no → [A10MAIN]
  yes → [B10INIT]
  no(continue)
  → [E30COUNT] → [E40DELAY] → [E10GAME]
2. 'm' → [E50SIGN] → <won the game?> → [E20AGAIN]
  no → [E10GAME]
3. 'c' → [E60ERASE] → [E10GAME]
4. '→←↑↓' → [E70~100] → [E10GAME]



2. 모듈 설계

1. A10MAIN procedure

- file의 type이 .COM 과 .EXE program으로 나뉘는데 복잡한 coding을 위해서는 .EXE program style로 하는 것이 좋을 것이다.
- A10 procedure은 전체적으로 큰 흐름을 잡아주는 부분이다.
- 일단 프로그램을 실행하면 각 sub-procedure을 호출하면서 게임을 운영해나간다.
- B10INIT를 호출한다.
- E10GAME에서 return이 되면 전체 프로그램을 종료시킨다.

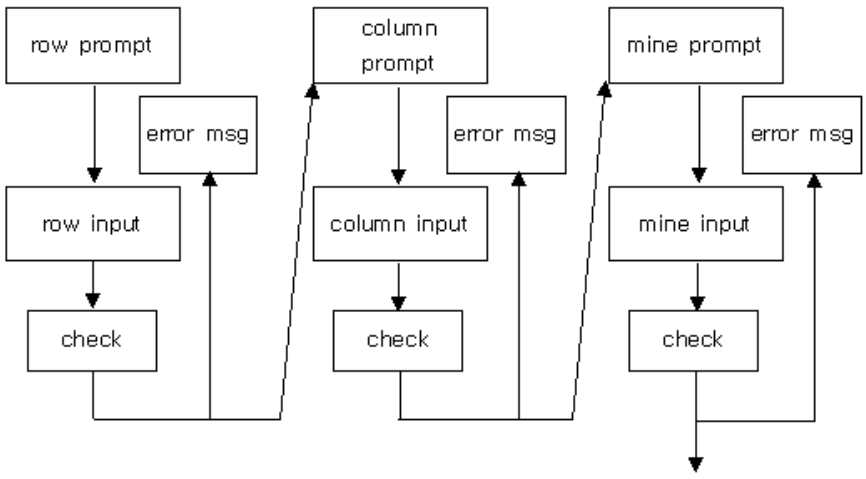
2. B10INIT procedure

- 기존의 video mode를 기억을 한 후에 (INT 10H-0FH 이용)
- 새로운 video mode로 설정을 시킨다. (INT 21H-H, INT 10H-00H, 이용)
- 화면을 클리어하고, 사용자로부터 게임 레벨을 받기 위한 폼, 메시지를 출력한다.(INT 10h - 1301h)

이 때의 모드는 비디오 텍스트 모드이다.

3. C10INPUT procedure

- 실행과 동시에 row, column. mine의 개수를 차례대로 입력을 받는다. 사용자가 입력한 값이 유효한 값인지 각 단계마다 확인한다. (INT 21H-0AH, 3FH 이용)
- 이때 유효하다는 것은 가로의 개수는 5개보다 크고 20개보다 작아야 하고, 세로의 개수는 5개보다는 크고 20개보다는 작아야한다. 그리고 지뢰의 개수는 1개 이상이고 지뢰밭의 넓이의 반을 초과할 수 없다.
- 올바른 입력이라면, D10SET procedure로 넘어간다.



3.1 CALL R10RINPUT

[R10RINPUT]

- 사용자로부터 ROW 값을 입력받는다.
- 사용자가 row의 개수를 잘못 입력했다면 다시 입력하라는 메시지를 주고 다시 입력받도록 프로시저의 처음부분으로 돌아간다.
- 올바른 입력이면. 그 값을 binrow에 저장한 후 return 한다.

3.2 CALL C10CINPUT

[G10MAXMINE]

- 사용자로부터 COLUMN 값을 입력받는다.
- 사용자가 column의 개수를 잘못 입력했다면 다시 입력하라는 메시지를 띄우고 다시 입력받도록 프로시저의 처음으로 돌아간다.
- 올바른 입력이면, 그 값을 bincol에 저장한 후 return 한다.

3.3 CALL G10MAXMINE

[G10MAXMINE]

- 사용자가 입력한 row와 column의 값을 토대도 최대 mine 수를 정하는 프로시저이다.
- 연산 후 return한다.

3.4 CALL M10MINPUT

[M10MINPUT]

- 사용자로부터 mine 값을 입력 받는다.
- 사용자가 mine의 개수를 잘못 입력했다면, 즉 제한 범위를 벗어나는 값을 입력했다면, 다시 입력하라는 메시지를 띄우고 다시 입력받도록 프로시저의 처음으로 돌아간다.
- 올바른 입력이면, 그 값을 binmine에 저장한 후 return한다.

3.5 CALL M10CLEAR

게임의 난이도를 결정하기 위한 게임 레벨 입력이 모두 끝나면 화면을 스크롤 한다.

4. D10SET procedure

- 사용자가 입력한 값을 가지고 지뢰밭(field)을 만드는 부분이다.
- 위 부분에서 설명한 것처럼 좌측 상단에서부터 밭의 끝을 잡아서 2차원 배열을 이용하여서 전체 밭을 만든다.

이때에, 좌측상단의 한계점을 (10H, 40H), 우측하단의 한계점을 (120H, 320H)로 잡고, 지뢰밭을 이루는 한 칸의 크기는 10H*10H로 한다. 그러므로 칸의 개수는 가로 12개 * 세로 28개가 최대의 크기가 된다.

- random하게 그 밭 중에서 지뢰가 있는 것과 지뢰가 없는 것으로 나누어서 그 값을 저장하는 기능까지 하게 된다.

- 그런데 assembly language에서 random number generator를 구현한 것이 text의 453 page에 나와 있는 것을 이용한다. 그 code를 적어보았다.

```
MOV     AX, 0           ; Interval timer
OUT     43H, AL;        via port 34H
IN      AL,40H          ; Make 2 accesses to
IN      AL,40H          ;       port 40H
```

- 당연히 이때 지뢰가 어디 있는지에 대한 정보는 사용자가 볼 수 없어야 한다. 1bit의 flag를 이용하면 될 것으로 생각된다.
- 지뢰밭의 전체적인 윤곽은 text의 182 page의 ASCII code들을 이용한다.

5. E10GAME procedure

- 실제적으로 게임을 진행하는 부분이다.
- 처음 시작 시에는 좌측상단의 칸이 blinking의 상태이다.
- 사용자가 'space bar'를 눌렀다면 E80CHECK를 호출한다.
- 사용자가 'm'을 누르면, E50SIGN을 호출한다.
- 사용자가 'c'을 누르면, E60ERASE를 호출한다.

5.1 E20AGAIN sub-procedure

- 화면에 다시 게임을 시작할지 아니면 그냥 게임을 그만할지를 물어본다.
- 'Y'이나 'y'를 입력받으면 B10INIT을 호출해서 처음부터 다시 시작한다.
- 'N'이나 'n'을 입력받으면 다시 A10MAIN으로 돌아가서 전체 프로그램을 종료시킨다.
- 부적절한 입력이 들어오는 경우에는, beep음을 출력한다.

5.2 E30COUNT sub-procedure

- 사용자가 지정한 칸 주위 8개의 칸에서 지뢰의 개수가 몇 개인지 카운트한다.
- 이 때, 가장자리의 칸인 경우에는 주위 칸의 개수가 3개, 5개가 될 수도 있다.
- 개수를 센 후에 받아서 화면에 출력을 한다.
- 그리고, E40DELAY를 호출한다.

5.3 E40DELAY sub-procedure

- 지정한 칸 주위의 지뢰개수를 출력해준 후 2초가 지난 후에 그 값을 지우는 역할을 수행한다.
- 그런데 assembly language에서는 시간을 재는 함수가 따로 없기 때문에, 어떤 한 procedure를 수행했을 때에 몇 CPU clock이 걸리는 지 확인을 한 후에 이 부분을 반복 수

행함으로써 2초가 되도록 해야 할 것이다.
- 뿌려진 숫자를 지운 후에, E10GAME으로 돌아간다.

5.4 E50SIGN sub-procedure

- 만약에 이미 사진으로 표시된 곳이나 'm'으로 표시된 곳에서의 입력이었다면, 그냥 E10으로 돌아간다.
- 'm'을 누른 칸의 상태가 검정색에서 사진으로 바뀐다.
- 모든 지뢰를 찾은 경우(게임을 이긴 경우), E20AGAIN을 호출하여 게임을 다시 시작할 지를 사용자에게 물어본다.
- 아직 찾지 못한 지뢰가 남아 있는 경우, 다시 E10GAME으로 돌아가서 사용자가 다른 지뢰를 찾는 것을 기다린다.
- 이미 지뢰 check가 되어 있는 곳에 m을 누른 것이라면 그냥 E10으로 돌아간다.

5.5 E60ERASE sub-procedure

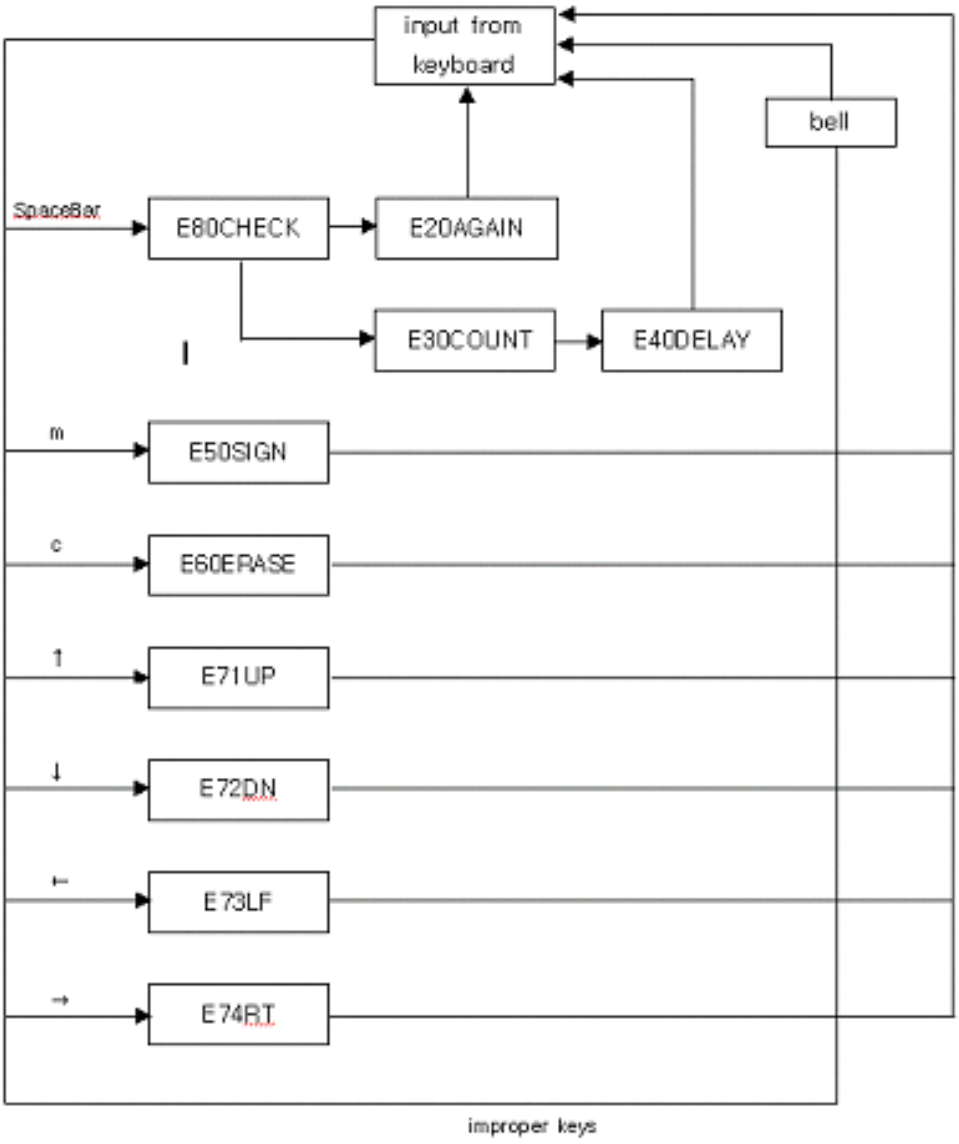
- 기존에 사진으로 되어있던 칸을 검정색의 칸으로 바꾸어준다.
- 사용자가 사진으로 되어 있지 않은 칸을 잘못 누른 것이라면 기존 상태를 유지한다.
- E10GAME으로 돌아간다.

5.6 E71UP, E72DN, E73LF, E74RT sub-procedures

- →←↑↓표시를 누르는 경우에만 유효한 입력으로 인정한다. 그리고 지뢰밭의 제일 끝에서 더 나아가는 방향의 key를 누른 경우와 유효하지 않은 입력을 넣은 경우에는 경고음을 발생시키고 그냥 멈추어 있게 한다.
- →←↑↓표시를 입력으로 받은 경우에 기존의 칸을 blinking 상태에서 no blinking 상태로 바꾸어주고 움직인 후의 칸을 no blinking에서 blinking으로 바꾸어준다.
- 사진이 출력된 상태로 이동이 된다면 blinking하지 않고 그냥 상태를 유지한다.

5.7 E80CHECK sub-procedure

- 이미 사진으로 표시되어 있거나 'm'으로 표시되어있는 곳에서의 입력이었다면 그냥 다시 E10으로 돌아간다.
- 이때 이미 사진으로 표시된 곳에서의 'm'이나 'space bar' 입력은 무시하고 그냥 E10으로 돌아간다.
- 적절한 space이였으면 그 칸의 flag를 보아서,
- 지뢰가 있으면(게임에서 지면), E20AGAIN을 호출한다.
- 지뢰가 없으면, E30COUNT를 호출한다.


input from keyboard
bell
SpaceBar
E80CHECK
E20AGAIN
E30COUNT
E40DELAY
m
E50SIGN
c
E60ERASE
↑
E71UP
↓
E72DN
←
E73LF
→
E74RT
improper keys

Ⅴ. 모듈기능 구현(3차~)

1. 구현하는 모듈에 대한 상세 설명

[A10MAIN]

- 레지스터를 초기화하고,
- INT 10H-0FH를 이용하여 현재의 모드를 저장한다.
- B10INIT을 호출하여 게임을 초기화한다.
- C10INPUT을 호출하여 게임에 필요한 값들을 입력받는다.
- D10SET을 호출하여 그래픽모드로 전환 후 게임을 셋팅한다.
- E10GAME을 호출하여 게임을 진행한다.
- INT 10H-00H를 이용하여 초기의 모드로 되돌아간다.
- INT 21H-4C00H를 이용하여 프로그램을 종료한다.

< Initialization module : B10INIT, C10INPUT >

[B10INIT]

- N10MODE를 호출하여 비디오모드로 전환한다.
- M10CLEAR를 호출하여 화면을 클리어한다.
- P10INITMSG를 호출하여 초기의 메시지를 출력한다.

[C10INPUT]

- Q10RINPUT으로서 ROW의 값을 입력받도록 한다.
- R10CINPUT으로서 COL의 값을 입력받도록 한다.
- S10MINPUT으로서 MINE의 값을 입력받도록 한다.

[M10CLEAR : 화면을 클리어]

- INT 10H-06H를 이용하여 화면 전체를 스크롤한다.
검정색 바탕에 파란색 글씨로 설정하며, 좌표는 (0,0)~(24,79)로 한다.

[P10INITMSG : 초기의 메시지를 출력]

- INT 10H-13H를 이용하여 먼저 그림자를 반복적으로 출력한다. 그림자는 회색이다. 애트릴뷰트를 회색바탕으로 한다.
- 그 위에 스트링을 반복적으로 출력한다. 바탕은 흰색이고 글씨는 검정색으로 한다.
- 이 때 행의 수를 하나 씩 증가시키면서 총 나열해야할 열의 수와 비교하며, 같으면 다 출

력했으므로 다음으로 넘어간다(INC, CMP, 그리고 JNE를 이용).
- 메시지가 출력되는 영역은 (8,10)~(15, 28)이다.
- 메시지가 다 출력되면 입력 모드가 시작된다.

[Q10RINPUT : ROW의 값을 입력]

- "Q20"... INT 10H-02H를 이용하여 ROW를 입력받기 위해 커서 위치를 설정한다.
- INT 21H-0AH를 이용하여 ROW값을 입력받도록 한다.
- 이 입력은 parameter list라는 자료구조를 이용한다(ROWLIST).
- 입력되지 않았다면 Q20으로 돌아간다.
- 입력되었다면 아스키코드 값을 실제숫자로 계산하여 AX에 넣고,
- 이제 이 값이 유효한(6<=ROW<=20) 값인지 조사한다.
- 유효한 값이 아니면 적절한 값을 입력하라는 메시지를 띄우고 Q20으로 돌아간다. 유효한 값이면 ROW에 결과 값을 넣는다.

[R10CINPUT : COL의 값을 입력]

이는 ROW를 입력 받는 형태와 상당히 유사하다.
- "R20"... INT 10H-02H를 이용하여 COL를 입력받기 위해 커서 위치를 설정한다.
- INT 21H-0AH를 이용하여 COL값을 입력받도록 한다.
- 이 입력은 parameter list라는 자료구조를 이용한다(COLLIST).
- 입력되지 않았다면 R20으로 돌아간다.
- 입력되었다면 아스키코드 값을 실제 숫자로 계산하여 AX에 넣고,
- 이제 이 값이 유효한(6<=COL<=20) 값인지 조사한다.
- 유효한 값이 아니면 적절한 값을 입력하라는 메시지를 띄우고 R20으로 돌아간다. 유효한 값이면 COL에 결과 값을 넣는다.

[S10MINPUT : MINE의 값을 입력]

- "S20"... INT 10H-02H를 이용하여 MINE를 입력받기 위해 커서 위치를 설정한다.
그리고 현재까지 입력된 ROW와 COL의 값을 곱하고 2를 나눈 값을 저장해 둔다. 이 값은 MINE의 값이 유효한 값인지를 판별할 때 쓰인다.
- INT 21H-0AH를 이용하여 MINE값을 입력받도록 한다.
- 이 입력은 parameter list라는 자료구조를 이용한다(MINELIST).
- 입력되지 않았다면 S20으로 돌아간다.
- 입력되었다면 아스키코드 값을 실제 숫자로 계산하여 AX에 넣고,
- 이제 이 값이 유효한(1<=MINE<=ROW*COL/2) 값인지 조사한다.
- 유효한 값이 아니면 적절한 값을 입력하라는 메시지를 띄우고 S20으로 돌아간다. 유효한 값이면 MINE에 결과 값을 넣는다.

< Setting Module : D10SET >

[d10set]

values : col, row, mine
output : 지뢰밭을 화면에 뿌려주고 내부적으로 cell을 세팅한다.
called : main
call 1. graphics mode로 전환한다.
2. showfield (field를 화면에 뿌려준다)
3. setcell (내부적으로 cell의 정보를 설정한다.)
4. current cell을 (10H, 40H)~(20H, 50H)으로 설정한다.

[setcell]

values : 상위 procedure에서 입력받은 col, row, mine의 개수
output : 내부적으로 cell에 대한 정보를 setting시킨다.
called : d10set
call : 1. [random] (random number를 generate한다.)
2. [count] (지정된 셀에 대하여 주위 8개의 지뢰개수를 센다.)

[showfield]

values : 상위 procedure에서 사용자에게서 입력받은 col, row, mine의 개수
output : 화면에 col * row 의 field를 뿌려준다.
called : d10set
call : [drawpix]

[random]

values : 현재 시각(random seed)
output : random number
called : setcell
call : X

[count]

values : 기존의 cell정보 (지뢰 설치가 끝난 상태)
output : cell정보 (지뢰가 없는 셀 각각에 count된 숫자가 들어감)
called : setcell
call : X

- random하게 그 밭 중에서 지뢰가 있는 것과 지뢰가 없는 것으로 나누어서 그 값을 저장하는 기능까지 하게 된다.

- 당연히 이때 지뢰가 어디 있는지에 대한 정보는 사용자가 볼 수 없어야 한다. 1bit의 flag를 이용하면 될 것으로 생각된다.
- 지뢰밭의 전체적인 윤곽은 text의 182 page의 ASCII code들을 이용할 수도 있다.

