2017년 2학기

객체지향프로그래밍

CSED232

Assignment #2

담당교수 : 윤은영

학번 : 20160074

학과 : 화학공학과

이름 : 고진민

POVIS ID : eric9709

--------------------------------------------------------------------------------------------------------------

<명예서약>

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

--------------------------------------------------------------------------------------------------------------

1. **프로그램 개요**

이 프로그램은 간단한 게임 프로그램이다. 처음에 게임 시작 방법을 입력 받은 뒤 게임이 실행되면, 화면에 지도와 여러 정보들이 표시된다. 사용자는 올바른 입력을 통해 이동하거나 어뢰를 발사해 배를 모두 격침시켜야 한다. 성공할 경우, 게임에 승리하고 남은 턴 동안 배를 모두 격침시키지 못하거나 어뢰가 모두 소진된 경우 게임에서 패배한다. 각 보트와 어뢰, 지도는 동적할당을 통해서 구현하게 된다.

메뉴별 간단한 기능은 다음과 같다. 1번 메뉴를 선택하면 미리 정해진 지도의 크기와 배의 수로 게임을 진행하게 되며, 2번 메뉴를 선택하면 사용자가 지도의 높이, 너비와 배의 수를 지정할 수 있게 된다. 잘못된 입력을 받은 경우에는 다시 입력을 받게 된다.

1. **Class 구성**

|  |  |
| --- | --- |
| **Object** | **Torpedo : public Object** |
| - int axisX;  - int axisY; | - bool torpedoStatus |
| = void moveUp();  = void moveDown();  +Object();  +Object(int a, int b);  +void moveLeft();  +void moveRight();  +int getAxisX() const;  +int getAxisY() const;  +void setAxisX(int const newAxisX);  +void setAxisY(int const newAxisY); | +Torpedo();  +Torpedo(int a, int b) : Object(a, b);  +bool getTorpedoStatus();  +void setTorpedoStatus(bool newTorpedoStatus);  +void move();  +enum TorpedoStatus  {  Unavailable = false,  Available = true  }; |
| **Boat : public Object** | **BoatA : public Boat** |
| - bool visibility;  - bool boatStatus; | - int boatAStatus; |
| +Boat();  +Boat(int a, int b) : Object(a, b);  +bool getVisibility() const;  +bool getBoatStatus() const;  +void setVisibility(bool const newVisibility);  +void setBoatStatus(bool const +newBoatStatus);  +enum BoatVisibility  {  Invisible = false,  Visible = true  };  +enum BoatStatus  {  Unavailable = false,  Available = true  }; | +BoatA();  +BoatA(int a, int b) : Boat(a, b);  +int getBoatAStatus() const;  +void setBoatAStatus(int newBoatStatus);  +void move();  +enum statusA  {  MidLeft = 1,  MidRight,  Left,  Right  }; |

|  |  |
| --- | --- |
| **BoatB : public Boat** | **BoatC : public Boat** |
| - int boatBStatus; | - int boatCStatus; |
| +BoatB();  +BoatB(int a, int b) : Boat(a, b)  +int getBoatBStatus() const;  +void setBoatBStatus(int newBoatStatus);  +void move();  +enum statusB  {  UpLeft = 1,  UpRight,  DownRight,  DownLeft  }; | +BoatC();  +BoatC(int a, int b) : Boat(a, b);  +int getBoatCStatus() const;  +void setBoatCStatus(int newBoatStatus);  +void move();  +enum statusC  {  Sink = 0,  UpOne,  UpTwo,  }; |