< Game Processing module : E10GAME >

[e10game]module

values : col, row, mine, csize,
output : 전체 게임을 진행시킨다.
called : [a10main]
call 1. keyboard mode:
'm' key 입력 => call e50sign
'c' key 입력 => call e60erase
space bar 입력 => call e70open
2. mouse mode:
right click => 홀수 번째 클릭 => call e50sign
=> 짝수 번째 클릭 => call e60erase
left click => call e60open

eup: label

1. 기존의 frame을 지우고, call recover
2. current cell pointer를 위쪽으로 옮기고,
3. 위쪽 셀에 frame을 뿌린다. call frame

edn: label

1. 기존의 frame을 지우고, call recover
2. current cell pointer를 아래쪽으로 옮기고,
3. 아래쪽 셀에 frame을 뿌린다. call frame

elf: label

1. 기존의 frame을 지우고, call recover
2. current cell pointer를 왼쪽으로 옮기고,
3. 왼쪽 셀에 frame을 뿌린다. call frame

eup: label

1. 기존의 frame을 지우고, call recover
2. current cell pointer를 오른쪽으로 옮기고,
3. 오른쪽 셀에 frame을 뿌린다. call frame

[e50sign]function

value : current cell pointer

output 1. cell정보에서 사용자가 check했다는 정보를 남기고,
2. 현재의 셀에 사진을 뿌려준다.
3. 현재의 셀에 지뢰가 있으면, 승리 점수와 check counter를 증가시킨다.
없었다면, check counter만 증가시킨다.

called : [e10game]

call : [curr] or [mousecurr], [picture]

[e60erase]function

value : current cell pointer

output 1. cell정보에서 사용자 check부분을 0으로 바꾸어 주고
2. 사진을 지우고 셀을 초기 상태로 되돌린다.
3. 현재의 셀에 지뢰가 있으면, 승리점수와 check counter를 감소시킨다.
없었다면, check counter만 감소시킨다.

called : [e10game]

call : [curr] or [mousecurr], [recover]

[e70open]function

value: current cell pointer

output : 1. cell정보를 보고 주위의 지뢰 개수를 출력한다.
2. 2초 동안의 시간 후에 숫자를 지운다.

called : [e10game]

call : [curr], [number], [delay], [recover]

2. 기능의 구현에 대한 상세 설명

< Initialization module : B10INIT, C10INPUT >

[A10MAIN]

현재의 모드를 저장할 때,

- INT 10H-0FH : Get Current Video Mode

현재의 비디오 모드를 결정할 수 있다.

이 연산은 AL에 현재의 비디오 모드를 AH에 스크린 열의 수를, 그리고 BH에 활성화된 비디오 페이지를 반환한다.

CALL & RETn

- CALL은 호출된 프로시저에게 컨드롤을 넘길 때 사용된다. RETn은 호출한 프로시저에게

호출된 프로시저의 결과를 반환할 때 쓰인다.
CALL procedure-name
RET[n] [immediate]

프로그램이 끝나고 초기의 모드로 되돌아 올 때,
- INT 10H-00H : Set Video Mode
비디오 모드를 설정한다. AL에는 필요한 모드가 들어간다. 이 연산은 어떠한 값도 반환하지 않는다. 또한 화면을 clear한다.

게임이 종료되고 프로그램을 다시 시작할 지에 대한 여부를 물을 때,
- "Do you want to play again? (y/n)"를 화면의 하단에 출력한다. INT 10H-13H를 이용한다.

```
mov     ah, 10h
int     16h
```

를 이용하여 키보드로부터 입력받고,
Y나 y이면 B10INIT으로, N나 n이면 프로그램을 종료하는 부분으로 분기한다.

프로그램을 종료할 때,
- INT 21H-4C00H : Terminate Program
Normal exit를 요청한다.

[M10CLEAR : 화면을 클리어]

화면 전체를 스크롤할 때,
- INT 10H-06H : Scroll Up Screen
텍스트나 그래픽 모드에서의 이 연산은 스크린의 특정 영역의 라인들을 스크롤 하도록 한다. 디스플레이된 라인들은 스크린의 위쪽으로 사라지고 새로운 라인들이 아래쪽에서 나타난다.
AL = Numbers of rows(00 for full screen)
BH = Attribute or pixel value
CX = Starting row:column
DX = Ending row:column
이 연산은 어떤 값도 반환하지 않는다.

[P10INITMSG : 초기의 메시지를 출력]

그림자와 문자열을 출력할 때,
- INT 10H-13H : Display Character String
텍스트나 그래픽 모드에서 이 강력한 연산은 속성을 설정하고 커서를 이동시키며, 어떤 길이의 스트링도 화면에 디스플레이 할 수 있도록 한다. ES:BP에 디스플레이될 스트링의 offset주소를 로드한다. 이 연산은 backspace, bell, carriage return 그리고 line feed에서

유효하나 tab키에서는 그렇지 않다.

```
MOV     AH, 13H          ;REQUEST DISPLA STRING
MOV     AL, SUBFUNCTION       ;00, 01, 02, OR 03(SEE BELOW)
MOV     BH, PAGE#       ;PAGE NUMBER
MOV     BL, ATTRIBUTE ;SCREEN ATTRIBUTE
LEA     BP, ADDRESS   ;ADRESS OF STRING IN ES:BP
MOV     CX, LENGTH     ;LENGTH OF STRING
MOV     DH, ROW         ;SCREEN ROW
MOV     DL, COLUMN     ;SCREEN COLUMN
INT     10H              ;CALL INTERRUPT SERVICE
```

SUBFUNCTION
00 Display string and attribute; do not advance cursor
01 Display string and attribute; advance cursor
02 Display character and then attribute; do not advance cursor
03 Display character and then attribyte; advance cursor

그림자(속성)는 회색이다.
실제 메시지 스트링의 바탕은 흰색이고 글씨는 검정색이다.

메시지가 출력되는 영역은 (8,10)~(15, 28)이며, 이 때 행의 수를 하나 씩 증가시키면서 총 나열해야할 열의 수와 비교하며, 같으면 다 출력했으므로 다음으로 넘어간다.
- INC & DEC
INC와 DEC는 레지스터와 메모리의 내용을 하나씩 증가시킬 때 유용하다. INC와 DEC는 하나의 operand가 필요하다.

- CMP
CMP는 두개의 숫자 데이터 필드를 비교할 때 쓰인다.
CMP register/memory, register/memory, memory/immediate

- Jumps Based on Signed (Arithmetic) Data

Symbol	Description	Flags Tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump Not Equal or Jump Not Zero	ZF
JG/JNLE	Jump Greater or Jump Not Less/Equal	OF, SF, ZF
JGE/JNL	Jump Greater or Equal or Jump Not Less	OF, SF
JL/JNGE	Jump Less of Jump Not Greater/Equal	OF, SF
JLE/JNG	Jump Less/Equal or Jump Not Greater	OF, SF, ZF

[R10RINPUT : ROW의 값을 입력]

ROW 값을 입력받기 위해 커서의 위치를 설정할 때,
- INT 10H-02H : Set Cursor Position
텍스트 모드나 그래픽 모등서 이 연산은 row와 column에 따라서 스크린의 어느 위치에나 커서를 설정할 수 있다. function 13H도 같은 역할을 한다.
BH = page number(0 은 디폴트값)
DH = row
DL = column
각 페이지의 커서 위치는 각각 독립적이다. 이 연산은 아무런 값도 반환하지 않는다.
그래픽 모드에서 커서가 보이지는 않더라도 설정할 수는 있다.

ROW 값을 입력받을 때.
- INT 21H-0AH : Buffered Keyboard Input
키보드로부터 데이터를 입력받을 때 강력한 function이다. input 영역은 parameter list라는 특정한 필드영역을 필요로 한다. 이 자료구조는 고급언어에서의 레코드나 구조체에 비교할 수 있다.
첫째로 parameter list의 이름이 필요하다.
둘째로, 입력받는 문자들의 최대 수로서 입력받는 문자들의 수를 제한한다.
셋째로, 실제 입력받아지는 문자들의 숫자를 기억하는 변수가 필요하다.
마지막으로, 실제 입력될 문자들의 주소(공간)가 필요하다.

입력되지 않았다면,
- Q20(다시 입력받기 위해)으로 돌아간다.

한 자리의 수가 입력되었다면,
- 그것은 하나의 문자에 대한 ASCII값이 받아진 것이므로 여기에서 30H를 뺀다. 그러면 실제의 숫자가 된다.

두 자리의 수가 입력되었다면,
- 그것은 두 문자에 대한 ASCII값이 받아진 것이므로 각각에서 30H를 빼고, 앞자리에(십의 자리 수) 받아진 것에서 30H을 뺀 것은 10을 곱하고, 뒤에(일의 자리 수) 받아진 것과 더하면 실제의 숫자가 된다.

위에서의 결과를 AX에 넣고 이제 이 값이 유효한(6<=ROW<=12) 값인지 조사한다.
- AX의 값이 12보다 크면 적절한 값을 입력하라는 메시지를 띄우고(Q50), Q20으로 돌아간다(JMP Q20:).
- 또 6보다 작으면 적절한 값을 입력하라는 메시지를 띄우고(Q50), Q20으로 돌아간다(JMP Q20:).
- 유효한 값이면 ROW에 결과 값을 넣고 프로시저를 마친다.

절절한 입력을 하라는 메시지("Valid value of the row : 6<=ROW<=12") 띄우기
- INT 10H-13H를 이용하여 화면 아래쪽에 문자열을 출력한다.
- esc키를 누르면 사라진다(메시지가 뿌려진 부분에 다시 검정색으로 덮어씌워 사라지게 한다).

[C10CINPUT : COL의 값을 입력]

COL 값을 입력받기 위해 커서의 위치를 설정할 때,
- INT 10H-02H : Set Cursor Position
을 이용한다.

COL 값을 입력받을 때.
- INT 21H-0AH : Buffered Keyboard Input
을 이용한다.

입력되지 않았다면,
- R20(다시 입력받기 위해)으로 돌아간다.

한 자리의 수가 입력되었다면,
- 그것은 하나의 문자에 대한 ASCII값이 받아진 것이므로 여기에서 30H를 뺀다. 그러면 실제의 숫자가 된다.

두 자리의 수가 입력되었다면,
- 그것은 두 문자에 대한 ASCII값이 받아진 것이므로 각각에서 30H를 빼고, 앞자리에(십의 자리 수) 받아진 것에서 30H을 뺀 것은 10을 곱하고, 뒤에(일의 자리 수) 받아진 것과 더하면 실제의 숫자가 된다.

위에서의 결과를 AX에 넣고 이제 이 값이 유효한(6<=COL<=12) 값인지 조사한다.
- AX의 값이 28보다 크면 적절한 값을 입력하라는 메시지를 띄우고(R50), R20으로 돌아간다(JMP R20:).
- 또 6보다 작으면 적절한 값을 입력하라는 메시지를 띄우고(R50), R20으로 돌아간다(JMP R20:).
- 유효한 값이면 COL에 결과 값을 넣고 프로시저를 마친다.

절절한 입력을 하라는 메시지("Valid value of the column : 6<=COL<=28") 띄우기
- INT 10H-13H를 이용하여 화면 아래쪽에 문자열을 출력한다.
- esc키를 누르면 사라진다(메시지가 뿌려진 부분에 다시 검정색으로 덮어씌워 사라지게 한다).

[M10MINPUT : MINE의 값을 입력]

MINE 값을 입력받기 위해 커서의 위치를 설정할 때,
- INT 10H-02H : Set Cursor Position
을 이용한다.

그리고 나서, 현재까지 입력된 COL과 ROW의 값을 곱하여 2로 나눈 값을 저장해둔다.
곱할 때,
- MUL : Word Times Word
두 개의 워드단위의 값을 곱하기 위해서 곱해지는 값은 AX에 있고 곱하는 값은 메모리나 다른 레지스터에 하나의 워드로서 저장되어 있다.
나눌 때,
- SHR/SAR : Shift Logical Right/ Shift Arithmetic Right
SHR/SAR register/memory, CL/immediate
두 번째 피 연산자는 Shift할 만큼의 값이고, 그것은 immediate의 값이거나 CL 레지스터의 참조이다.
SHR와 SAR의 차이점은 SHR은 unsigned bit의 경우에, SAR은 signed bit의 경우에 사용된다는 것이다.

MINE 값을 입력받을 때.
- INT 21H-0AH : Buffered Keyboard Input
을 이용한다.

입력되지 않았다면,
- S20(다시 입력받기 위해)으로 돌아간다.

한 자리의 수가 입력되었다면,
- 그것은 하나의 문자에 대한 ASCII값이 받아진 것이므로 여기에서 30H를 뺀다. 그러면 실제의 숫자가 된다.

두 자리의 수가 입력되었다면,
- 그것은 두 문자에 대한 ASCII값이 받아진 것이므로 각각에서 30H를 빼고, 앞자리에(십의 자리 수) 받아진 것에서 30H을 뺀 것은 10을 곱하고, 뒤에(일의 자리 수) 받아진 것과 더하면 실제의 숫자가 된다.

위에서의 결과를 AX에 넣고 이제 이 값이 유효한(1<=MINE<=ROW*COL/2) 값인지 조사한다.
- AX의 값이 ROW*COL/2보다 크면 적절한 값을 입력하라는 메시지를 띄우고(S50), S20으로 돌아간다(JMP S20:).
- 또 1보다 작으면 적절한 값을 입력하라는 메시지를 띄우고(S50), S20으로 돌아간다(JMP S20:).

- 유효한 값이면 MINE에 결과 값을 넣고 프로시저를 마친다.

절절한 입력을 하라는 메시지("Valid value of the mine : 1<=MINE<=ROW*COL/2") 띄우기
- INT 10H-13H를 이용하여 화면 아래쪽에 문자열을 출력한다.
- esc키를 누르면 사라진다(메시지가 뿌려진 부분에 다시 검정색으로 덮어씌워 사라지게 한다).

< Setting module : D10SET >

[showfield]

- graphic mode로 전환하기 위해서는 일단 int 10h - function 0FH를 이용해서 기존의 mode를 저장한다.

```
mov     ah,0fh          ;get orginal video mode
int     10h
push    ax              ;and save
```

- 그리고 int 10h - function 00H를 이용하여 VGA mode로 전환한 후, 파레트를 설정한다.

```
mov     ah,00h
mov     al,12h
int     10h             ;video mode vga

mov     bh,00           ;background red
mov     bl,04           ;palette
int     10h
```

- 지뢰밭의 전체적인 윤곽을 나타내는 방법에는 여러 가지가 있을 수 있다.

1. 첫 번째로 가능한 방법은, ASCII code에 포함되어 있는 여러 가지 기호들을 이용하는 것이다.
- 이렇게 구현하였을 때는 INT 10H - function 09H를 이용해야 한다.
간단한 코드를 살펴보면 다음과 같다.

```
mov     ah,09h          ;request display
mov     al,0c4h         ;solid single line
mov     bh,00           ;page 0
mov     bl,1eh          ;blue background, brown foreground
mov     cx,25           ;25 repetitions
```

```
        int     10h
```

- 그리고 사용될 수 있는 ASCII code들을 나열하면 다음과 같다.

characters	single line		double line		mixed lines			
straight lines:								
1.horizontal	c4h	─	cdh	═				
2.vertical	b3h	│	bah	‖				
corners:								
top left	dah	┌	c9h	╔	d6h	╓	d5h	╒
top right	bfh	┐	bbh	╗	b7h	╖	d8h	╕
bottom left	c0h	└	c8h	╚	d3h	╙	d4h	╘
bottom right	d9h	┘	bch	╝	bdh	╜	beh	╛
middle:								
left	c3h	├	cch	╠	c7h	╟	c6h	╞
right	b4h	┤	b9h	╣	b6h	╢	b5h	╡
top	c2h	┬	cbh	╦	d2h	╥	d1h	╤
bottom	c1h	┴	cah	╩	d0h	╨	cfh	╧
center	c5h	┼	ceh	╬	d7h	╫	d8h	╪

2. 두 번째로 가능한 방법은 pixel을 직접 화면에 뿌려주는 방법이다.

- 아무래도 ASCII code를 이용하는 방법은 다소 화면의 질이 떨어지기 때문에 pixel을 이용하는 방법이 나을 것 같다.
- 이 때에는 INT 10H - function 0CH를 이용해야 한다.
기본 코드를 살펴보면 다음과 같다.

```
;page 0의 좌표값(200,50)에 점을 찍는다.
        mov     ah,0ch          ;request write dot
        mov     al,03           ;color of the pixel
        mov     bh,0            ;page number
        mov     cx,200          ;horizontal x-coordinate(column)
        mov     dx,50           ;vertical y-coordinate(row)
```

- 점 하나 하나를 반복적으로 찍어나가면 선을 그을 수 있다.
row값을 가지고 있는 DX를 일정하게 유지시키면서 column값을 가지고 있는 CX를 연속적

으로 INC시키면 가로축의 선을 그을 수 있다.
반대로 CX를 일정하게 유지시키면서 DX를 연속적으로 INC시키면 세로축의 선을 그을 수 있다. 그러므로 가로축을 다 표현하기 위해 이중 loop이 필요하게 되고, 또한 세로축을 표현하기 위해 이중 loop이 필요하게 된다.

-알고리즘-
1. 좌표를 (10h,40h)(col,row)에 이동시킨다.
2. for col = 10 ~ # of col (max: 120)
    2.1 for row = 40 ~ # of row(max: 320)
        2.1.1 현재 좌표값에 pixel점을 찍는다.
        2.1.2 col = col + 1
        2.1.3 if col != # of col, goto 2.1.1
            if col == # of col, goto 2.2 (한 개의 가로선 완성)
    2.2 row = row +10
    2.3 if row != # of row, goto 2.1.1
        if row == # of row, goto 3 (모든 가로선 완성)
3. for row = 40 ~ # of row
    3.1 for col = 10 ~ # of col
        3.1.1 현재 좌표값에 pixel점을 찍는다.
        3.1.2 row = row + 1
        3.1.3 if row != # of row, goto 3.1.1
            if row == # of row, goto 3.2 (한 개의 세로선 완성)
    3.2 col = col + 10
    3.3 if col != # of col, goto 3.1.1
        if col == # of col, (모든 세로선 완성)

- 이때에, 좌측상단의 한계점을 (10H, 40H), 우측하단의 한계점을 (120H, 320H)로 잡고, 지뢰밭을 이루는 한 칸의 크기는 10H*10H로 한다. 그러므로 칸의 개수는 가로 12개 * 세로 28개가 최대의 크기가 된다.

[setcell]

- 이 함수는 전체의 셀들 중에 어떤 셀에 지뢰를 설치할 지를 정하는 기능을 수행한다.

- 그리고 각각의 셀에 지뢰가 존재하는지에 대한 정보와 지뢰가 존재하지 않았을 때는 주위에 지뢰가 몇 개있는지에 대한 정보를 이미 가지고 있도록 한다.
또한 사용자가 지뢰가 있을 것 같은 셀에 표시를 하면 그것을 저장할 공간이 필요할 것이다.
그래서 한 셀에 대한 정보를 저장하기 위해서는 한 byte가 필요하다. 그러므로 모든 셀에 대한 정보를 저장하기 위해서는 col*row*1byte만큼의 공간이 필요하다.
그림으로 표시해보면 다음과 같다.

| 0001 1111 | : 지뢰가 설치되어 있고(1111), 사용자가 깃발을 표시한 셀(0001)
| 0000 0101 | : 지뢰가 없고 주위에 6개(0101)의 지뢰가 설치되었고, 사용자가 깃발을 표시하지 않은 셀

- assembly상에서는 배열이라는 abstract data type이 존재하지 않기 때문에 DB를 이용하여 string을 잡은 다음에 2차원의 연산을 따로 정의해야 할 것이다.
예를 들어 설명한다면, col의 개수가 10이고 row의 개수가 5라고 하자.
그러면 (0,0)~(10,0)의 셀을 access하는 것은 그냥 col의 값만으로 가능하다.
그러나 (8,5)의 셀을 access하기 위해서는 (0,0)에서부터 pointer를
(col의 개수) * 5 + 8
만큼 오른쪽으로 옮겨야 할 것이다.
그러므로 일반적인 경우로 생각해보면, (x,y)의 셀을 access하기 위해서는
(col의 개수) * y + x
만큼 pointer를 옮겨야 할 것이다.

[random]

- 지뢰는 물론 random하게 설치되어야 한다.
- assembly language에서 random number generator를 구현한 것이 text의 453 page에 나와 있는 것을 이용한다. 그 code를 적어보면,

```
mov     ax,0            ; Interval timer
OUT     43H, AL         ; via port 43H
INT     AL,40H          ; Make 2 accesses to
INT     AL,40H          ;       port 40H
```

이 코드가 실행되면 register AL에 random한 값이 저장된다.

- random한 값을 받으면 그 값을 나머지 연산을 이용해서 지뢰가 위치하게 될 셀을 지정하게 된다.

[count]

- 지뢰가 없는 cell 주위에 몇 개의 지뢰가 있는지를 counting하는 기능을 수행한다.
- 지정된 cell(cell[i])의 앞뒤의 cell에 지뢰가 있는지 살피고, (cell[i-1],cell[i+1]) 윗줄에 같은 위치에 있는 cell과 그 앞뒤의 지뢰를 살피고,(cell[i-col],cell[i-col-1],cell[i-col+1) 아랫줄에 같은 위치에 있는 cell과 그 앞뒤의 지뢰를 살피면 된다. (cell[i+col],cell[i+col+1],cell[i+col-1])
- 이때, 가장자리에 있는 cell들은 예외적으로 처리를 해주어야 한다.