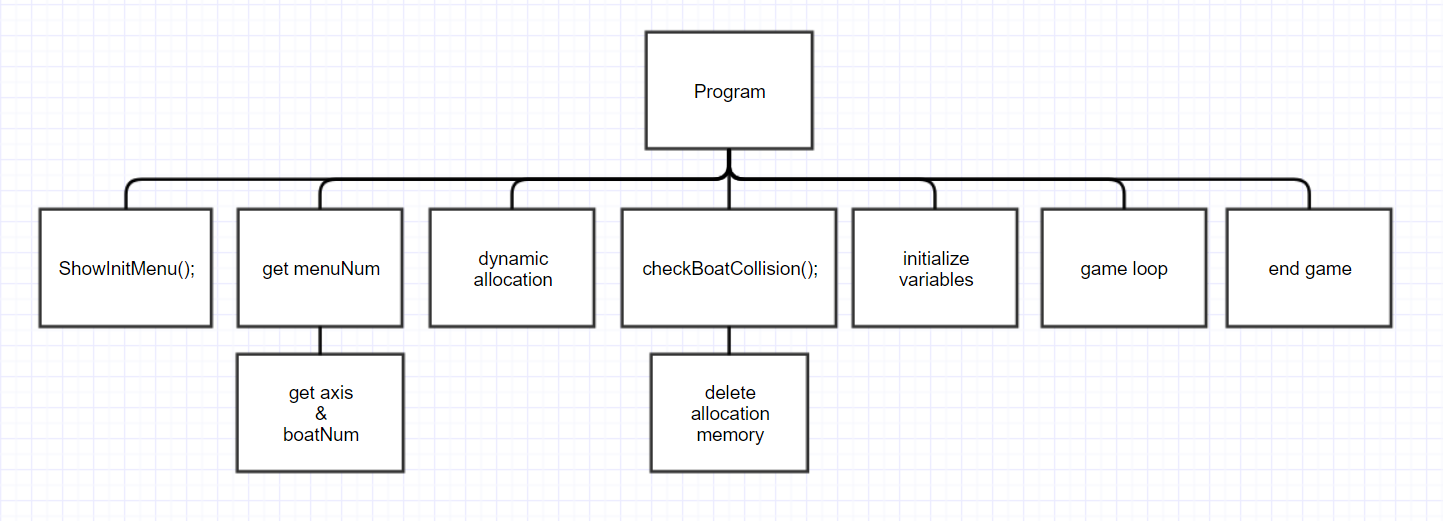
각 클래스는 public 상속을 통해 이어져 있다. BoatA/BoatB/BoatC 클래스는 Boat 클래스를 상속받고 Boat/Torpedo 클래스는 Object 클래스를 상속받는다. Object 클래스에는 좌표를 나타내는 변수와 해당 변수를 불러오고 설정하는 get/set 메소드가 존재한다. 또한 move 함수들이 존재하는데 방향별로 좌표를 바꾸는 역할을 한다. 단, Up과 Down은 protected, Right와 Left는 public이다. Torpedo 클래스에는 어뢰의 상태를 저장하는 변수와 해당 변수를 설정하고 불러오는 get/set 메소드와 어뢰를 이동시키는 메소드가 존재한다. 상태는 enum 선언을 통해 Available / Unavailable 두 가지가 존재한다. Boat 클래스는 visibility와 boatStatus 두 변수를 가지며 각 변수는 보트의 잠수 여부와 격침 여부를 나타낸다. 각 변수를 불러오고 설정하는 get/set 메소드가 존재한다. 격침 여부는 상태는 enum 선언을 통해 Available / Unavailable 두 가지가 존재하고 visibility는 역시 enum 선언을 통해 Visible / Invisivble 두 가지가 존재한다. BoatA/BoatB/BoatC 클래스는 각 보트의 status(Boat 클래스의 그것과는 다르다)를 저장하는 변수를 가지고 변수를 불러오고 설정하는 get/set 메소드가 존재한다. 또한 해당 변수를 이동시키는 메소드가 존재하며 각 보트의 status(배의 위치와 다음에 어디로 이동할지를 표현하는 상수)에 대한 상수가 enum으로 선언되어 있다.

클래스 모두에 대해 생성자(constructor)를 이용해 멤버 변수 중 초기화가 필요한 변수들을 초기화했다. 기본적으로 default constructor와 두 정수 변수를 입력받는 constructor가 있다. 각각 default, custom 게임에 대한 것이며, 좌표 입력 외에는 다른 것이 없다. Object에서는 생성자가 인스턴스의 좌표를 rand 함수를 이용해 임의로 지정한다. Torpedo에서는 생성자가 인스턴스의 status 변수를 Unavailable로 설정한다. 어뢰는 쏘기 전까지는 비활성 상태이기 때문이다. Boat에서는 visibility와 status를 각각 Visible, Available로 설정한다. BoatA/BoatB/BoatC에서는 각 보트의 status(위치 방향 등)를 rand 함수를 이용해 임의로 설정한다.

각 함수의 구체적인 기능과 구현은 소스 코드 및 프로그램 설명 부분에서 진행할 것이다.

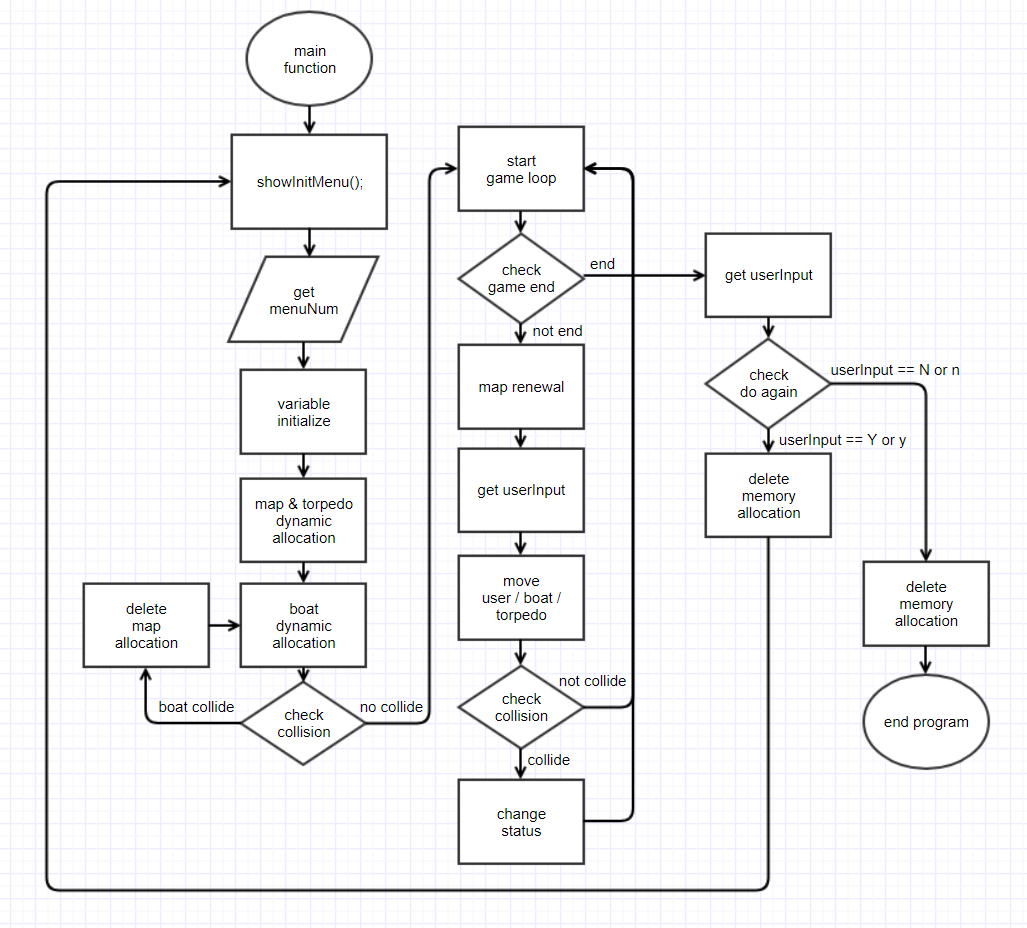
1. **구조도 및 알고리즘**

이 프로그램의 전체적인 structure chart는 다음과 같다.



<전체 프로그램의 structure chart>

main 함수의 대략적인 알고리즘은 아래 그림과 같다.



<main 함수의 flow chart>

1. **소스 코드 및 프로그램 설명**

main 함수를 제외한 .cpp 파일에 대한 헤더 파일은 class의 정의뿐이고 기타 메소드의 구현은 .cpp 파일에 진행되어 있으므로 헤더 파일에 대한 설명은 생략하도록 한다. 또한 class 구현 파일에서의 get, set 메소드에 대한 설명은 자명하므로 생략한다.

**<main.cpp>**

핵심이 되는 코드라고 할 수 있는 main.cpp 코드이다. 이 파일에서는 크게 5개의 함수를 사용한다.

**void showInitMenu 함수**는 초기 메뉴 화면을 출력하는 함수이다.

**void initMap 함수**는 mapArray와 map의 길이, 너비를 넘겨받아 mapArray를 모두 ~로 초기화하는 함수이다.

**void printMap 함수**는 mapArray와 map의 길이, 너비와 보트 / 어뢰 / 횟수 / 점수를 넘겨받아 화면에 출력하는 함수이다.

**bool checkBoatCollision 함수**는 boat가 이동하면서 문제가 발생하는지 확인하는 함수이다. 충돌이 발생하면 false, 충돌하지 않으면 true를 반환한다. 각 boat의 배열과 수, map의 길이, 너비, 남은 횟수를 넘겨받아 판단한다. 이 함수의 목적은 동적 할당을 통해 배치된 배들이 이동하면서 이동하면 안 되는 영역을 침범하거나 서로 충돌하는 경우가 있는지 판단한다. 전체 attempt가 12 이상이면 12회(3과 4의 LCM)만 턴을 진행시키고 12보다 작으면 해당 횟수만큼 판단한다. 각 경우에서는 공통적으로 다음과 같은 순서로 충돌 여부를 판별한다. 보트가 벽을 넘거나 유저 영역을 침범하는지 확인 -> A끼리의 충돌 판별 -> B끼리의 충돌 판별 -> C끼리의 충돌 판별 -> A, B사이의 충돌 확인 -> A, C사이의 충돌 확인 -> B, C사이의 충돌 확인. 이후 충돌하지 않으면 턴을 넘겨 다시 확인하고, 12회 혹은 attempt만큼 진행하기 전에 충돌하면 false를, 충돌하지 않으면 true를 반환한다.