< Game Processing module : E10GAME >

[e10game]module

1. keyboard의 입력을 받는다.(e20loop:)
- 방향키를 입력으로 받아야 하는데 방향키의 ASCII code는 모두 E0이므로 scan code까지 입력받아야 한다. 필요한 interrupt와 code값은 다음과 같다.

```
mov     ah,10h          ;read keyboard character
int     16h             ;키보드 입력까지 기다린다.
```

	normal(scan, ascii)	
c and C	2E	63
m and M	32	6D
space	39	20
DnArrow	50	00
UpArrow	48	00
RtArrow	4D	00
LfArrow	4B	00

2. 입력받은 값에 따라서 e50sign, e60erase, e70open등의 함수를 호출하거나 esign, eerase, eopen등의 label로 jump한다.

[delay]

output : cell을 연 상태로 두었다가 2초 후에 닫는다.
delay를 주는 방법에는 여러 가지가 있을 수 있는데
명령어 하나를 수행하는데 걸리는 clock수를 알아낸 다음에 그것을 반복해서 수행하는 방식으로 한다.

[e50sign]

output : cell정보에서 사용자가 check했다는 정보를 남기고 사진을 뿌려준다.
current cell정보에서 사용자check부분이 1로 되어있다면 그냥 무시하고 부모 프로시져로 돌아가고 0으로 되어있다면 1로 셋팅을 시킨 후에, 사진을 화면에 뿌려준다.
일단 사진을 10H*10H의 bitmap으로 만든 다음에 화면에 활성화된 셀에 mapping한다.

[e60erase]

output : cell정보에서 사용자 check부분을 0으로 바꾸어 주고 cell을 닫는다.
cell정보에서 사용자check부분이 1로 되어있다면 0으로 바꾸어 주고,

사진을 지우고 기존의 attribute로 복귀한다.
이때에는 INT 10h - function 0CH를 이용하여 이중루프를 돌린다.
0으로 이미 되어있다면 그냥 부모 프로시져로 리턴한다.

eup: label

output : current cell을 위쪽의 cell으로 바꾼다.
current가 cell[i]이었다면 cell[i-col]으로 바꾸어 주고
깜빡거리는 화면의 cell이 (x, y)이었다면 그것의 blinking을 없애고
(x-10h, y)를 blinking으로 바꾼다.
만약에 젤 위쪽의 cell이었다면 아무 수행을 하지 않는다. cell[0]~cell[col]

edn: label

output : current cell을 아래쪽의 cell으로 바꾼다.
current가 cell[i]이었다면 cell[i+col]으로 바꾸어 주고
깜빡거리는 화면의 cell이 (x, y)이었다면 그것의 blinking을 없애고
(x+10h, y)를 blinking으로 바꾼다.
만약에 젤 아래쪽의 cell이었다면 그냥 넘어간다. cell[(row-1)*col]~cell[row*col]

elf: label

output : current cell을 왼쪽의 cell으로 바꾼다.
current가 cell[i]이었다면 cell[i-1]으로 바꾸어 주고
깜빡거리는 화면의 cell이 (x, y)이었다면 그것의 blinking을 없애고
(x, y-10h)를 blinking으로 바꾼다.
만약에 젤 왼쪽의 cell이었다면 그냥 넘어간다. cell[n*col+1]

ert: label

output : current cell을 오른쪽의 cell으로 바꾼다.
current가 cell[i]이었다면 cell[i+1]으로 바꾸어 주고
깜빡거리는 화면의 cell이 (x, y)이었다면 그것의 blinking을 없애고
(x, y+10h)를 blinking으로 바꾼다.
만약에 젤 왼쪽의 cell이었다면 그냥 넘어간다. cell[n*col-1]

VI. 프로그램 구현(4차~)

1. 본 프로젝트의 모듈의 다이어그램 및 상세 설명

=> 전체 프로젝트의 모듈 다이어그램은 다음 장에 첨부한다.

[MAIN PROCEDURE] 지뢰 찾기 게임의 메인 모듈은

- 게임을 시작하기 전에 모든 상태를 초기화 하고, (CALL B10INIT)
- 입력 모드로 전환하여 게임에 필요한 값들을 입력 받은 후, (CALL C10INPUT)
- 게임의 진행 모드를 (키보드 / 마우스) 결정을 입력받으며,
- 게임을 셋팅, (CALL D10SET)
- 게임의 진행, (CALL E10GAME)
- 게임의 종료와 재시작의 분기,
- 프로그램 종료

를 수행한다.

구체적인 구조는 다음 장의 다이어그램을 참고.

```
;================================================================================
======
A10MAIN PROC    FAR
        MOV     AX, @data
        MOV     DS, AX
        MOV     ES, AX

A10AGAIN:
        MOV     AH, 0FH         ;REQUEST GET CURRENT MODE
        INT     10H
        PUSH    AX

        CALL    B10INIT         ;INITIAL
        CALL    C10INPUT

AGN:                            ;SELECTING INPUT MODE
        MOV     AH, 10H
        INT     16H
        CMP     AL, 'M'         ;GO MOUSE MODE
        JE      AMODE
        CMP     AL, 'm'
        JE      AMODE
        CMP     AL, 'k'         ;GO KEYBOARD MODE
        JE      AMODE
```

```
        CMP     AL, 'K'
        JE      AMODE
        JMP     AGN

AMODE:                          ;CONVERTING PART
        MOV     INPUTMODE, AL
        MOV     AX,BINROW
        MOV     ROW,AL
        MOV     AX,BINCOL
        MOV     COL,AL
        MOV     AX,BINMINE
        MOV     MINE,AX
        MOV     WIN,0

        MOV     AL,COL
        MOV     AH,00H
        MUL     ROW
        MOV     TOTAL,AX                ;TOTAL # OF CELLS = COL*ROW

        MOV     AH,00H          ;FUNCTION
        MOV     AL,12H          ;REQUIRED MODE
        ;MOV    AL,13H
        INT     10H             ;VIDEO MODE VGA
        MOV     CSIZE,20        ;12H MODE=>CELL SIZE20

        CALL    D10SET
        CALL    E10GAME

        MOV     AH,00H
        MOV     AL,03H
        INT     10H             ;TEXT MODE

        CMP     USERWIN,1
        JE      A12WIN
        CMP     USERWIN,0
        JE      A13LOSE
A12WIN:
        CALL    W10WIN
        JMP     A20ASK

A13LOSE:
        CALL    L10LOSE

A20ASK:
        MOV     AH,10H          ;AGAIN
        INT     16H             ;GET A KEYBOARD INPUT
```

```
        CMP     AL,'y'
        JE      A30INIT
        CMP     AL,'Y'
        JE      A30INIT
        CMP     AL,'n'
        JE      EXITP
        CMP     AL,'N'
        JE      EXITP
        JMP     A20ASK

A30INIT:
        POP     AX
        MOV     AH, 00H         ;REQUEST RESET MODE
        CALL    GAMEAGAIN
        JMP     A10AGAIN

EXITP:
        POP     AX
        MOV     AH, 00H         ;REQUEST RESET MODE
        INT     10H
        MOV     AX, 4C00H
        INT     21H
A10MAIN ENDP
;-------------------------------------------------------------------------
-------
```

여기까지가 main 프로시저 이며 이제 메인 프로시저에서 초기에 호출되는 b10init과 c10init에 대해 살펴본다.

..... 1. B10INIT MODULE .....

[B10INIT]

=> 이 모듈은 다음과 같이 N10MODE, M10CLEAR, P10INITMSG을 호출하여, 모드를 전환하고, 화면을 클리어 하고, 초기의 INPUT을 위한 메시지를 출력한다.

```
;-------------------------------------------------------------------------
-------
;       [B10INIT]
;       1. INITIALIZE THE DISPLAY MODE
;-------------------------------------------------------------------------
-------
B10INIT  PROC    NEAR
```

```
        CALL    N10MODE                 ;GET/SET MODE
        CALL    M10CLEAR                ;SCROLL FULL SCREEN
        CALL    P10INITMSG      ;INITIALIZE A GAME
        RET
B10INIT ENDP
;-------------------------------------------------------------------------
```

[N10MODE] 비디오 모드에서 표준 컬러를 출력할 수 있도록 인터럽트를 걸어 모드를 설정한다.

```
;-------------------------------------------------------------------------
-------
;       [SET MODE]
;       : STANDARD COLOR TEXT MODE FOR INPUTTING
;-------------------------------------------------------------------------
-------
N10MODE         PROC    NEAR
        PUSHA
        MOV     AH, 00H         ;Request new mode
        MOV     AL, 03H         ;Standard color
        INT     10H
        POPA
        RET
N10MODE         ENDP
;-------------------------------------------------------------------------
-------
```

[M10CLEAR] 인터럽트로써 화면 전체를 스크롤 한다.

```
;-------------------------------------------------------------------------
-------
;       [CLEAR SCREEN]
;-------------------------------------------------------------------------
-------
M10CLEAR        PROC    NEAR
        PUSHA
        MOV     AX, 0600H       ;SCROLL FULL SCREEN
        MOV     BH, 04H         ;RED ON BLACK
        MOV     CX, 0000        ; FROM(0, 0)
        MOV     DX, 184FH       ; TO(24, 79)
        INT     10H
        POPA
        RET
M10CLEAR        ENDP
;-------------------------------------------------------------------------
```

```
-------
```

[P10INITMSG] 게임에 필요한 ROW, COLUMN, 그리고 MINE의 수를 입력받기 위한 메시지 폼을 출력한다. 이는 다음과 같은 데이터들을 필요로 한다. 일정한 애트리뷰트와 화면상에서의 위치, 메시지를 데이터 세그먼트에 저장해 놓고 사용한다.

```
;-------------------------------------------------------------------------
------
;       INITIAL MESSAGE DATA
;-------------------------------------------------------------------------
------
TOPROW EQU    09              ;TOP OF THE RANGE OF ROW
BOTROW EQU    16              ;BOTTOM OF THE RANGE OF ROW
LEFCOL  EQU   30              ;LEFT OF THE RANGE OF COLUMN
MSGAT  DW     0070H           ;INITIAL MESSAGE ATTRIBUTE - BLACK ON WHITE
SHDAT  DW     0008H           ;SHADOW        ATTRIBUTE - GRAY ON BLACK
SHADOW DB     19 DUP(0DBH)
INITMSG DB    0C9H, 17 DUP(0CDH), 0BBH
       DB     0BAH, '     Welcome !   ', 0BAH
       DB     0BAH, ' Input the level ', 0BAH
       DB     0BAH, '    you want.    ', 0BAH
       DB     0BAH, '  row    :       ', 0BAH
       DB     0BAH, '  column :       ', 0BAH
       DB     0BAH, '  mine   :       ', 0BAH
       DB     0C8H, 17 DUP(0CDH), 0BCH
;-------------------------------------------------------------------------
--------
```

인터럽트를 걸어 하나의 스트링을 출력하고 스트링의 옵셋을 증가시킨 후 또 그 위치의 스트링을 인터럽트를 통해 출력하는 방법으로 인터럽트를 반복적으로 걸어 구현하였다.

```
;-------------------------------------------------------------------------
-------
;       [INITIALIZE A GAME BY SHOWING MESSAGE]
;       : DISPLAY SHADOW BOX, AND NEXT MESSAGE
;-------------------------------------------------------------------------
-------
P10INITMSG      PROC    NEAR
       PUSHA
       MOV     AX, 1301H              ;DISPLAY SHADOW AND ADVENCE CURSOR
       MOV     BX, SHDAT              ;PAGE 0, GRAY ON BLACK(GRAY SHADOW)
       LEA     BP, SHADOW
       MOV     CX, 19
       MOV     DH, TOPROW+1
       MOV     DL, LEFCOL+1
```

```
P20:
        INT     10H
        INC     DH
        CMP     DH, BOTROW+2
        JNE     P20

        MOV     AX, 1301H               ;DISPLAY MESSAGE AND ADVENCE CURSOR
        MOV     BX, MSGAT               ;PAGE 0, BLACK ON WHITE
        LEA     BP, INITMSG
        MOV     CX, 19
        MOV     DH, TOPROW
        MOV     DL, LEFCOL

P30:
        INT     10H
        ADD     BP, 19
        INC     DH
        CMP     DH, BOTROW+1
        JNE     P30

        POPA
        RET
P10INITMSG      ENDP
;-------------------------------------------------------------------------------
-------
```

..... 2. C10INPUT MODULE .....

[C10INPUT]

=> 이 모듈은 다음과 같이 R10RINPUT, C10CINPUT, G10MAXMINE, M10MINPUT, M10CLEAR , MODEMSG을 호출하여, ROW, COLUMN, 그리고 MINE의 수를 입력받는다. 에러에 대한 처리도 한다. 또 게임 진행 모드(키보드 / 마우스)를 선택하도록 하는 메시지를 출력한다.

```
;-------------------------------------------------------------------------------
-------
;       [C10INPUT]
;       1. GET THE # OF COLUMNS, ROWS, AND MINES
;-------------------------------------------------------------------------------
-------
C10INPUT        PROC    NEAR
        CALL    R10RINPUT               ;INPUT THE value OF THE ROW
```

```
        CALL    C10CINPUT               ;INPUT THE value OF THE COLUMN
        CALL    G10MAXMINE      ;GET THE LIMIT OF THE value OF MINE
        CALL    M10MINPUT               ;INPUT THE value OF THE MINE
        CALL    M10CLEAR                ;SCROLL FULL SCREEN
        CALL    MODEMSG                 ;DISPLAY MESSAGE ABOUT SELECTION MODE
        RET
C10INPUT        ENDP
;-------------------------------------------------------------------------------
-------
```

이러한 입력 모드에서는 다음과 같은 데이터 공간이 필요하다. row와 col의 한계 범위는 정적으로 주어지고, mine의 한계 범위는 입력되는 row와 column에 따라서 다른 서브프로시저를 통해 계산된다.

여기서 중요한 것은 키보드로부터 받는 값들은 ASCII값들이기 때문에 사용자가 원하는 값을 실제로 사용하기 위해서는 Binary로 값으로 전환하는 과정이 필요하다. 입력받는 값들이 한 자리일 때, 또는 두 자리나 세 자리 일때를 각각 처리해야 한다.

```
;-------------------------------------------------------------------------------
----------
;       VALUES : ROW, COLUMN AND MINE
;-------------------------------------------------------------------------------
----------
MULFACT         DW      1       ;MULTIFLICATION FACTOR FOR CHANGING FROM ASCII TO
BINARY

MINROW EQU      5               ;MINIMUM OF THE VALUE OF ROW
MAXROWEQU       20              ;MAXIMUM OF THE VALUE OF ROW
BINROW DW       ?               ;ACTUAL BINARY NUMBER IN USING
;ASCROW         DB      2 DUP(' '), '$'     ;FOR DEBUGGING THE VALUE

MINCOL  EQU     5               ;MINIMUM OF THE VALUE OF COLUMN
MAXCOL EQU      20              ;MAXIMUM OF THE VALUE OF COLUMN
BINCOL  DW      ?               ;ACTUAL BINARY NUMBER IN USING
;ASCCOLDB       2 DUP(' '), '$'     ;FOR DEBUGGING THE VALUE

MINMINE EQU     1               ;MINIMUM OF THE VALUE OF MINE
MAXMINE         DW      ?               ;MAXIMUM OF THE VALUE OF MINE
;ASCMAXM        DB      3 DUP(' '), '$'     ;FOR DEBUGGING THE VALUE
BINMINE DW      ?               ;ACTUAL BINARY NUMBER IN USING
;ASCMINE        DB      2 DUP(' '), '$'     ;FOR DEBUGGING THE VALUE

ROWLISTLABEL    BYTE            ;ROW PARAMETER LIST
RMAXLEN         DB      3               ;MAXIMUM LENGTH OF VALUE OF ROW
RACTLEN         DB      ?               ;NO. OF CHARACTERS ENTERED
```

```
KBROW  DB      3 DUP(' ')      ;ENTERED VALUE OF ROW

COLLIST LABEL  BYTE            ;COLUMN PARAMETER LIST
CMAXLEN        DB      3               ;MAXIMUM LENGTH OF VALUE OF COLUMN
CACTLEN        DB      ?               ;NO. OF CHARACTERS ENTERED
KBCOL  DB      3 DUP(' ')      ;ENTERED VALUE OF COLUMN

MINELISTLABEL  BYTE    ;MINE PARAMETER LIST
MMAXLEN        DB      4               ;MAXIMUM LENGTH OF VALUE OF MINE
MACTLEN        DB      ?               ;NO. OF CHARACTERS ENTERED
KBMINE  DB     4 DUP(' ')      ;ENTERED VALUE OF MINE
;-------------------------------------------------------------------------
-----
```

[R10RINPUT] 첫 번째로 row 값을 입력

- 위치시킬 커서의 값을 Q10CURSOR에 넘겨 하여 커서를 위치시킨다. (인터럽트 사용)
- INT 21H - 0AH 인터럽트를 이용해서 키보드로부터 입력을 받는다.
- ASCII가 입력되었으므로 일단 입력 값을 binary로 변환한 후,
- 이 값이 제한 범위(5<=row<=20)를 벗어나는지 만족하는지를 체크한다.
- 범위에 만족되면 이를 BINROW라는 곳에 바인딩하고,
- 만약 범위를 벗어났다면 소리를 발생하는 RINGBELL 프로시저를 호출하여 경고음을 내고, 경고 메시지를 보여준 후, 아무키나 받도록 하여 메시지를 없애고(CALL D10DELERR),
- 다시 이 프로시저의 처음부분으로 돌아와 적절한 입력을 재시도 한다.

여기에서의 구현은 대부분이 인터럽트를 이용한 구현이다.

```
;-------------------------------------------------------------------------
-------
;       1) ROW INPUT
;       : INPUT THE VALUE FROM KEYBOARD AND CONVERT IT INTO BINARY FORMAT
;       ACCORDING TO LIMIT VALUES
;-------------------------------------------------------------------------
-------
R10RINPUT      PROC    NEAR
        PUSHA
R20:
        MOV     BINROW, 0               ;INITIALIZE BINROW & MULTIFLICATION FACT
        MOV     MULFACT, 1
        MOV     DH, 13          ;SET CURSOR...row , column
        MOV     DL, 42

        CALL    Q10CURSOR
        MOV     AH, 0AH         ;REQUEST KEYBOARD INPUT - ROW
```

```
        LEA     DX, ROWLIST
        INT     21H

        CMP     RACTLEN, 00     ;IF NOTHING ENTERED, TRY AGAIN
        JE      R60
        CMP     RACTLEN, 01     ;IF THERE IS ONE CHARACTER,
        JE      R35             ;GO R35-PROCESSING ONE DIGIT VALUE

        MOVZX   CX, RACTLEN     ;ASCII -> BINARY(THE CASE OF 2 DIGITS VALUE)
        LEA     SI, KBROW+1     ;PROCESS ONE BY ONE CHARACTER FROM RIGHT TO
LEFT

R30:
        MOV     AL, [SI]
        AND     AX, 000FH
        MUL     MULFACT
        ADD     BINROW, AX
        MOV     AX, MULFACT
        IMUL    AX, 10
        MOV     MULFACT, AX
        DEC     SI
        LOOP    R30
        JMP     RCMP            ;AND COMPARE THE VALUE WITH LIMIT VALUE

R35:
        LEA     SI, KBROW               ;ASCII -> BINARY(THE CASE OF 1 DIGIT VALUE)
        MOV     AL, [SI]        ;PROCESS ONE CHARACTER
        AND     AX, 000FH
        MUL     MULFACT
        ADD     BINROW, AX
        MOV     AX, MULFACT
        IMUL    AX, 10
        MOV     MULFACT, AX
        JMP     RCMP            ;AND COMPARE THE VALUE WITH LIMIT VALUE

RCMP:
        CMP     BINROW, MAXROW          ;IF ACTUAL VALUE OF THE ROW IS OVER OR
BELOW THE LIMIT
        JA      R60             ;GO R60-SHOW ERROR MESSAGE AND TRY AGAIN
        CMP     BINROW, MINROW
        JB      R60
        JMP     R70             ;THE ACTUAL VALUE OF THE ROW IS VALID -> END OF
PROCEDURE

R60:
        CALL    RINGBELL
```

```
        MOV     AX, 1301H               ;INVALID, PRINT ROW ERROR MESSAGE
        MOV     BX, ERRAT               ;ATTRIBUTE OF ERROR MESSAGE-RED ON WHITE
        LEA     BP, ROWERR
        MOV     CX, 33
        MOV     DH, TOPERR
        MOV     DL, ERRLEF

R62:
        INT     10H
        ADD     BP, 33
        INC     DH
        CMP     DH, BOTERR+1
        JNE     R62
        MOV     AH, 10H         ;IF YOU PRESS ANY KEY, ERROR MESSAGE DISAPPEAR
        INT     16H
        CALL    D10DELERR
        JMP     R20             ;RETRYING TO INPUT-ROW

R70:
        POPA
        RET
R10RINPUT       ENDP
;-------------------------------------------------------------------------------
```

[C10CINPUT] 다음으로 COLUMN 값을 입력

- 위치시킬 커서의 값을 Q10CURSOR에 넘겨 호출하여 커서를 위치시킨다. (인터럽트 사용)
- INT 21H - 0AH 인터럽트를 이용해서 키보드로부터 입력을 받는다.
- ASCII가 입력되었으므로 일단 입력 값을 binary로 변환한 후,
- 이 값이 제한 범위(5<=col<=20)를 벗어나는지 만족하는지를 체크한다.
- 범위에 만족되면 이를 BINCOL라는 곳에 바인딩하고,
- 만약 범위를 벗어났다면 소리를 발생하는 RINGBELL 프로시저를 호출하여 경고음을 내고, 경고 메시지를 보여준 후, 아무키나 받도록 하여 메시지를 없애고(CALL D10DELERR),
- 다시 이 프로시저의 처음부분으로 돌아와 적절한 입력을 재시도 한다.

여기에서의 구현은 대부분이 인터럽트를 이용한 구현이다.

```
;-------------------------------------------------------------------------------
;       2) COL INPUT
;       : INPUT THE VALUE FROM KEYBOARD AND CONVERT IT INTO BINARY FORMAT
;       ACCORDING TO LIMIT VALUES
```

```
;-------------------------------------------------------------------------
-------
C10CINPUT       PROC    NEAR
        PUSHA
C20:
        MOV     BINCOL, 0               ;INITIALIZE BINROW & MULTIFLICATION FACT
        MOV     MULFACT, 1
        MOV     DH, 14          ;SET CURSOR...row , column
        MOV     DL, 42

        CALL    Q10CURSOR
        MOV     AH, 0AH         ;REQUEST KEYBOARD INPUT - COL
        LEA     DX, COLLIST
        INT     21H

        CMP     CACTLEN, 00     ;IF NOTHING ENTERED, TRY AGAIN
        JE      C60
        CMP     CACTLEN, 01     ;IF THERE IS ONE CHARACTER,
        JE      C35             ;GO C35-PROCESSING ONE DIGIT VALUE

        MOVZX   CX, CACTLEN     ;ASCII -> BINARY(THE CASE OF 2 DIGITS VALUE)
        LEA     SI, KBCOL+1     ;PROCESS ONE BY ONE CHARACTER FROM RIGHT TO
LEFT
C30:
        MOV     AL, [SI]
        AND     AX, 000FH
        MUL     MULFACT
        ADD     BINCOL, AX
        MOV     AX, MULFACT
        IMUL    AX, 10
        MOV     MULFACT, AX
        DEC     SI
        LOOP    C30
        JMP     CCMP            ;AND COMPARE THE VALUE WITH LIMIT VALUE

C35:
        LEA     SI, KBCOL               ;ASCII -> BINARY(THE CASE OF 1 DIGIT VALUE)
        MOV     AL, [SI]        ;PROCESS ONE CHARACTER
        AND     AX, 000FH
        MUL     MULFACT
        ADD     BINCOL, AX
        MOV     AX, MULFACT
        IMUL    AX, 10
        MOV     MULFACT, AX
        JMP     CCMP            ;AND COMPARE THE VALUE WITH LIMIT VALUE
```

```
CCMP:
        CMP     BINCOL, MAXCOL ;IF ACTUAL VALUE OF THE COL IS OVER OR BELOW THE
LIMIT
        JA      C60             ;GO C60-SHOW ERROR MESSAGE AND TRY AGAIN
        CMP     BINCOL, MINCOL
        JB      C60
        JMP     C70             ;THE ACTUAL VALUE OF THE COL IS VALID -> END OF
PROCEDURE

C60:
        CALL    RINGBELL
        MOV     AX, 1301H       ;INVALID, PRINT COLUMN ERROR MESSAGE
        MOV     BX, ERRAT       ;ATTRIBUTE OF ERROR MESSAGE-RED ON WHITE
        LEA     BP, COLERR
        MOV     CX, 33
        MOV     DH, TOPERR
        MOV     DL, ERRLEF

C62:
        INT     10H
        ADD     BP, 33
        INC     DH
        CMP     DH, BOTERR+1
        JNE     C62
        MOV     AH, 10H         ;IF YOU PRESS ANY KEY, ERROR MESSAGE DISAPPEAR
        INT     16H
        CALL    D10DELERR
        JMP     C20             ;RETRYING TO INPUT-COL

C70:
        POPA
        RET
C10CINPUT       ENDP
;-------------------------------------------------------------------------
------
```

[G10MAXMINE] 이 프로시저는 현재까지 받은 ROW와 COL값을 연산하여 MINE값의 한계값을 정한다. 이 때 BINROW와 BINCOL을 이용하여 연산하며 결과값은 MAXMINE에 들어간다.

```
;-------------------------------------------------------------------------
------
;       [GET MAXIMUM VALUE OF MINE]
;       : ROW*COL/2
;-------------------------------------------------------------------------
```

```
------
G10MAXMINE      PROC    NEAR
        PUSHA
        MOV     AX, BINROW      ; MAXMINE = BINROW*BINCOL/2
        MOV     DX, BINCOL
        MUL     DL
        SHR     AX, 01
        MOV     MAXMINE, AX
        POPA
        RET
G10MAXMINE      ENDP
;-------------------------------------------------------------------------
-------
```

[M10MINPUT] MINE 값 입력

- 위치시킬 커서의 값을 Q10CURSOR에 넘겨 호출하여 커서를 위치시킨다. (인터럽트 사용)

- INT 21H - 0AH 인터럽트를 이용해서 키보드로부터 입력을 받는다.

- ASCII가 입력되었으므로 일단 입력 값을 binary로 변환한 후,

- 이 값이 제한 범위(1<=MINE<=MAXMINE)를 벗어나는지 만족하는지를 체크한다.

- 범위에 만족되면 이를 BINMINE라는 곳에 바인딩하고,

- 만약 범위를 벗어났다면 소리를 발생하는 RINGBELL 프로시저를 호출하여 경고음을 내고, 경고 메시지를 보여준 후, 아무키나 받도록 하여 메시지를 없애고(CALL D10DELERR),

- 다시 이 프로시저의 처음부분으로 돌아와 적절한 입력을 재시도 한다.

여기에서의 구현은 대부분이 인터럽트를 이용한 구현이다.

```
;-------------------------------------------------------------------------
-------
;       3) MINE INPUT
;       : INPUT THE VALUE FROM KEYBOARD AND CONVERT IT INTO BINARY FORMAT
;       ACCORDING TO LIMIT VALUES
;-------------------------------------------------------------------------
-------
M10MINPUT       PROC    NEAR
        PUSHA
M20:
        MOV     BINMINE, 0      ;INITIALIZE BINROW & MULTIFLICATION FACT
        MOV     MULFACT, 1
        MOV     DH, 15          ;SET CURSOR...row , column
        MOV     DL, 42

        CALL    Q10CURSOR
```

```
        MOV     AH, 0AH         ;REQUEST KEYBOARD INPUT - MINE
        LEA     DX, MINELIST
        INT     21H

        CMP     MACTLEN, 00     ;IF NOTHING ENTERED, TRY AGAIN
        JE      M60
        CMP     MACTLEN, 01     ;IF THERE IS ONE CHARACTER,
        JE      M35             ;GO M35-PROCESS ONE CHARACTER
        CMP     MACTLEN, 02     ;IF THERE ARE TWO CHARACTERS,
        JE      M36             ;GO M36-PROCESS TWO CHARACTERS

        MOVZX   CX, MACTLEN     ;ASCII -> BINARY(THE CASE OF 3 DIGITS VALUE)
        LEA     SI, KBMINE+2    ;PROCESS ONE BY ONE CHARACTER FROM RIGHT TO
LEFT

M30:
        MOV     AL, [SI]
        AND     AX, 000FH
        MUL     MULFACT
        ADD     BINMINE, AX
        MOV     AX, MULFACT
        IMUL    AX, 10
        MOV     MULFACT, AX
        DEC     SI
        LOOP    M30
        JMP     MCMP            ;AND COMPARE THE VALUE WITH LIMIT VALUE

M35:
        LEA     SI, KBMINE      ;ASCII -> BINARY(THE CASE OF 1 DIGIT VALUE)
        MOV     AL, [SI] ;PROCESS ONE CHARACTER
        AND     AX, 000FH
        MUL     MULFACT
        ADD     BINMINE, AX
        MOV     AX, MULFACT
        IMUL    AX, 10
        MOV     MULFACT, AX
        JMP     MCMP            ;AND COMPARE THE VALUE WITH LIMIT VALUE

M36:
        MOVZX   CX, MACTLEN     ;ASCII -> BINARY(THE CASE OF 2 DIGITS VALUE)
        LEA     SI, KBMINE+1    ;PROCESS ONE BY ONE CHARACTER FROM RIGHT TO
LEFT

M37:
        MOV     AL, [SI]
        AND     AX, 000FH
```

```
        MUL     MULFACT
        ADD     BINMINE, AX
        MOV     AX, MULFACT
        IMUL    AX, 10
        MOV     MULFACT, AX
        DEC     SI
        LOOP    M37
        JMP     MCMP            ;AND COMPARE THE VALUE WITH LIMIT VALUE

MCMP:
        MOV     AX, BINMINE
        MOV     BX, MAXMINE
        CMP     AX, BX          ;IF ACTUAL VALUE OF THE MINE IS OVER OR BELOW THE
LIMIT
        JA      M60             ;GO M60-SHOW ERROR MESSAGE AND TRY AGAIN
        CMP     AX, MINMINE
        JB      M60
        JMP     M70             ;THE ACTUAL VALUE OF THE MINE IS VALID -> END OF
PROCEDURE

M60:
        CALL    RINGBELL
        MOV     AX, 1301H               ;INVALID, PRINT MINE ERROR MESSAGE
        MOV     BX, ERRAT               ;ATTRIBUTE OF ERROR MESSAGE-RED ON WHITE
        LEA     BP, MINEERR
        MOV     CX, 33
        MOV     DH, TOPERR
        MOV     DL, ERRLEF

M62:
        INT     10H
        ADD     BP, 33
        INC     DH
        CMP     DH, BOTERR+1
        JNE     M62
        MOV     AH, 10H         ;IF YOU PRESS ANY KEY, ERROR MESSAGE DISAPPEAR
        INT     16H
        CALL    D10DELERR
        JMP     M20             ;RETRYING TO INPUT-MINE

M70:
        POPA
        RET
M10MINPUT       ENDP
;-------------------------------------------------------------------------
-------
```

여기까지의 과정에서, BINROW, BINCOL, 그리고 BINMINE의 값들이 모두 결정되었다. 이들은 모두 워드단위의 변수들이며 다시 입력받는 경우에는 다시 입력받은 대로 값이 변화하도록한다.
그리고 실제 게임 셋팅으로 들어가면 연산을 용이하게 하기위해서 BINROW와 BINCOL 은 각각 바이트 단위의 ROW, COL로 전환하여 사용한다.

입력에 대한 ERROR가 발생했을 때 호출되는 RIGNBELL 프로시저와 D10DELERR 프로시저를 잠시 살펴보면 다음과 같다.

[RINGBELL] 이 프로시저는 "BELL"이 참조하는 문자열에 벨의 16진 값을 넣고 인터럽트(INT 21H - 09H)를 거는 방법으로 벨소리가 출력되도록 한다. 어떤 입력, 출력 값도 없는 프로시저이다.

```
;-------------------------------------------------------------------------------
----------------------
;       [RINGBELL]
;       : BELL WHEN IT OCCUR AN ERROR
;-------------------------------------------------------------------------------
----------------------
RINGBELL        PROC    NEAR
        PUSHA
        MOV     AH, 09H
        LEA     DX, BELL
        INT     21H
        POPA
        RET
RINGBELL        ENDP
;-------------------------------------------------------------------------------
----------------------
```

[D10DELERR] 출력된 경고 메시지를 제거하기 위한 프로시저로, 입력에 대한 경고 메시지는 늘 같은 위치에 출력되므로, 그 위치에 검정색 바탕 검정색 글씨로 SHADOW를 출력하는 방법으로 한다. 경고 메시지가 반복적인 인터럽트(INT 10H - 13H)를 통해 출력되듯 여기에서도 마찬가지로 반복적인 인터럽트로 수행한다.

실제 경고메시지에 대한 데이터 그룹은 다음과 같으며 아래쪽의 DELERR는 이 메시지를 COVER하는 SHADOW임을 알 수 있다.

```
;-------------------------------------------------------------------------------
-----
```

```
;       ERROR MESSAGE ABOUT INPUT DATA
;       : ABOUT THE VALUES OF THE ROW, COLUMN AND MINE
;-------------------------------------------------------------------------
-----
TOPERR EQU     03              ;TOP OF THE RANGE OF ROW
BOTERR EQU     07              ;BOTTOM OF THE RANGE OF ROW
ERRLEF EQU     24              ;LEFT OF THE RANGE OF COLUMN
ERRAT  DW      0074H           ;ERROR MESSAGE ATTRIBUTE - RED ON WHITE
ROWERR DB      0DAH, 31 DUP(0C4H), 0BFH
       DB      0B3H, ' You cannot input the value!   ', 0B3H
       DB      0B3H, ' Possible that, 4 < ROW < 21.  ', 0B3H
       DB      0B3H, ' Press any key, and try again. ', 0B3H
       DB      0C0H, 31 DUP(0C4H), 0D9H
COLERR DB      0DAH, 31 DUP(0C4H), 0BFH
       DB      0B3H, ' You cannot input the value!   ', 0B3H
       DB      0B3H, ' Possible that, 4 < COL < 21.  ', 0B3H
       DB      0B3H, ' Press any key, and try again. ', 0B3H
       DB      0C0H, 31 DUP(0C4H), 0D9H
MINEERRDB      0DAH, 31 DUP(0C4H), 0BFH
       DB      0B3H, ' You cannot input the value!   ', 0B3H
       DB      0B3H, ' Possible 0 < MINE < ROW*COL/2.', 0B3H
       DB      0B3H, ' Press any key, and try again. ', 0B3H
       DB      0C0H, 31 DUP(0C4H), 0D9H

ERDLAT DW      0000H           ;DELETE (ERROR MESSAGE) ATTRIBUTE - BLACK ON
BLACK
DELERR DB      33 DUP(0DBH)    ;DELETE BY SHADOW
;-------------------------------------------------------------------------
----------
```

여기까지 게임의 난이도를 정하기 위한 데이터들을 모두 입력받고 실제 계산에 이용될 수 있는 값들도 모두 저장이 되었다. 앞으로 남은 문제는 키보드로 이 게임을 진행할 것인지 아니면 마우스로 게임을 진행할 것인지에 대한 결정이다.

[MODEMSG] 이 프로시저가 사용자로 하여금 게임을 어떤 입력 모드로 진행할 것인지를 묻는다. 마찬가지로 반복적인 인터럽트(INT 10H - 13H)로 구현한다.

```
;-------------------------------------------------------------------------
-------
;       [SELECT MODE MESSAGE]
;       : DISPLAY SHADOW BOX, AND NEXT MESSAGE
;-------------------------------------------------------------------------
-------
MODEMSG        PROC    NEAR
       PUSHA
```

```
        MOV     AX, 1301H               ;DISPLAY SHADOW AND ADVENCE CURSOR
        MOV     BX, SHDAT               ;PAGE 0, GRAY ON BLACK(GRAY SHADOW)
        LEA     BP, MODESH
        MOV     CX, 29
        MOV     DH, MODETOP+1
        MOV     DL, MODELEF+1

MOD20:
        INT     10H
        INC     DH
        CMP     DH, MODEBOT+2
        JNE     MOD20

        MOV     AX, 1301H               ;DISPLAY MESSAGE AND ADVENCE CURSOR
        MOV     BX, MSGAT               ;PAGE 0, BLACK ON WHITE
        LEA     BP, MODE
        MOV     CX, 29
        MOV     DH, MODETOP
        MOV     DL, MODELEF

MOD30:
        INT     10H
        ADD     BP, 29
        INC     DH
        CMP     DH, MODEBOT+1
        JNE     MOD30

        POPA
        RET
MODEMSG         ENDP
;-------------------------------------------------------------------------
-------
```

여기까지 게임을
- 초기화하는 B10INIT 프로시저
- 게임에 필요한 조건(난이도)를 입력받은 후 모드를 결정하도록 하는 메시지 출력까지 보여주는 C10INPUT 프로시저

를 마친다.

이제 다시 MAIN 프로시저로 돌아가서
- 키보드로부터 모드 선택에 대한 입력(“k" or "K", "m" or "M")을 받아
- inputmode라는 변수에 그 값을 저장하고,
- 워드 단위의 BINROW는 바이트 단위의 ROW에,
- 워드 단위의 BINCOL은 바이트 단위의 COL에 저장,

- BINMINE은 MINE에 저장하고,

이제부터는 ROW와 COL만큼의 게임 필드를 출력하는 프로시저 구현에 대해 자세히 설명한다.

..... 3. D10SET MODULE .....

[d10set]module
=> [setcell], [background], [showfield] 등의 함수로 구성된 이 모듈은 게임을 시작하기 전에 배경화면과 지뢰밭 등의 사용자 interface를 제공하고 내부적으로 미리 지뢰를 설치하는 부분이다.

#1. [setcell]
=> 내부적으로 지뢰의 정보를 만드는 함수이다.

- 기본적으로 지뢰밭에서 한 개의 cell의 정보에는
-지뢰가 있는가 여부와
-지뢰가 없다면 주위의 cell들에 몇 개의 지뢰가 있는가(0~8)
-그리고 사용자가 지뢰가 있을 것이라 예상해서 표시를 하였는가
를 가지고 있어야 한다.
그러므로 한 byte로 충분히 나타낼 수 있다.
```
cinfo    db      600 dup('0')            ; 600개로 충분하게 선언하였다.
```
이런 식으로 각각의 cell 정보를 가지고 있는 string을 선언하였다.

구현 시에는,
-지뢰가 있다면 40h (ASCII '@')
-지뢰가 없다면 주위의 지뢰개수는 30h~38h (ASCII '0'~'8')
-사용자가 지뢰예상표시를 했다면 기존의 정보에 +20h
즉, 지뢰가 있는 곳에 지뢰예상표시를 했다면 40h+20h=60h('P')
지뢰가 없는 곳에 표시를 했다면 50h~58h
의 정보를 가지고 있도록 구현하였다. 이런 식으로 정보를 가지고 있게 한 이유는 구현할 때에 디버그를 편하게 하기 위함이었다. 30h에서 38h는 ASCII로 표현하였을 때에도 '0'~'8'로 나타난다. 지뢰를 가지고 있는 표시인 40h는 ASCII로 표현하면 '@'으로 나타난다. 그러므로 debugger("debug.exe")를 통해서 내부메모리를 참조하면서 debug할 필요 없이 cell의 정보들을 모두 출력하면 제대로 실행되고 있는지를 직관적으로 관찰할 수 있다.

#1.1 [random]
=>이 함수를 이용하여 무작위로 지뢰 심는다.

-입력받아서 저장해놓은 mine이라는 data를 보고 mine의 수만큼 cinfo에 저장한다.

-[random]이라는 함수는 책에 나와 있는 random number generator를 이용하였는데 이는 현재의 시간을 seed value로 사용하여 register BL에 random한 수를 만든다.
-그런데 짧은 시간 안에 여러 번 [random]을 호출하기 때문에 seed값이 비슷하게 나오는 경향이 있고 결과적으로 지뢰밭에 지뢰가 골고루 퍼지지 않고 한 곳에 뭉치는 현상을 발견할 수 있었다.
-이런 현상을 방지하기 위해 [random]을 호출하는 사이에 delay를 주는 방법을 사용하여 보았는데, 이 경우에서는 100개의 지뢰를 뿌리기 위해서는 적지 않은 시간이 낭비가 되었다.
-그래서 사용한 방법이 [random]함수 자체에서 random number를 2개 발생시켜서 2개의 random number를 더한 수를 register BL에 return하는 방식으로 고쳤더니 비교적 골고루 지뢰가 퍼지는 결과를 얻을 수 있었다.

```
;;-------------------------source code-------------------------------------
        mov     ax,0            ; Interval timer
        out     43h,al          ; via port 43H
        in      al,40h          ; Make 2 accesses to
        in      al,40h          ;       port 40H
        mov     bl,al

        mov     ax,0            ; Interval timer
        out     43h,al          ; via port 43H
        in      al,40h          ; Make 2 accesses to
        in      al,40h          ;       port 40H
        add     al,bl           ; BL에 랜덤한 값이 들어간다.
;;-------------------------------------------------------------------------
----
```

#1.2 [count]

=> 지뢰가 없는 모든 cell에 대하여 주위의 지뢰개수 count하기

-[count]함수는 하나의 cell에 대해서 count를 하도록 하였다.
-이때에 신경을 써야했던 것이, 모든 cell이 주위에 8개의 cell을 가지지 않는다는 것이었다. 제일 좌측, 우측, 상단, 하단의 cell은 주위에 5개의 cell만 존재하고 4개의 귀퉁이에 있는 cell은 주위에 3개의 cell밖에 존재하지 않게 된다.
-이런 예외적인 상황에 대하여 처리하기 위해서 한 cell이 주어지면 그 cell의 좌우상하에 다른 cell이 존재하는지 check한 후에 그 cell의 정보가 '@'인지를 파악하도록 하였다.

-모든 cell의 정보를 가지고 있는 cinfo string은 1차원으로 이루어졌기 때문에 이것을 이용하여 2차원과 같이 사용하기 위해서는 적절한 처리가 필요하다.
예를 들어, 10*10의 지뢰밭이 있다고 하자. cinfo[current]의 정보에 대해서
        좌측의 정보는 cinfo[current-1]를,
        우측의 정보는 cinfo[current+1]를,

상단의 정보는 cinfo[current-10-1],cinfo[current-10],cinfo[current-10+1]를,
하단의 정보는 cinfo[current+10-1],cinfo[current+10],cinfo[current+10+1]를
참조하여서 지뢰가 있는지 파악해야 한다.

#2. [background]
=> 지뢰밭 주위에 보기 좋게 테두리를 그리는 작업을 수행한다.

-이중 loop를 이용하여서 cx,dx의 값을 증가시키면서 pixel을 뿌리도록 하였다.

#3. [showfield]
=> 실제적으로 게임이 진행이 될 지뢰밭을 뿌리는 작업을 수행한다.

-이 procedure는 밭이 뿌려질 시작점의 cx,dx에서부터 끝나는 점의 cx,dx까지 증가시키면서 [drawpix]를 호출한다.

여기에서 시작점(STARTROW, STARTCOL)은 다음과 같은 프로시저를 통해 구해진다.
- GETSTARTROW : (10-ROW/2)*20을 연산하여 STARTROW에 저장,
- GETSTARTCOL : (15-COL/2)*20을 연산하여 STARTCOL에 저장,
이 두 변수는 지뢰밭을 화면의 가운데 출력하기 위한 연산들이며 게임이 새로이 진행될 때 언제든지 변할 수 있는 값이다.

```
;;-------------------------source code----------------------------------------
        mov     cx,startcol         ;get the information starting point coordinate
        mov     dx,startrow

showfield20loop:
        call    drawpix
        inc     cx
        mov     al,col
        mul     csize
        add     ax,startcol
        cmp     cx,ax               ;if col reaches the end
        je      showfield30loop
        jmp     showfield20loop

showfield30loop:
        mov     cx,startcol
        inc     dx
        mov     al,row
        mul     csize
        add     ax,startrow
```

```
        cmp     dx,ax           ;if row reaches the end
        je      showfield90
        jmp     showfield20loop
;;-------------------------------------------------------------------------
----
```

#3.1[drawpix]

=>showfield에서 이중루프를 이용하여 drawpix를 호출하여 하나하나의 pixel에 색 뿌린다.

-이 부분을 처음 설계할 때에는, 하나의 cell모양을 저장한 다음에 그 cell을 여러 번 찍어내는 방식으로 전체 지뢰밭을 뿌리려고 하였다. 그런데 이 방식은 다소 복잡한 연산이 필요한 것으로 판단되었고 새로운 방식을 택하기로 하였다.

-색을 뿌리면서 각각의 cell의 구분을 하기 위해서는 다른 색의 경계선을 그려 넣는 것도 하나의 방법이지만 각 cell에 입체감을 주어서 구분할 수 있도록 할 수도 있다.
예를 들어, 하나의 cell이 10*10개의 pixel로 이루어진다고 하자.
이때,

최상단의 pixel들과 제일 좌측의 pixel들을 밝은 색 계열로 처리하고

최하단의 pixel들과 제일 우측의 pixel들을 어두운 색 계열로 처리하면

입체감을 표현할 수 있다.
즉, 좌측상단의 시작점을 (0,0)이라고 하였을 때에

(0,0)~(0,9)과 (0,0)~(9,0)까지는 밝은 색으로 하고

(0,9)~(9,9)과 (9,0)~(9,9)까지는 어두운 색으로 한다.

이것은 나머지 연산을 통해서 구현가능하다.
색을 칠하려는 위치 즉, (cx,dx)가 주어지면 cell의 size로 나누어서 그 나머지를 본 다음에 분기시켜서 색을 구별하면 된다.

```
;;-------------------------source code-----------------------------------------
        mov     btmp,0
        mov     ax,cx           ; to compare col
        div     csize           ; cx/10
        cmp     ah,0
        jbe     drawpix10       ;remainder = 0 (start)
        mov     bl,csize
        mov     btmp,bl
        sub     btmp,1
        cmp     ah,btmp
        ;cmp    ah,9
        jae     drawpix20       ;remainder = 9 (end)
        jmp     drawpix30       ;0<remainder<9 (middle)

drawpix10:
        mov     ax,dx           ; to compare row
```

```
        div     csize           ; dx/10
        mov     bl,csize
        mov     btmp,bl
        sub     btmp,1
        cmp     ah,btmp
        ;cmp    ah,9            ; remainder
        jae     drgray          ; (0,9)
        jmp     drbright        ; (0,0)~(0,8)

drawpix20:
        mov     ax,dx           ; to compare row
        div     csize           ; dx/10
        cmp     ah,0
        jbe     drbright        ; (9,0)
        jmp     drgray          ; (9,1)~(9,9)

drawpix30:
        mov     ax,dx           ; to compare row
        div     csize           ; dx/10
        cmp     ah,0
        jbe     drbright        ; (1,0)~(8,0)
        mov     bl,csize
        mov     btmp,bl
        sub     btmp,1
        cmp     ah,btmp
        ;cmp    ah,9            ;
        jae     drgray          ; (1,9)~(8,9)
        jmp     drwhite         ; (1,1)~(8,8)
;;-------------------------------------------------------------------------
----
```

..... 4. E10GAME MODULE .....

[e10game]module

=>게임의 설정이 끝난 후부터 사용자의 입력을 받아서 게임을 진행하는 부분이다.

-[e50sign], [e60erase], [e70open], [number], [frame], [picture], [curr] 등의 함수들을 호출하면서 전체적으로 게임을 운영해 나간다.

#1. 사용자의 입력방법: 키보드와 마우스

-사용자가 입력할 수 있는 방법은 크게 키보드 입력과 마우스 입력으로 나누었다.
-처음에는 키보드와 마우스를 동시에 사용하여서 게임을 진행하도록 구현하려고 했으나 키

보드입력을 받는 인터럽트는 입력이 주어질 때까지 기다리기 때문에 그 인터럽트에 일단 들어가면 마우스의 입력은 무시가 되는 것을 알 수 있다. time-slicing을 구현하여서 일정 시간동안에는 마우스의 입력을 기다리고 일정 시간동안에는 키보드의 입력을 기다리는 방법으로 하면 두 개의 입력을 동시에 받을 수 있다.
-하지만 이 방법의 구현을 아직 하지 못해서 부득이하게 사용자에게 어떤 입력을 받을지를 결정하게 한 뒤에 한 가지 방법으로만 입력을 받도록 하였다.

#1K. 키보드 입력모드

- 기본적으로 키보드 입력을 기다리는 인터럽트를 이용하여서 게임이 끝날 때까지 while loop을 도는 형식으로 구현하였다.
- 먼저 커서를 제일 왼쪽 위의 셀에 설정한다.
- 키보드로 받아서 처리할 경우는 'm', 'M', 'c', 'C', space bar, 좌우상하 커서(←, →, ↑, ↓) 등 총 9개이다. 이때, int 16h- function ah,10h을 이용하여 keyboard 입력을 받는다.

- 'm'이나 'M'을 입력으로 받으면, label esign으로 가서,
  [e50sign]을 호출하여 지뢰표시를 한다.
=> 지뢰가 있는 모든 cell에 지뢰표시가 되었으면 게임을 이긴 것이므로 게임을 종료한다. (e10win, e20win)

- 'c'이나 'C'를 입력으로 받으면, label eerase으로 가서,
  [e60erase]을 호출하여 지뢰표시를 지운다.

- space bar를 입력으로 받으면, label eopen으로 가서,
  [e70open]을 호출하여 숫자를 2초 동안 보여주다가 지운다.
=> 사용자가 연 cell에 지뢰가 있었다면, 게임을 진 것이므로 모든 셀에 대한 정보를 뿌려주고 게임을 종료한다.

- 좌우상하 커서(←, →, ↑, ↓)을 입력으로 받으면,
  각각 label elf, ert, eup, edn으로 가서, 진행방향의 cell에 커서를 뿌리고, 기존에 있었던 커서는 지운다.
  이때, 현재의 위치가 경계선 상이었는데 다시 나가려고 하면,
  eerror으로 가서, bell을 울린다.

-주어진 입력이 부적절한 경우였다면, label eerror으로 가서, bell을 울린다.

#1K.1 [curr]
=> 현재 사용자가 지정하고 있는 셀의 위치를 알아내는 함수이다.

-키보드 입력모드에서 가장 자주 쓰이는 함수이다. 내부적으로 어느 위치에 있는지 나타내

는 current 변수를 보고 화면 상에 현재위치가 어디인지 register cx,dx에 나타낸다.

```
;;------------------------source code---------------------------------------
        mov     ax,current
        div     col             ; ah:remainder al:quotient
        mov     al,ah
        mov     ah,00h
        mul     csize
        add     ax,startcol
        mov     cx,ax           ;get column pixel

        mov     ax,current
        div     col             ; ah:remainder al:quotient
        mov     ah,00h
        mul     csize
        add     ax,startrow
        mov     dx,ax           ;get row pixel
;;-------------------------------------------------------------------------
----
```

#1M. 마우스 입력모드

- 마우스 모드가 선택되면 키보드 모드에서 현재의 커서 위치를 보여주는 프레임을 보여주지 않고 마우스 포인터를 보여준다. CALL MOUSEINIT
- 마우스는 폴링방법이기 때문에 인터럽트를 사용한 출력을 이용한 현 프로젝트에서는 새로운 이벤트가 일어나기 전에 셀의 상태를 REFRESH해 주는 부분이 필요하다.
- 마우스의 상태를 알아내는 인터럽트(INT 33H - 03H)를 이용해 마우스의 상태를 체크하고
- 왼쪽 클릭이 발생하면 EOPEN을 호출하여 해당 셀을 열고,
- 오른쪽 클릭이 발생하면 먼저 전에 이미 지뢰표시를 했던 셀인지 파악하고, 표시하지 않았던 셀이면 ESIGN을 호출하여 표시하고, 표시했던 셀이면 EERASE을 호출하여 표시를 지우도록 한다.
- EOPEN과 ESIGN, 그리고 EERASE 등에서는 INPUTMODE의 값을 토대로, 마우스 모드인지 파악하는 과정이 필요할 것이며, 마우스 경우 MOUSECURR을 불러 현재의 셀의 해당 셀 정보에 프로시저를 적용한다.
- 이러한 과정을 사용자가 패배하거나 승리할 때까지 반복적으로 수행한다.
- 게임이 끝나면 마우스 포인터를 사라지게 한다. CALL HIDEMOUSEP

#1M.1 [MOUSEINIT]

=> 마우스의 포인터를 인터럽트(INT 33H - 00H)를 걸어 출력한다.

#1M.2 [MOUSECURR]

=> 마우스의 포인터가 있는 곳을 알아낸다.

입력은 CX(VALUE X OF MOUSE POINT)와 DX(VALUE Y OF MOUSE POINT)이며,
반환값은 CURRENT로 아래와 같은 연산을 한다.
CURRENT
(QUOTIENT OF (CX-STARTROW)/20)*COL + (QUOTIENT OF (DX-STARTCOL)/20))

#1M.3 [HIDEMOUSEP]
=> 마우스의 포인터를 인터럽트를(INT 33H -02H)를 걸어 사라지게 한다.

#2. [e50sign]
=> 현재의 셀에 지뢰표시 즉, 사진을 뿌리고 지뢰의 내부정보를 바꾸는 함수이다.

- 'm'이나 'M'을 입력을 받는 경우는 사용자가 현재 커서가 위치해 있는 cell에 지뢰가 있을 것이라 예상하여서 표시하는 것이다. 이 경우에는 [e50sign] procedure가 호출된다.
- 이 함수는 현재의 셀 정보를 얻어오고,
    1. 지정된 셀에 이미 지뢰 표시가 되어있었다면, 그냥 넘어간다.
    2. 아무 표시가 없는 경우에,
        2.1 지뢰가 실제 있는 곳에 지뢰표시를 한 것이었다면, 승리점수++
        2.2 지뢰가 없는 곳에 잘못 표시한 것이라면, 승리점수--
    2.1,2.2 경우 모두 [picture]을 호출하여 지정된 셀에 사진을 뿌린다.

```
;;------------------------source code----------------------------------------
        :change the cell info
        mov     bx,current
        cmp     cinfo[bx],'@'           ;이미 셀에 지뢰표시가 있었다면,
        ja      e50ret                  ;아무 변화를 주지 않고 return한다.

        add     cinfo[bx],20h           ;1.mine없는 cell :0xh => 2xh  2.mine:10h=>30h
        cmp     cinfo[bx],60h           ;지뢰가 있는 곳에 지뢰표시를 한 것이라면,
        je      e50true                 ;승리점수++
        jmp     e50display
e50true:
        inc     win                     ;if false, 승리점수--
e50display:
        call    curr                    ;현재위치를 받는다.
        call    picture                 ;그림을 뿌린다.
        inc     checked                 ;지뢰표시의 개수++
;;-------------------------------------------------------------------------
----
```

#2.1 [picture]
=> 현재의 셀에 사진을 뿌리는 함수이다.

- 이 부분을 처음 구현할 때의 계획은 10*10(pixel)의 256 color bmp file읽어 들여서 그것을 화면에 뿌리는 모듈로 만드는 것이었다.
- 하지만 bmp file의 형식을 제대로 이해하지 못하여서, 수작업으로 색깔 정보를 data segment에 저장한 후에, 차례대로 뿌리는 방식을 택하였다. 마찬가지로 2중루프를 돌리도록 하였다. 256 color의 정보를 제대로 알지 못하여서 16 color로 20*20의 정보를 만들었다.

다음은 실제 이용된 비트맵 파일과 그 구현방법이다.

```
;;-------------------------source code---------------------------------------
yms     DB      06H,06H,0EH,0EH,06H,06H,08H,00H,00H,00H, 00H,08H,06H,00H,00H,06H,0EH,06H,06H,0EH
        DB      06H,0EH,0EH,06H,06H,05H,08H,06H,00H,00H, 00H,06H,05H,00H,00H,05H,06H,05H,06H,06H
        DB      06H,0EH,06H,06H,05H,08H,00H,00H,08H,00H, 00H,00H,08H,00H,08H,06H,05H,06H,06H,06H
        DB      05H,06H,06H,08H,00H,08H,08H,05H,00H,06H, 0EH,0FH,0EH,05H,08H,00H,00H,05H,06H,06H
        DB      0FH,0EH,06H,08H,00H,08H,06H,00H,06H,0EH, 0EH,0EH,0EH,06H,05H,06H,00H,06H,05H,00H
        DB      0EH,0EH,05H,07H,00H,08H,00H,06H,0FH,0EH, 0EH,07H,0EH,0EH,06H,08H,00H,06H,00H,00H
        DB      06H,0EH,00H,00H,00H,08H,06H,0EH,0FH,07H, 0EH,0EH,0EH,07H,06H,05H,05H,06H,00H,00H
        DB      06H,06H,08H,05H,00H,08H,05H,08H,07H,0EH, 0EH,0EH,0EH,06H,06H,00H,05H,06H,05H,06H
        DB      06H,06H,06H,00H,08H,05H,0CH,0FH,0EH,07H, 0EH,0EH,0CH,0EH,0EH,06H,05H,08H,05H,06H
        DB      00H,05H,08H,00H,08H,05H,06H,00H,08H,0EH, 0FH,0EH,07H,00H,06H,06H,08H,05H,00H,00H

        DB      05H,05H,06H,00H,08H,06H,0EH,0EH,0CH,0EH, 0FH,06H,0EH,0EH,0CH,06H,05H,00H,05H,00H
        DB      05H,06H,08H,00H,08H,06H,0FH,0CH,0EH,07H, 0FH,06H,0EH,0CH,07H,06H,05H,05H,06H,00H
        DB      00H,06H,05H,00H,05H,06H,0FH,0EH,0EH,0CH, 0FH,06H,06H,0EH,0CH,06H,05H,06H,00H,00H
        DB      06H,00H,00H,08H,05H,06H,0FH,0EH,07H,0EH, 0CH,06H,06H,0CH,0EH,06H,05H,06H,08H,00H
        DB      00H,05H,00H,05H,05H,06H,0EH,07H,0EH,0FH, 0CH,0EH,06H,07H,0CH,00H,06H,05H,05H,00H
        DB      06H,05H,06H,05H,00H,05H,0EH,0EH,0DH,0DH, 0CH,04H,06H,0EH,07H,00H,06H,05H,08H,00H
        DB      05H,06H,0CH,05H,00H,05H,06H,0EH,0FH,0EH, 0CH,0DH,0EH,00H,00H,05H,06H,06H,00H,00H
        DB      0EH,0EH,0EH,0EH,05H,08H,00H,0EH,07H,0FH, 0EH,0EH,0EH,06H,00H,00H,08H,05H,06H,00H
        DB      0EH,07H,0EH,0CH,0EH,00H,00H,05H,0CH,0FH, 07H,0EH,06H,00H,00H,00H,06H,08H,06H,00H
        DB      0EH,0CH,0EH,0EH,0DH,06H,00H,06H,06H,0EH, 06H,00H,00H,00H,05H,00H,05H,08H,00H,00H

;-------------------------------------------------------------------------------
-----
        call    curr            ;get the current cx,dx
        mov     ax,20
        mov     bx,20
        mov     wtmp,ax
        mov     wtmp2,bx
        add     wtmp,cx
        add     wtmp2,dx

        lea     si,yms          ;picture color information
```

```
picture20:
        mov     ah,0ch          ;function
        mov     al, [si]        ;color
        mov     bh,0            ;page 0
        int     10h             ;
        inc     cx
        inc     si
        cmp     wtmp,cx
        je      picture30
        jmp     picture20

picture30:
        sub     cx,20
        inc     dx
        cmp     wtmp2,dx
        je      picture90
        jmp     picture20
;;-------------------------------------------------------------------------
----
```

#3. [e60erase]

=> ‘c’나 ‘C’를 입력받으면 현재의 cell에 지뢰표시를 지우는 함수이다.

- 지뢰표시가 없는 곳이었다면, 아무 변화 없이 return한다.
- 지뢰표시가 있었다면,
  - -해당 cell의 내부 정보를 바꾼다. 20h만큼 감소시킨다.
  - -[recover] 함수를 호출하여 지뢰표시를 지운다.

```
;;-------------------------source code-------------------------------------
        ;change the cell info
        mov     bx,current
        cmp     cinfo[bx],50h           ;현재의 cell에 지뢰표시가 없었다면,
        jb      e60ret                  ;그냥 return

        sub     cinfo[bx],20h           ;현재의 cell에 지뢰표시가 있었다면, 지운다.
        cmp     cinfo[bx],'@'           ;잘못된 지뢰표시를 지운것이라면,
        je      e60false                ;승리점수++
        jmp     e60display
e60false:
        dec     win
e60display:
        call    curr                    ;현재의 지뢰에 지뢰표시를 지운다.
        call    recover

        call    frame                   ;현재 위치에 커서를 뿌린다.
```

```
        dec     checked
;;-------------------------------------------------------------------------
----
```

#4. [e70open]

=> space bar를 만나서, 현재의 cell을 open하여 숫자나 지뢰를 뿌리는 함수이다.

-[number] 함수를 불러서 0~8의 숫자 또는 지뢰를 현재 cell에 뿌린다.
-[delay] 함수를 불러서 2초간의 시간을 잰다.
-[recover] 함수를 불러서 숫자를 지운다.

```
;;-------------------------source code-------------------------------------
        mov     bx,current
        cmp     cinfo[bx],'@'           ;현재의 cell에 지뢰가 있었다면,
        je      e70lose                 ;게임에서 진 것이다.

        call    curr                    ;현재의 cell에 지뢰가 없었다면,
        call    number                  ;주위의 지뢰 개수를 뿌려준다.

        call    delay                   ;2초의 delay를 준다.
        call    delay

        call    curr                    ;숫자뿌린 것을 지운다.
        call    recover

        call    frame                   ;커서를 뿌린다.
;;-------------------------------------------------------------------------
----
```

#4.1 [number]

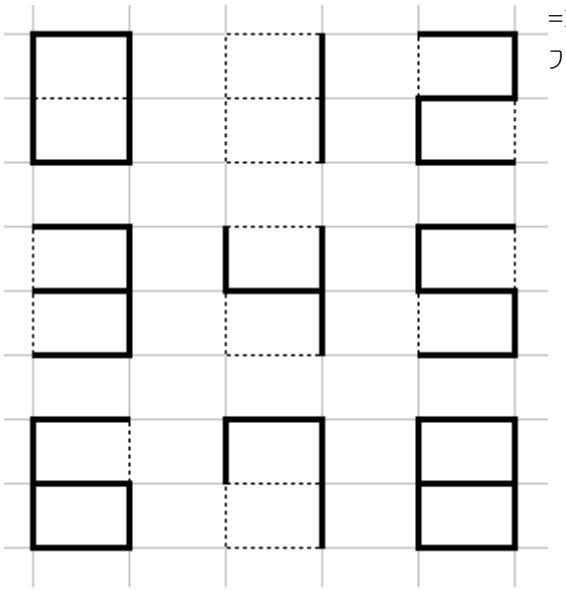
=> 현재 사용자가 지정한 cell의 정보를 확인하고 0에서 8의 숫자나 지뢰를 뿌리는 함수이다.
- 이 함수를 처음 구현할 때에는 숫자 하나하나에 대하여 디자인을 하여서 뿌려주는 방식으로 하려고 하였으나, 비효율적이라는 생각이 들었다.
- 그래서 전자시계에서 사용되는 방법을 차용하기로 하였다.

다음과 같은 방법으로 0부터 8까지의 숫자를 뿌릴 수 있도록 하였다. 그러므로 단지 필요한 것은

-가로로 줄을 긋는 함수와
-세로로 줄을 긋는 함수만

있으면 모든 숫자를 표현할 수 있게 된다.

#4.1.1 [horizontal]



=> 현재의 위치를 받아서 그 위치에 길이 3인 가로선을 긋는다.

```
;;-------------------------source code---------------------------------------
        mov     wtmp,cx
        add     wtmp,3
        mov     bh,0            ;page0
horiloop:
        mov     ah,0ch          ;function
        int     10h             ;display a pixel at (cx,dx)
        inc     cx
        cmp     cx,wtmp
        jbe     horiloop
;;-------------------------------------------------------------------------
----
```

#4.1.2 [vertical]

=> 현재의 위치를 받아서 그 위치에 길이 2인 세로선을 긋는다.

```
;;-------------------------source code---------------------------------------
        mov     wtmp,dx
        add     wtmp,2
        mov     bh,0
vertiloop:
        mov     ah,0ch          ;function
        int     10h
        inc     dx
        cmp     dx,wtmp
        jb      vertiloop
;;-------------------------------------------------------------------------
----
```

#4.1.3 [display@]

=> 현재의 위치를 받아서 지뢰를 뿌린다.

#4.1.4~4.1.12 [display0]~[display8]

=> 현재의 위치를 받아서 0~8의 숫자를 뿌린다.

-각각의 숫자마다 고유의 색을 가지도록 하였다.

#4.2 [delay]

=> 1초의 delay를 만드는 함수이다.

- 같은 instruction을 반복해서 주는 방식으로 delay를 주었다. 그래서 CPU의 속도가 느린 컴퓨터에서는 2초보다 긴 delay가 만들어 질 수도 있고 CPU의 속도가 빠른 컴퓨터에서는 2초보다 짧은 delay가 만들어 질 수도 있다.

- 위에서 사용된 [random]함수에서 random한 수를 얻기 위해서 쓰는 방법이 현재 시간을 seed값으로 받는 것이었다. 그런데 동시에 여러 번 호출하게 되면 비슷한 random값이 나온다. 이때에 사용할 수 있는 방법 중에 하나가 [random]을 호출하는 사이에 약간의 delay를 주는 것이다.

```
;;-------------------------source code--------------------------------------
        mov     bx,1388h
delay20:
        dec     bx
        mov     cx,2710h
delay30:
        mov     al,00h
        mov     ah,00h
        loop    delay30
        cmp     bx,0000h
        ja      delay20
;;-------------------------------------------------------------------------
----
```

2. 구현된 기능 및 화면 출력 결과

2.1 지뢰 찾기 게임에 구현된 주 기능

이 프로젝트에서 구현된 기능은 크게 다음과 같다.

- 게임 레벨(난이도) 입력과 오류체크 :

비디오 텍스트 모드에서 게임 난이도를 결정하기 위한 입력 값들 (row, column and mine)을 입력받으며, 제한 범위(5<= row <= 20, 5<= column <= 20, 0 < mine <= row*column/2)를 설정하여 이 범위를 벗어나는 경우, 오류 체크를 하여 벨소리와 경고 메시지를 보여주고 다시 입력받도록 한다.

- 두 가지 입력모드 (Keyboard / mouse) :

입력이 끝나면 키보드를 이용해서 게임을 할 것인지, 마우스를 이용해 게임을 할 것인지에 대해 사용자에게 물어 사용자가 선택하는 입력 모드로 게임을 진행하도록 하였다.

키보드 모드에서는 "m" or "M", "c" or "C", space bar, 각 방향키를 사용, 마우스 모드에서는 왼쪽 클릭은 지뢰를 여는 데에, 오른쪽 클릭은 지뢰라고 예상되는 곳에 표시하는 데에 쓰이면 표시된 지뢰에 다시 오른쪽 클릭을 하면 지뢰 표시가 사라지도록 하였다.

- 난수 발생을 통한 임의영역에의 지뢰 설치 :

게임 마다 난수를 발생시켜 임의의 영역에 지뢰가 설치된다.

- 화면 출력 및 비트맵 파일 출력 :

게임의 진행 화면은 비디오 그래픽 모드로 구현되었으며, 사용자가 지뢰라고 생각하는 부분에는 'm (or M)' 키를 눌러, 또는 마우스 오른쪽 클릭을 하여 팀 구성원 비트맵 사진을 출력한다.

- 주변의 지뢰의 개수 표시와 딜레이 :

사용자가 지뢰가 아닌 셀을 열었을 때, 주변에 있는 지뢰의 개수를 보여주며 1초 딜레이 후에 셀이 닫힌다.

- 간단한 소리 기능 :

게임 시작 전에 사용자가 게임 레벨을 설정할 때 제한된 범위를 벗어나게 되면, 그리고 키보드로 게임 진행 중 사용자가 게임 필드에서 커서를 이동시킬 때, 필드 범위를 벗어나려고 하는 이벤트(예를 들어, 맨 윗 줄 임에도 불구하고 그 윗줄로 커서를 이동시키려고 하는 경우)에 있어서는 간단한 벨소리를 출력한다.

- 게임의 종료 및 재시작 :

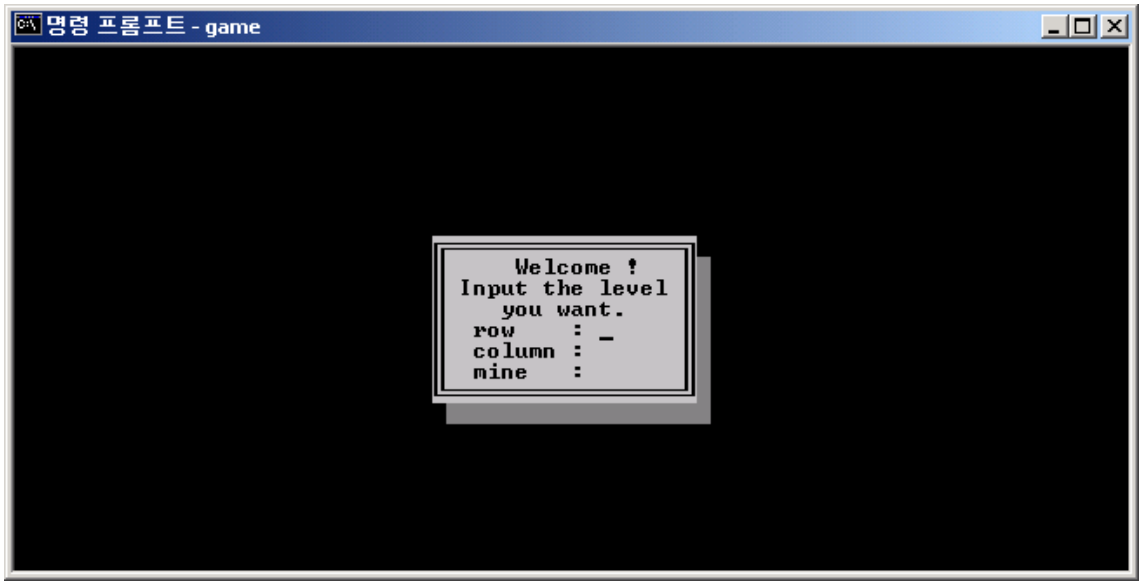
사용자가 지뢰를 열거나 모든 지뢰에 지뢰라고 생각되는 표시를 했을 경우 게임이 끝나는데, 사용자가 게임에서 진 경우에는 전체 지뢰의 위치를 보여주며, 2초 정도의 딜레이 후 자동으로 축하(승리한 경우) 또는 유감(패배한 경우)의 메시지를 보여주고, 게임을 다시 시작할 것인지 종료할 것인지에 대한 메시지를 출력하고 사용자로부터 입력을 요구했으며 재시작을 원하면 새로운 게임 레벨(난이도)을 설정하고 게임을 다시 시작할 수 있도록 하였다.

프로젝트에서의 요구하는 사항 모두를 구현하였으며, 마우스 기능과 간단한 소리 기능을 추가하였다.

2.2 화면 출력 결과

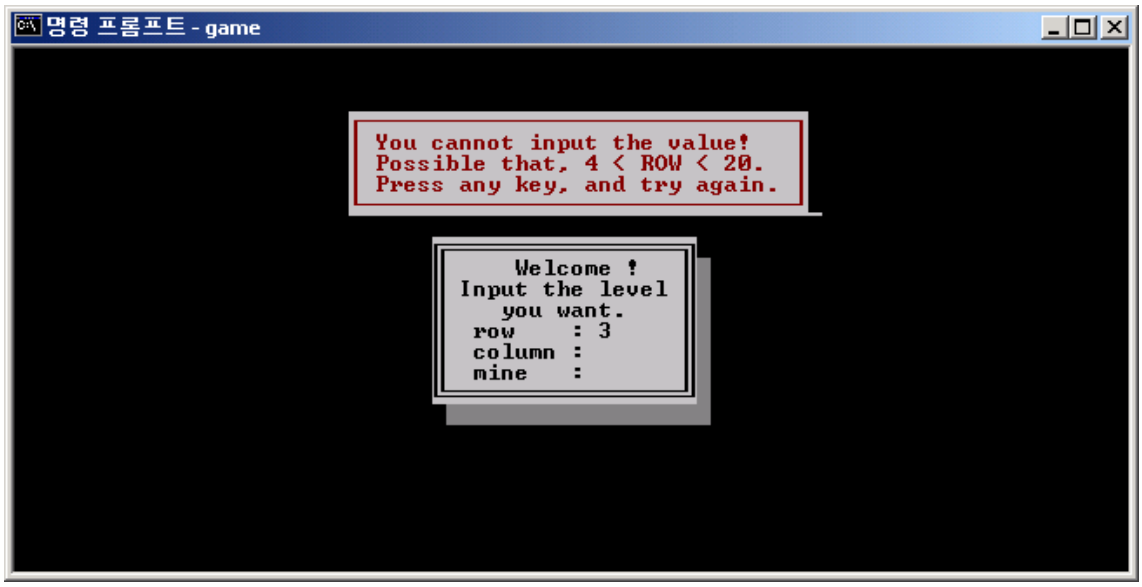
여기에서는 두 번의 게임 과정을 출력하였다. 첫 번째는 키보드 input 모드로 게임을 진행하고 게임에 실패한 경우를, 두 번째는 마우스 input 모드로 게임을 진행하고 게임에 승리한 경우를 보여준다.

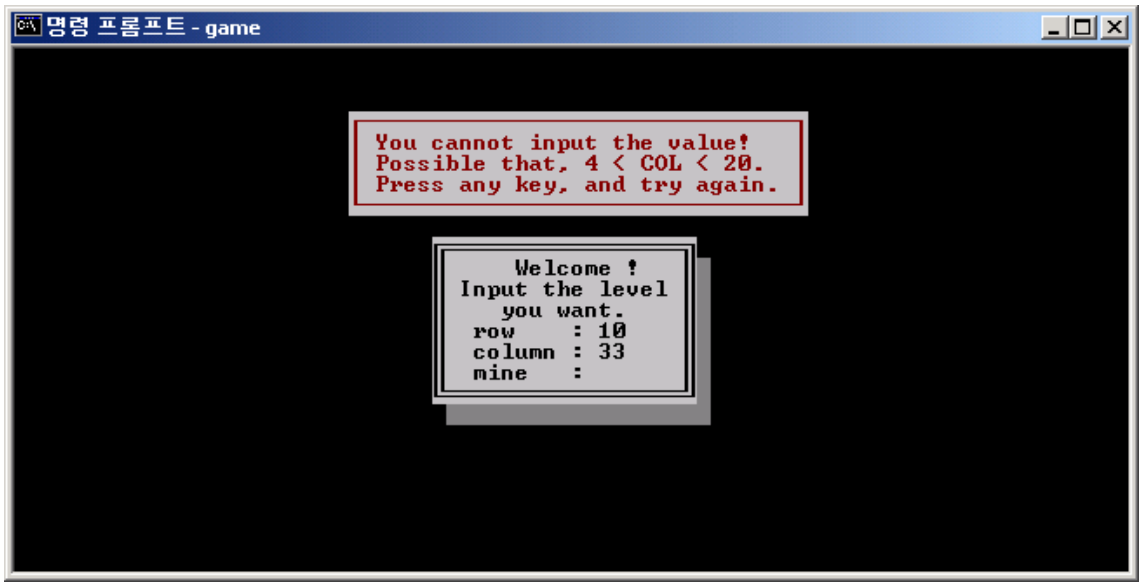
▼ 다음은 게임을 실행했을 때의 초기 화면이며, 게임의 난이도를 사용자로부터 결정할 수 있다.



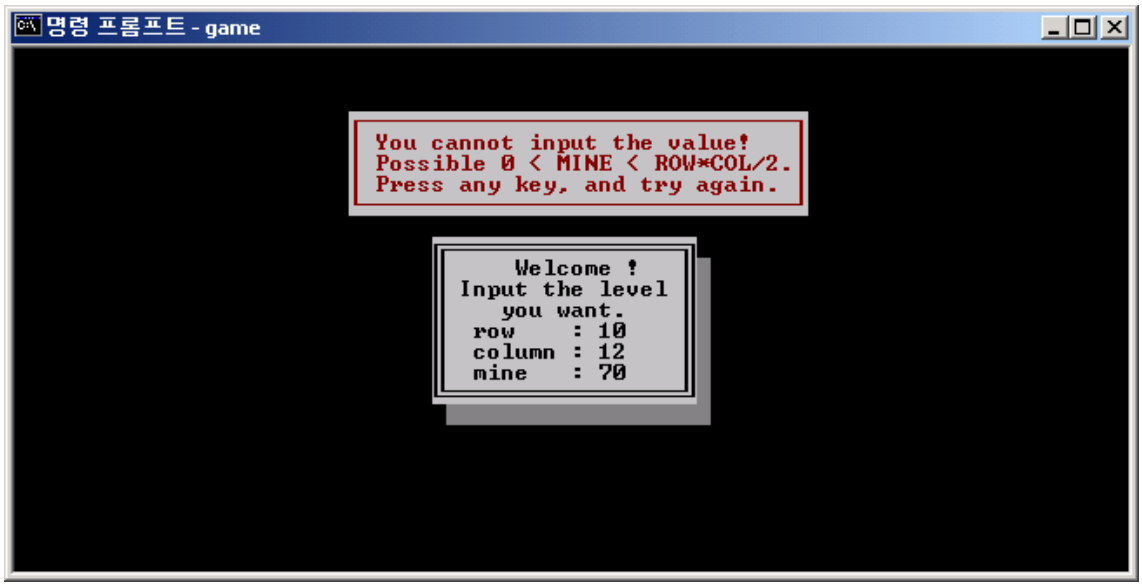
난이도는 제한 범위 내에서 가능하다. 만약 입력 값이 해당 범위에서 벗어나면 경고 메시지를 보여준다.

▼ 다음은 일련의 게임 레벨 설정 과정이다.



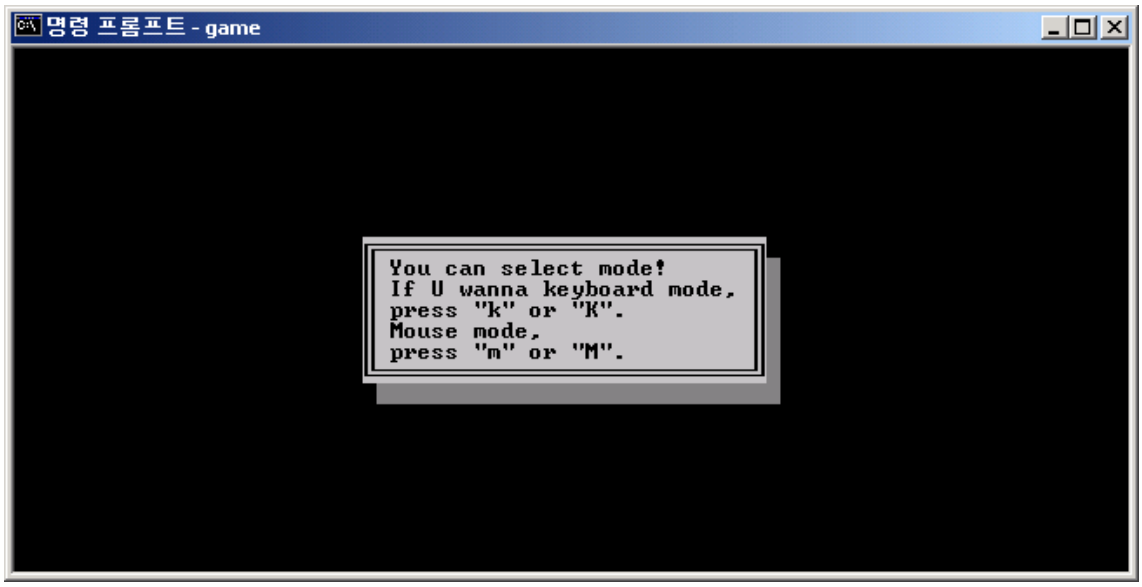

명령 프롬프트 - game
You cannot input the value!
Possible that, 4 < COL < 20.
Press any key, and try again.
Welcome !
Input the level
you want.
row : 10
column : 33
mine :


명령 프롬프트 - game
Welcome !
Input the level
you want.
row : 10
column : 12
mine : _


명령 프롬프트 - game
You cannot input the value!
Possible 0 < MINE < ROW*COL/2.
Press any key, and try again.
Welcome !
Input the level
you want.
row : 10
column : 12
mine : 70

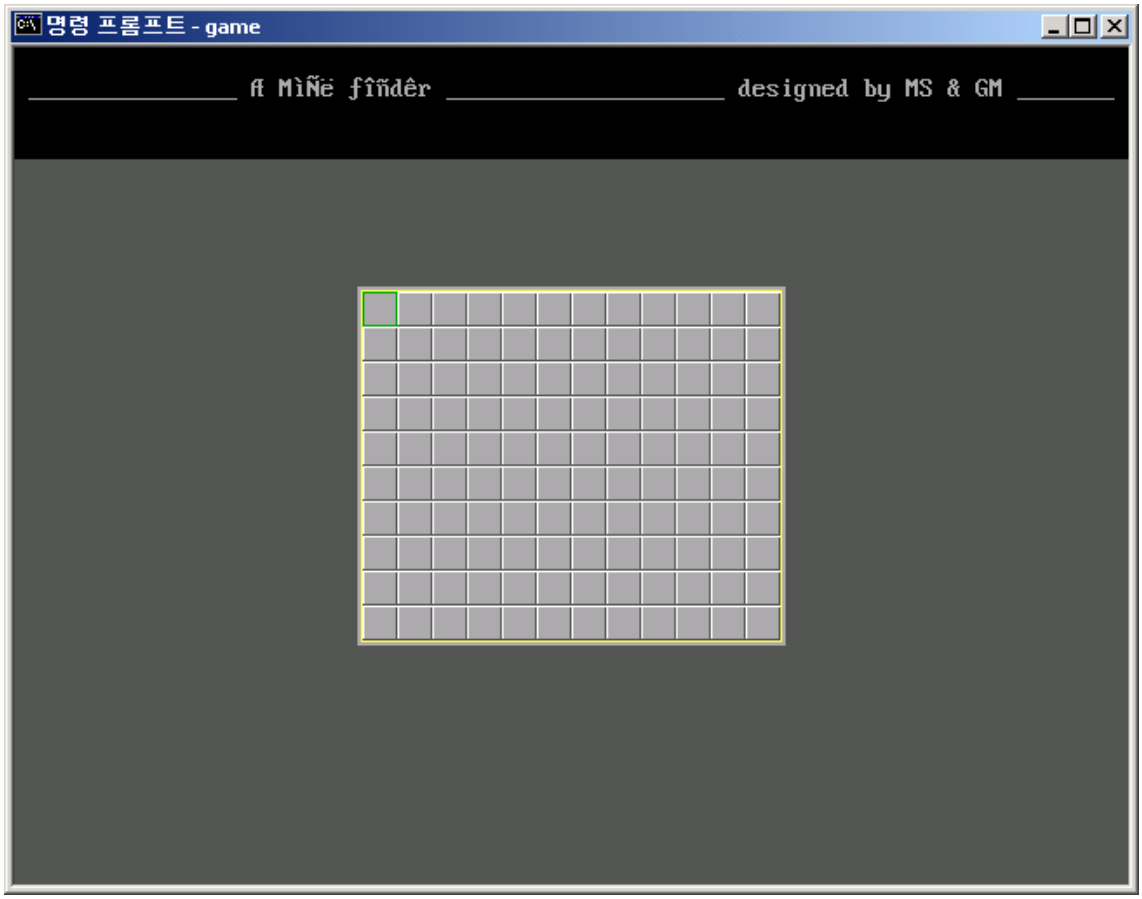

명령 프롬프트 - game
Welcome !
Input the level
you want.
row : 10
column : 12
mine : 35

게임의 난이도를 결정한 후에 게임을 진행할 모드를 결정한다.

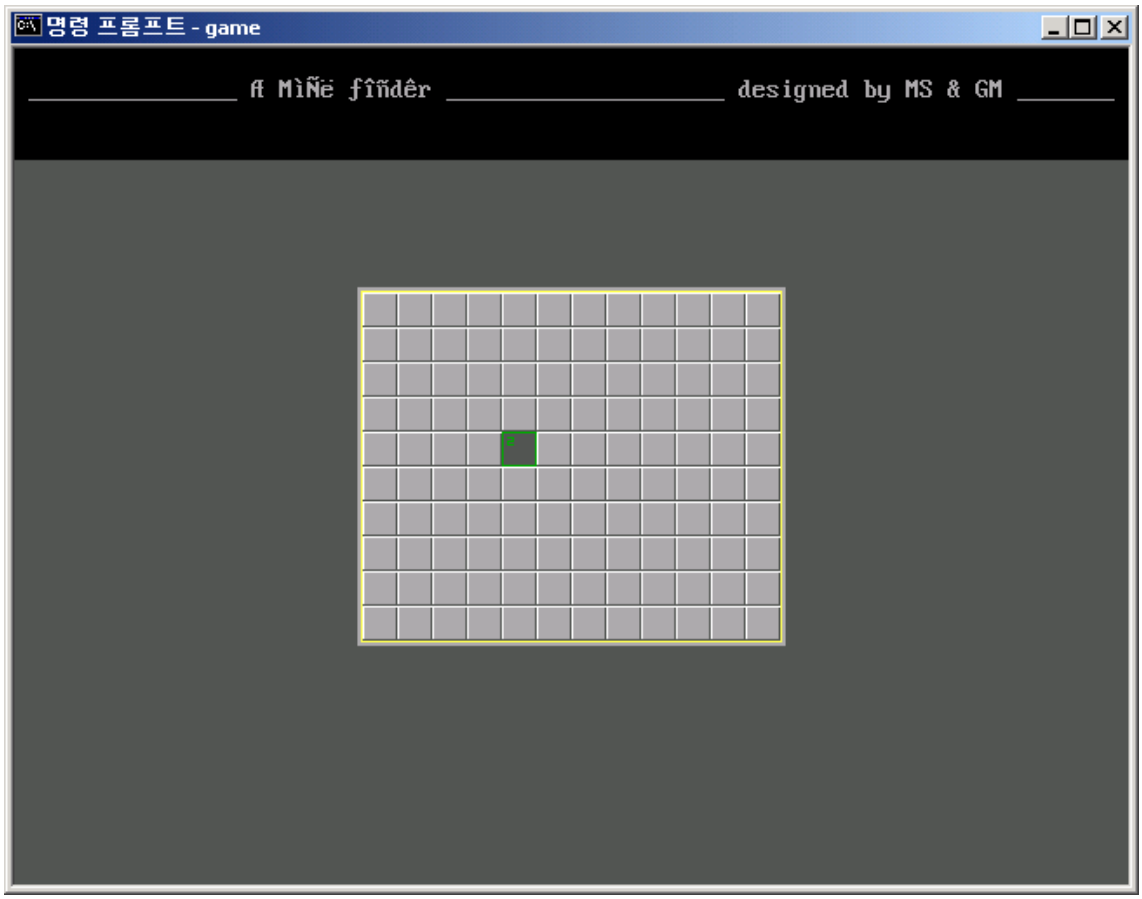


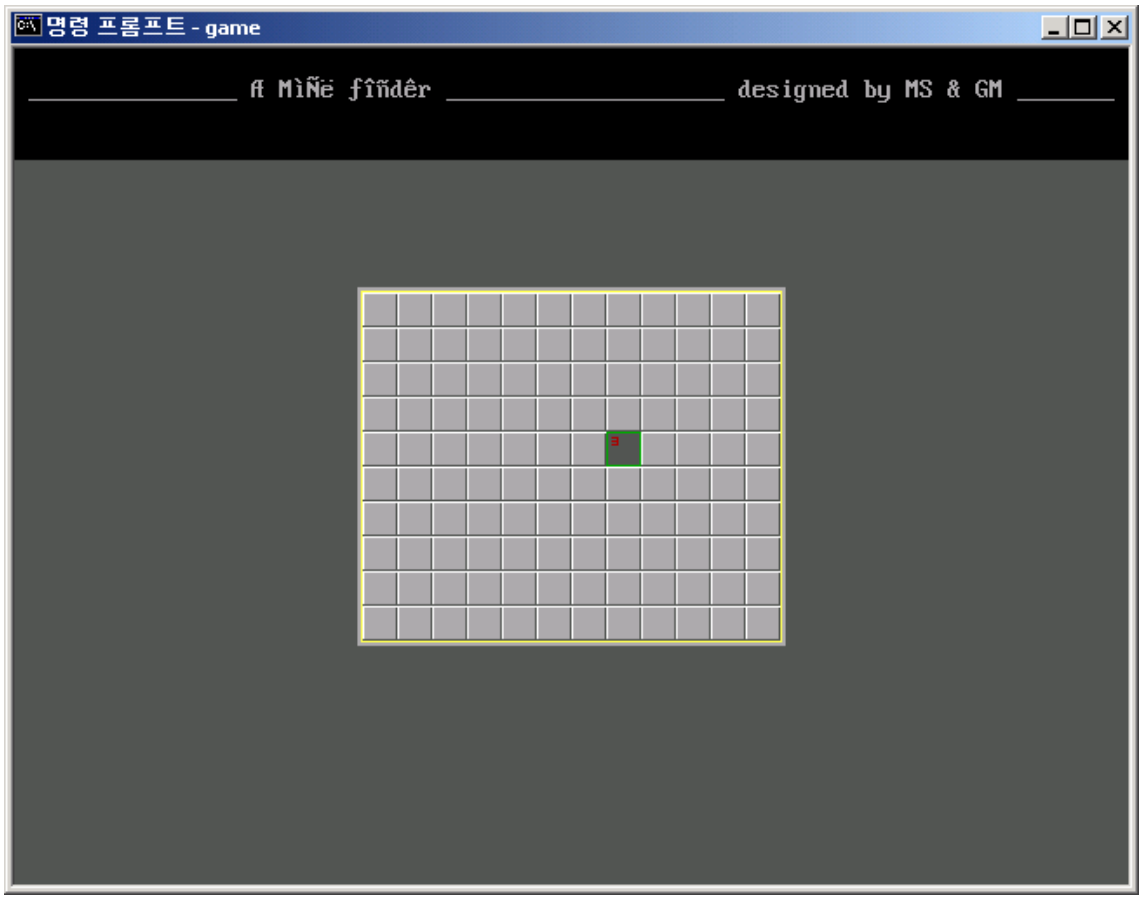
1) 키보드 모드로 패배하는 게임

첫 번째 칸에 커서의 이동을 보여줄 프레임이 배치되면서 시작한다.

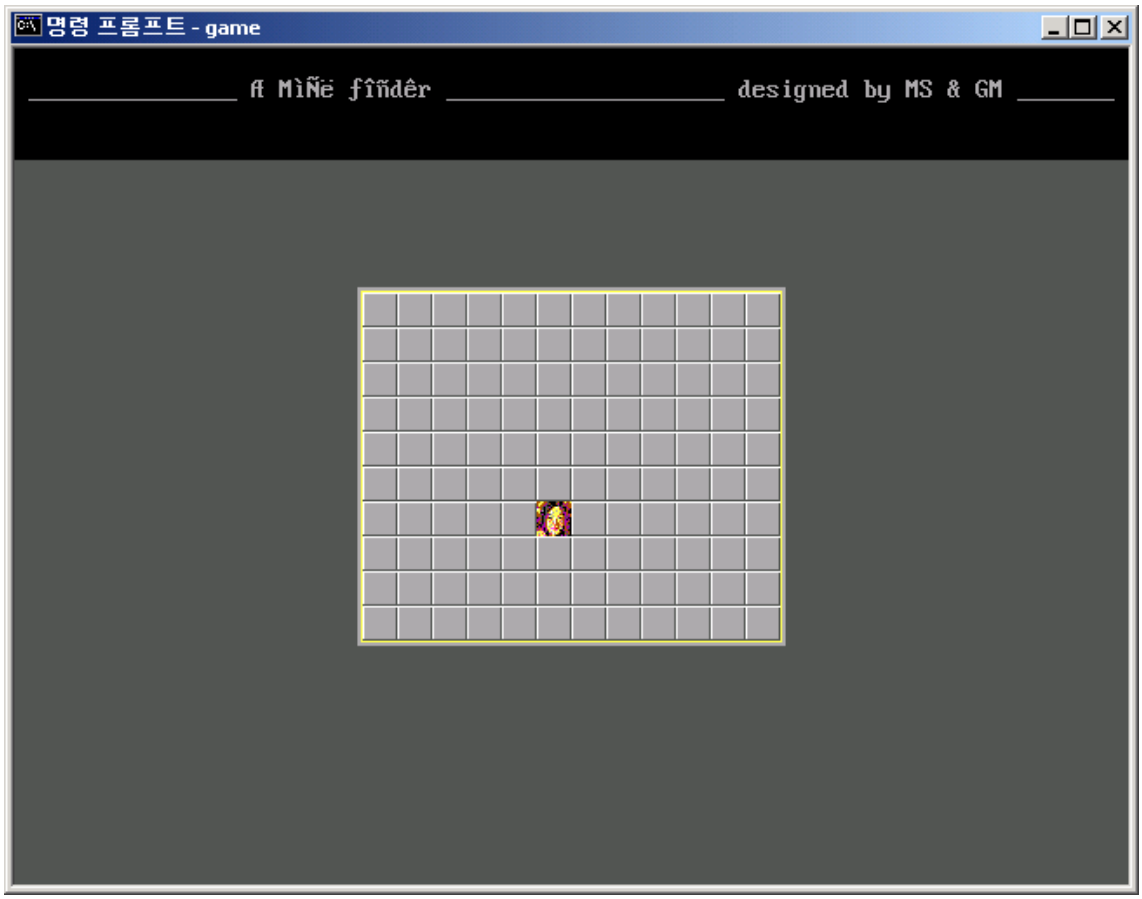


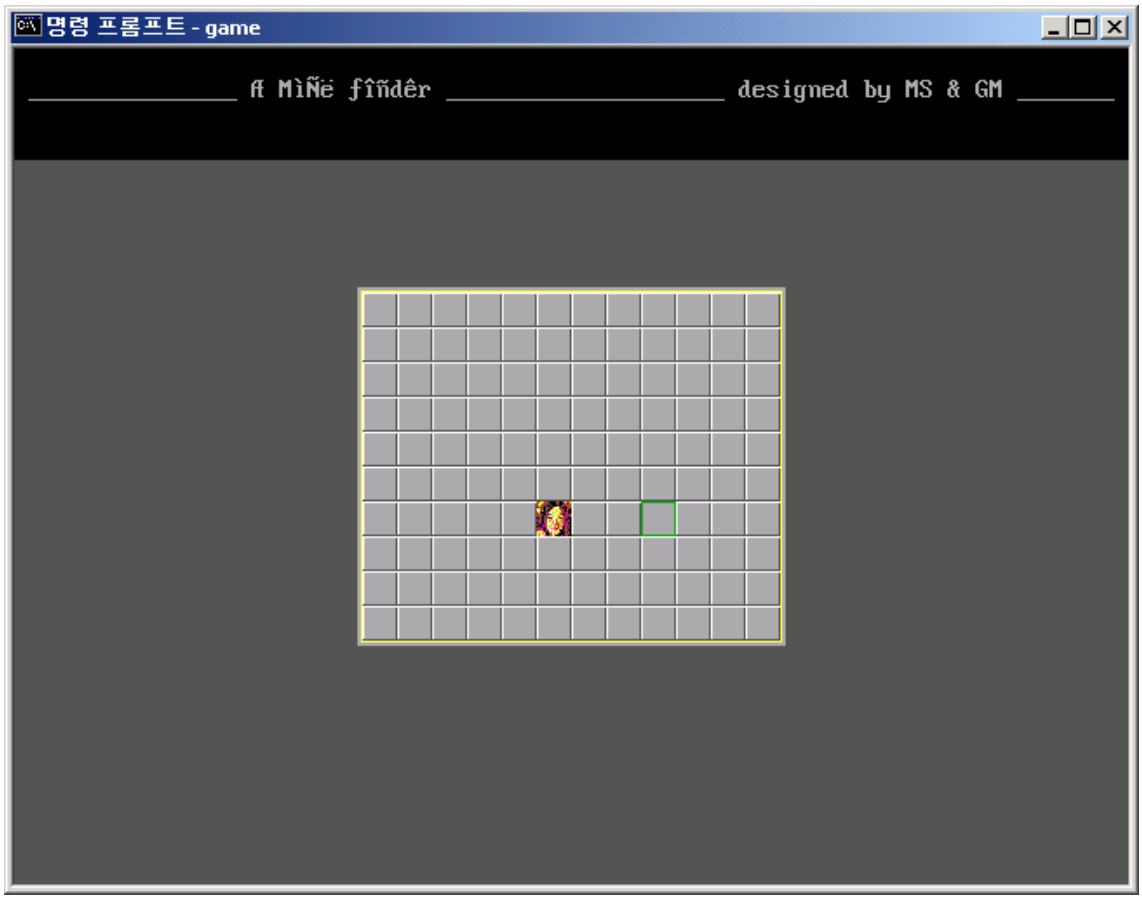
커서를 움직이고 스페이스 바로 셀을 열어 본다.



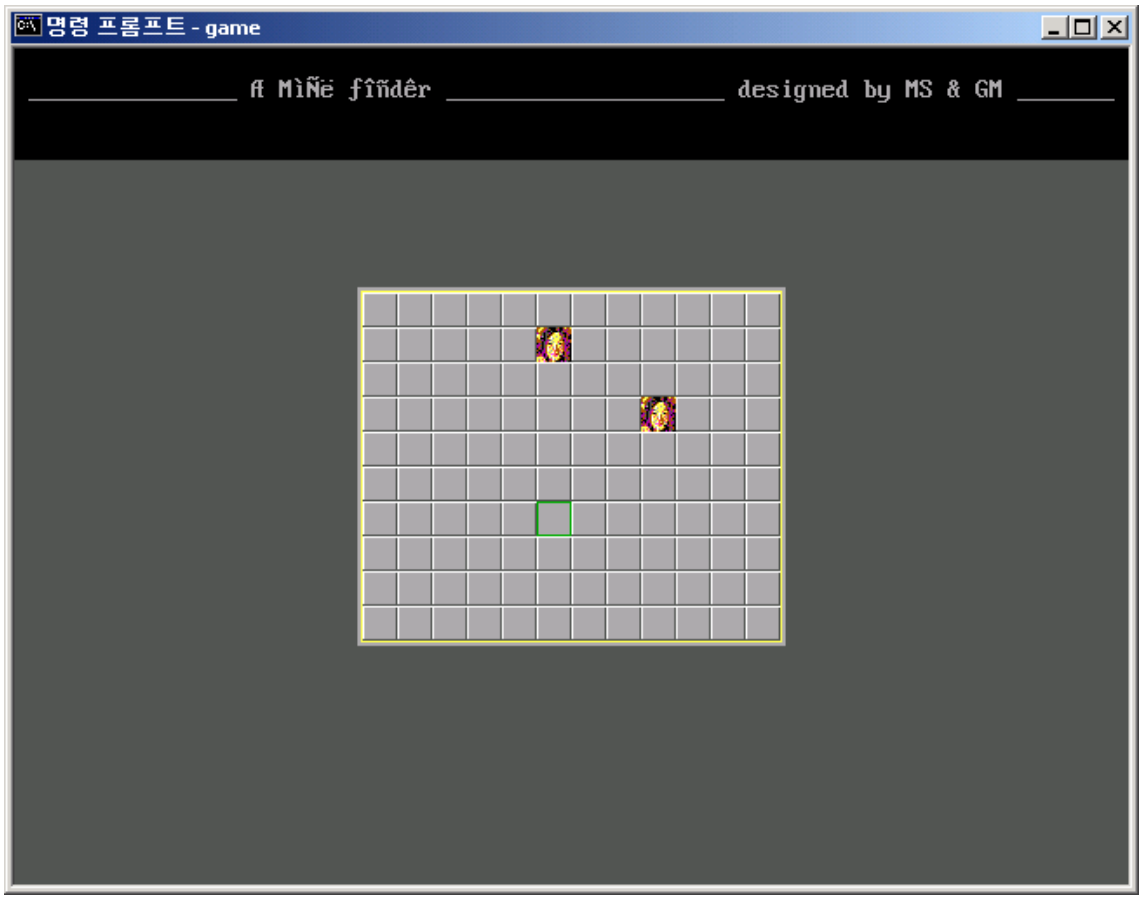

명령 프롬프트 - game
A MiNe finder
designed by MS & GM

"m" or "M"을 눌러 지뢰라고 예상되는 곳에 지뢰표시를 한다.

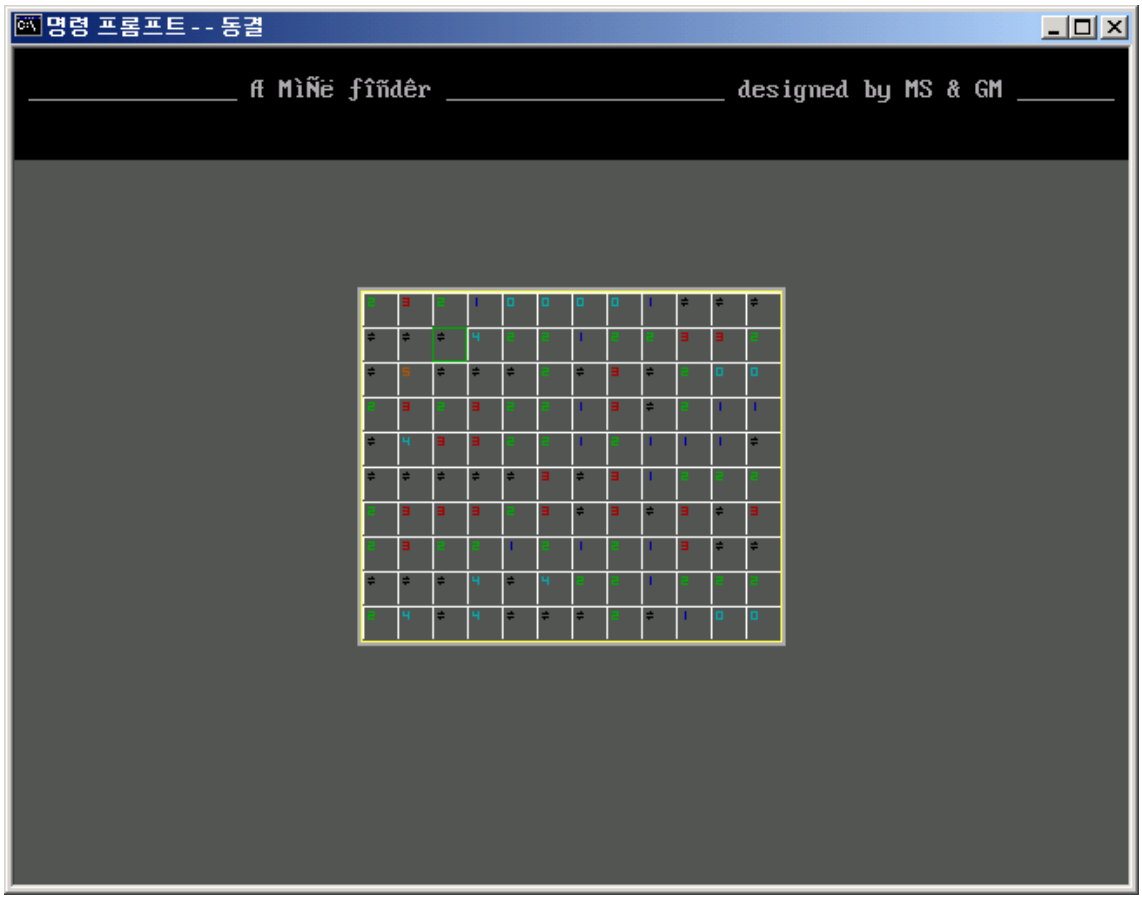



명령 프롬프트 - game
A MiNe finder
designed by MS & GM

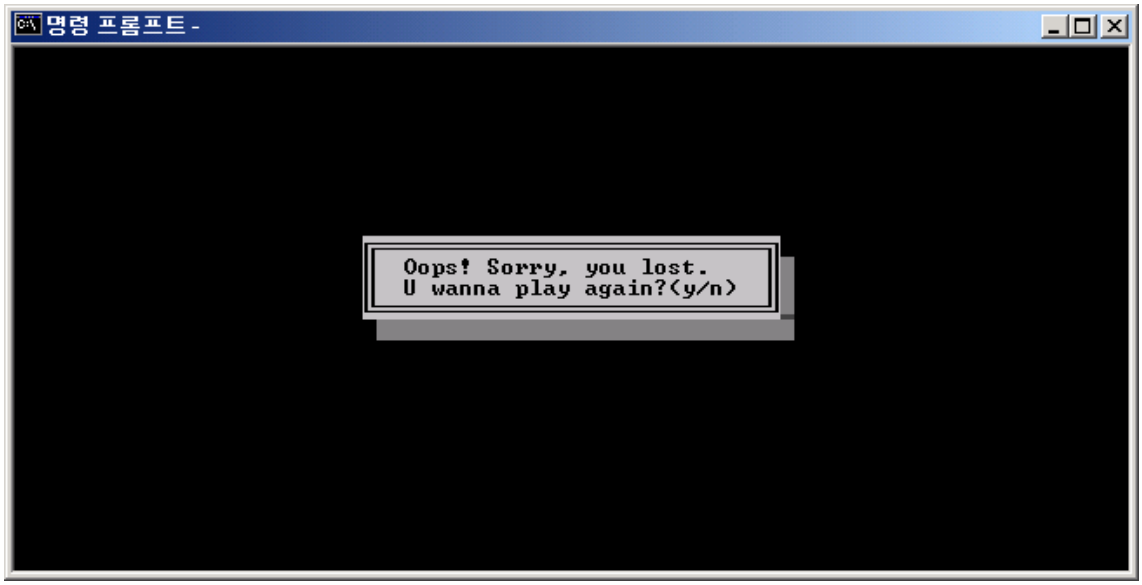
지뢰 표시를 삭제하고 다른 곳에 지뢰표시를 한다.



지뢰 셀을 열게 된 경우, 셀들의 모든 정보가 공개된다.



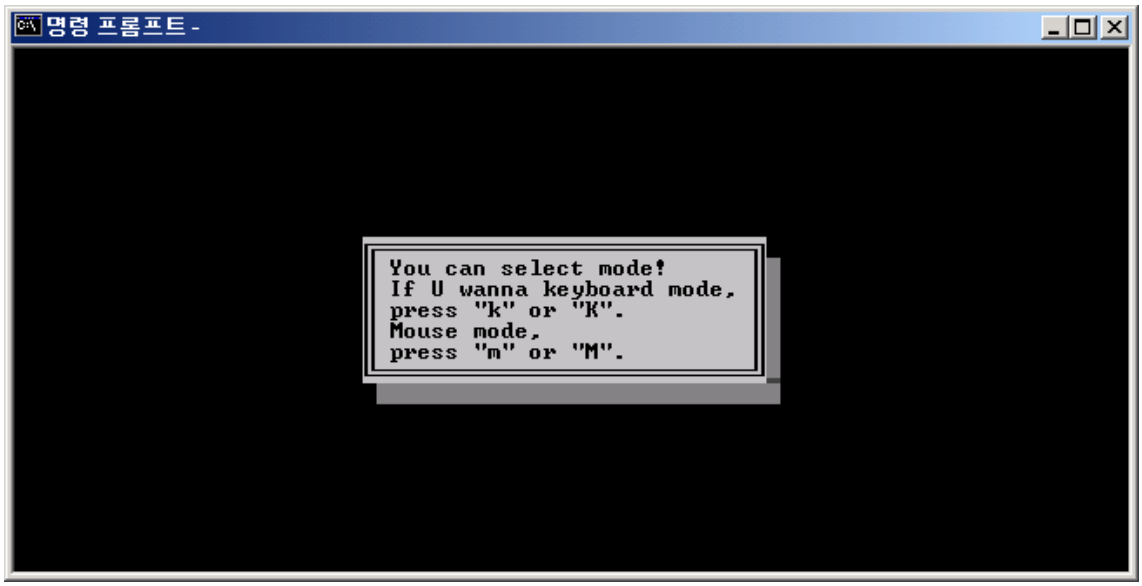
3초 후 자동으로 메시지를 보여주고 게임을 다시 시작할 것인지에 대해 묻는다.



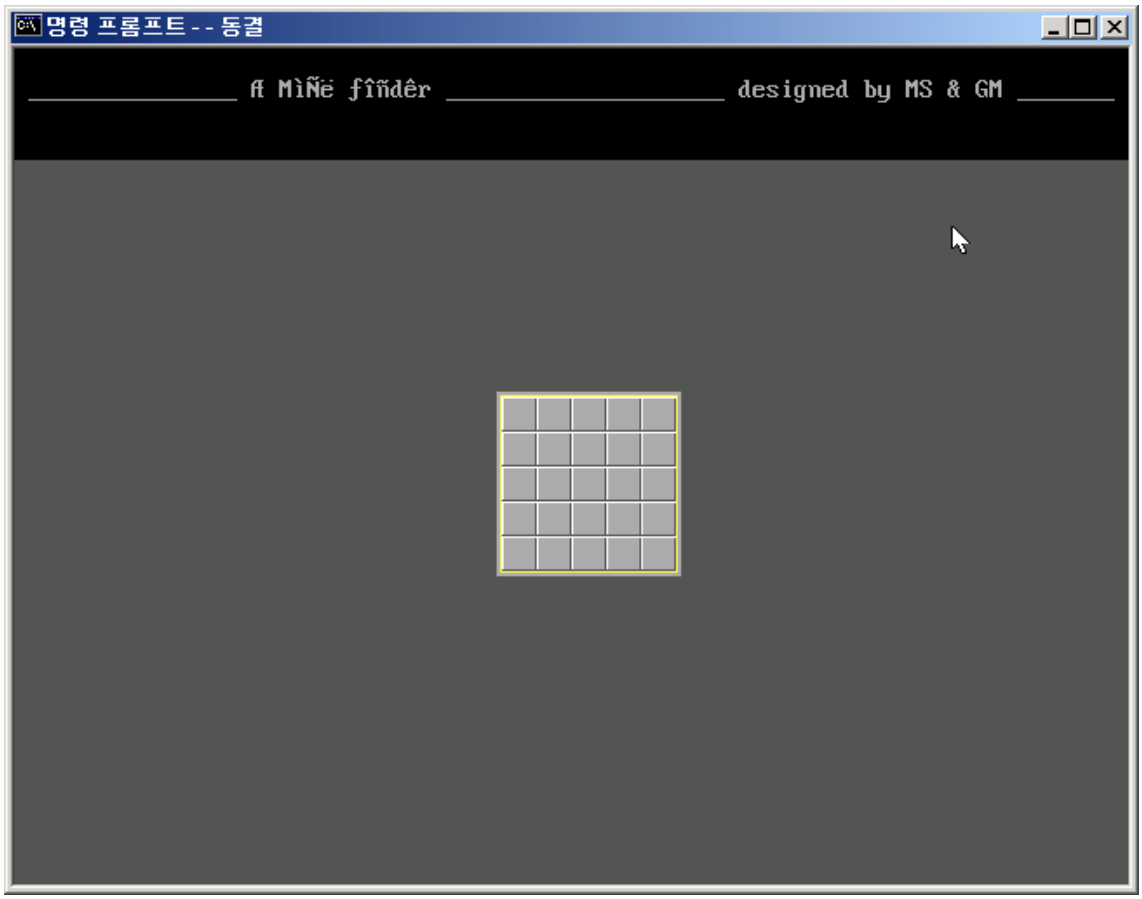
게임은 다시 초기화 되고 시작 단계에서 게임 레벨을 설정하도록 한다.

다음과 같이 난이도를 입력받는다.

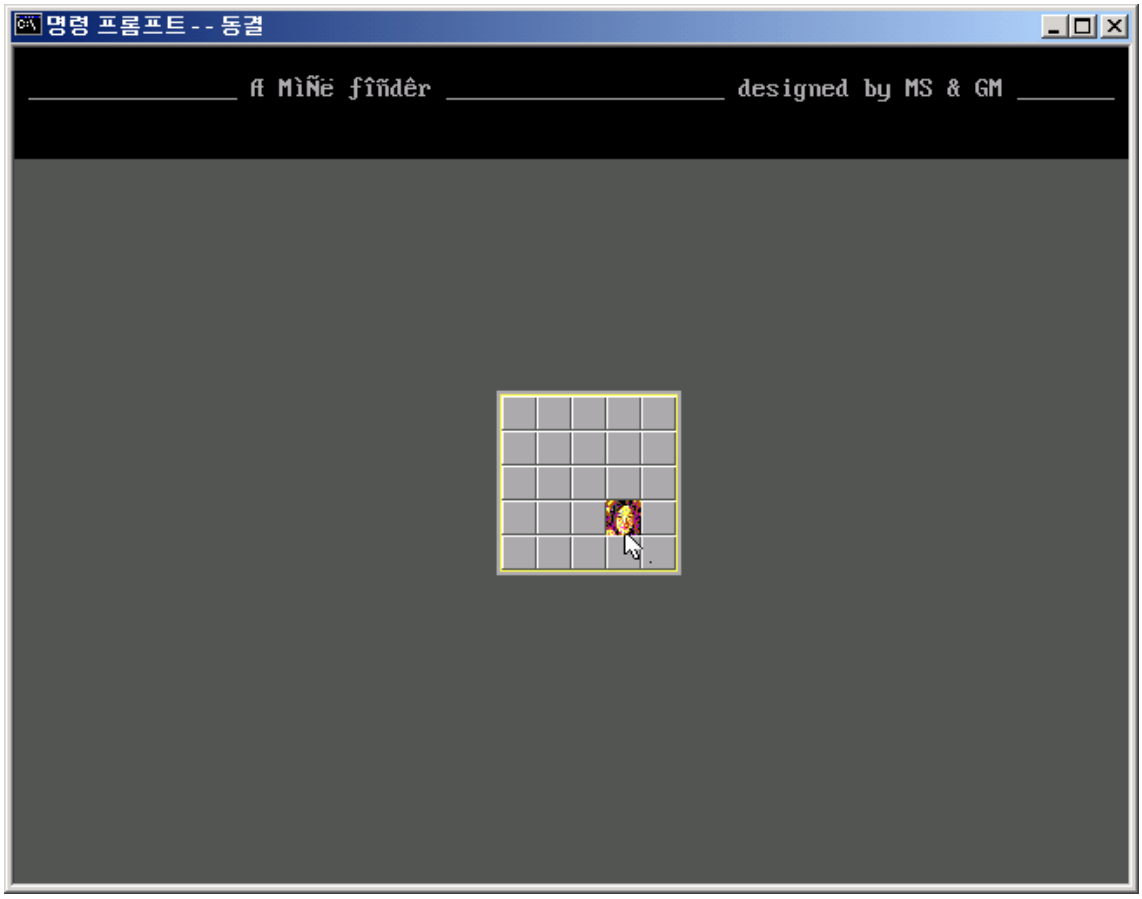
이번에는 마우스 모드를 선택해 본다.



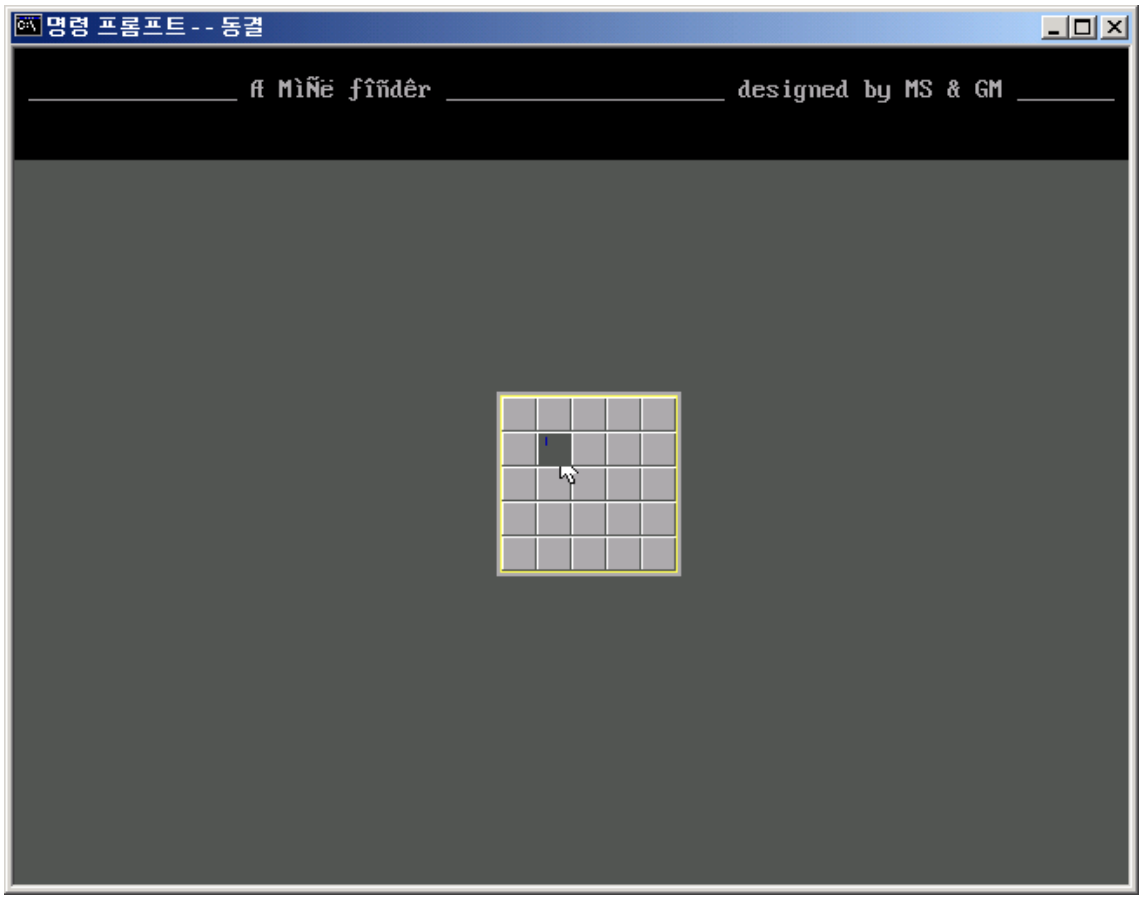
2) 마우스 모드를 게임을 진행하고 게임에서 승리하는 경우



마우스 포인터가 초기화되고 이번에는 커서의 이동을 보여주는 프레임이 보이지 않으며, 오른쪽 클릭을 하여 지뢰임을 표시할 수 있다.



셀을 열어 보면서 게임을 진행한다.



지뢰가 있다고 예상되는 곳에 지뢰표시를 한다.

게임에서 승리했기 때문에 3초 뒤, 종료 또는 재시작 메시지를 띄운다.

Ⅷ. Conclusion

1. 성능 개선 방법이나 어려웠던 부분 기술

1. 개선방안

(1) 마우스와 키보드를 동시에 사용할 수 있도록

=> 키보드 입력을 받는 인터럽트는 입력이 들어올 때까지 기다린다. 그러므로 time-slicing을 통해서 분할된 일정의 시간동안에는 키보드의 버퍼를 읽고 입력이 없으면 다시 일정 시간동안 마우스의 클릭을 읽는 방법으로 해결할 수 있다. 즉, 키보드와 마우스를 동기적으로 사용하도록 구현한다면 프로그램이 개선될 수 있다.

(2) 윈도우2000에서만 실행되는 단점개선 (dos기반의 모든 platform에서 실행 가능하도록)

=> 현재의 프로그램은 Windows 98과 Windows XP에서는 동작하지 않는다. 그리고 일반적으로 사용되지 않는 그래픽카드를 보유하고 있는 노트북과 같은 환경에서는 잘 작동하지 않는다. 아직까지 확실한 이유를 파악하지는 못했지만, 해결방안은 찾은 것 같다. 현재의 프로그램에서는 화면에 어떤 모양을 뿌릴 때에 dos 인터럽트를 이용해서 하였는데 video display 영역에 직접 접근하여서 화면에 뿌리는 작업을 한다면 대부분의 환경에서 실행될 것으로 생각된다.

(3) 마우스가 움직일 때에 포인터에 의해 배경색이 지워지는 현상 없애기

=> 현재의 프로그램에서 마우스 입력 모드로 실행이 되면 마우스가 움직이면서 기존의 이미지를 손상하는 경우를 발견할 수 있다. 임시방편으로 마우스 입력을 받는 중간 중간에 이미지를 반복적으로 refresh시키는 방법으로 손상된 이미지를 복구시키도록 하였다.

2. 어려웠던 부분

(1) assembly와 같은 저급 언어를 처음 접하게 되었기 때문에, assembly에서의 프로그래밍 방식에 익숙해지는 때까지 오랜 시간이 걸렸다. 하지만 지금은 오히려 저급언어의 프로그래밍이 더 편해졌다.

(2) 그리고 assembly가 구현 환경을 많이 타는 언어이기 때문에, 어떤 컴퓨터에서는 잘 실행이 되다가도 다른 컴퓨터에서는 잘 실행되지 않는 등 소프트웨어적으로는 이해할 수 없는 버그들이 발견되어서 그것을 개선하는 것이 힘들었다.

(3) 그 밖에 구현 사항에 있어서 비트맵 파일을 분석하는 방법이 어려워 파일을 I/O로 처리하지 않고 비트맵의 pixel을 분석하여 데이터 세그먼트에 올려 인터럽트를 거는 방식으로 출력할 수밖에 없었다.

2. 실행 방법

1. 실행 환경

- OS : Windows 2000 Professional
- 모니터 : 평면이 아닌 모니터 17인치 권장(평면 모니터에서는 실행하지 마시길...)

2. 실행순서

1. DOS command 창을 띄운다.

2. 실행 파일이 있는 directory로 이동한다.

3. 영문모드로 전환한다.
c:₩???> chcp 437

4. 전체화면모드로 전환한다.
=> alt-ENTER를 누른다.

5. 실행 파일을 실행시킨다.
c:₩???> game

6. 원하는 지뢰밭의 가로 길이와 세로 길이, 지뢰 개수를 입력한다.
-row, column의 수는 5개 이상 20개 이하로 한다.
-mine의 수는 지뢰밭의 넓이 반개까지 허용한다.

7. 키보드 입력모드와 마우스 입력모드 중에 한 가지를 선택한다.

8.1 키보드 입력모드로 들어갔다면, 커서로 이동을 하고
-지뢰표시하려면 'm',
-지뢰표시 지우려면 'c'를 누른다.
-셀을 열려면, 스페이스바를 누른다.

8.2 마우스 입력모드로 들어갔다면,
-오른쪽클릭을 홀수번하면 지뢰표시가 되고(길게 눌러야 할때도 있다.)
-오른쪽클릭을 짝수번하면 지뢰표시가 지워진다.
-셀을 열려면, 왼쪽클릭을 한다.

9. 지뢰를 열면, 게임이 끝나고 다시 하려는 메시지가 나온다.
-다시 시작하려면, 'y'를 누르고 -그만하려면, 'n'을 누른다.

역할 분담

2000160157 유 미 선 :

initialization, input module, game processing module(공동/마우스)
외부 디자인

200160184 이 진 무 :

game setting and display module, game processing module(공동/키보드)
내부 디자인

스케줄

기간	내용	
9/27 ~ 10/5	9/27	팀 구성
	9/28 ~ 10/1	배경 지식 조사
	10/2 ~ 10/3	요구사항 분석 및 구현 사항에 대한 논의
	10/4 ~ 10/5	1차 보고서 작성 및 제출
10/6 ~ 10/26	10/6 ~ 10/13	어셈블리 언어 익히기
	10/14 ~ 10/15	프로젝트 전체 설계
	10/16 ~ 10/18	모듈 설계 및 역할 분담
	10/19 ~ 10/23	구현 시작
	10/24 ~ 10/26	2차 보고서 작성 및 제출
10/27 ~ 11/16	10/27 ~ 11/13	모듈 및 기능 구현
	11/14 ~ 11/16	3차 보고서 작성 및 제출
11/17 ~ 12/13	11/17 ~ 12/10	디버깅 및 성능 개선
	12/11 ~ 12/13	4차 보고서 작성 및 제출