**main 함수**는 게임의 코어 구현을 담당하는 함수이다. 먼저 처음에는 변수를 선언하고 srand로 시간 종속 seed를 생성한다. 이후 메뉴 루프로 진입한다. 화면을 지우고 초기 메뉴를 출력한 뒤 메뉴 번호를 입력받는다. 1, 2 이외의 입력이 진행된 경우 루프를 나가 프로그램을 종료한다. 1이 입력된 경우 미리 정해진 map의 크기와 보트의 수로 게임을 진행하고(map 너비 5, 높이 10, 보트 수 5) 2가 입력된 경우 map의 크기와 보트의 수를 유저가 입력하게 된다. 이 때, 입력하는 내용이 정수가 아닌 경우, 다시 입력받게 되며 구현이 불가능한 경우(높이가 1이라 보트가 있을 수 없거나 map의 크기보다 보트의 수가 더 많은 경우) 역시 다시 입력받는다.

이후 과정은 초기 메뉴에서 1과 2를 입력한 것에서 차이가 없으므로 공통으로 설명하겠다. 변수들을 초기화한 뒤, 지도와 어뢰를 크기에 맞게 동적할당 한다. 어뢰의 수는 보트 수의 2배, 전체 횟수는 map의 가로와 세로의 합에 보트의 수를 곱한 수이다. 보트를 격침시켰을 때에는 보트A는 100점, B는 200점, C는 300점을 얻는다. 승리 시, 남은 횟수에 50점을 곱하고 남은 어뢰에 200점을 곱해 점수에 추가한다. 이후 배를 배치하는 루프를 시행한다. 루프의 처음에는 A, B, C의 배의 수를 랜덤으로 배정한다. 이후 그 수에 맞게 각 배의 배열을 동적할당한다. 이후 checkBoatCollision 함수를 이용해 충돌 여부를 검사한 뒤 충돌하는 경우 할당받은 메모리를 모두 반환하고 다시 루프를 시작한다. 충돌이 발생하지 않는 경우에는 루프를 빠져나가 계속 진행한다.

배가 배치된 후에는 초기화된 값들을 바탕으로 게임 루프를 진행한다. 루프 한 번이 게임 유저 입력 한 번이다. 루프의 처음에는 게임에 승리할 조건(map에 남은 보트가 없을 경우)과 패배할 조건(턴이 없거나 어뢰가 소진되었고 map에 어뢰가 없는 경우)을 확인하여 만족되면 게임 루프를 이탈한다. 승리 조건과 패배 조건이 동시에 만족되는 경우 승리가 우선순위가 높다. 이후 map을 갱신해 출력한다. 먼저 initMap 함수를 호출한 뒤 유저, 어뢰, 보트들을 map에 배치하고(유효하면서 보이는 것들만) printMap을 호출해 map을 출력한다. 이후 유저 명령을 입력받는다. 대소문자 구분 없이 w, a, s, d가 아닌 입력을 받는 경우 이 루프를 다시 시작해 다시 입력받도록 한다. 올바른 입력이 이루어진 경우 각 입력에 맞는 행동을 진행한다. S의 경우 유저 이동 없이 넘어간다. A와 D의 경우 벽에 붙은 경우가 아니면 이동을 진행하고, 벽에 붙은 경우 해당 방향으로 진행하려 하는 경우 잘못된 입력으로 간주한다. W의 경우 어뢰를 발사하지만, 남아있는 어뢰가 없는 경우에는 잘못된 입력으로 간주한다. 어뢰를 발사하는 과정은 발사하는 torpedoArray에서 올바른 순서에 해당하는 어뢰의 위치를 보트 바로 위로 초기화하고, status를 Available로 갱신한다. 이후 오브젝트들을 이동시킨다. 어뢰의 경우, 처음 만들어진 어뢰는 이동시키지 않으며 map의 끝에 도달한 어뢰는 status를 unavailable로 바꾼다. 이외의 map 위에 있는 어뢰는 이동시킨다. 이후 boat들도 이동시킨다. 이들은 배치 단계에서 이미 충돌에 대한 검증이 끝나서 추가 확인은 필요 없다. 이후 어뢰와 보트가 충돌하는지를 검사한다. 충돌하는 경우 보트와 어뢰의 status를 모두 unavailable로 바꾸고 각종 변수(점수, 보트 수 등)를 갱신한다. 이후 남은 횟수를 1 줄인 뒤 루프의 처음으로 이동한다. 게임이 종료되어 루프 밖으로 나온 경우, 종료 이유에 따라 메시지와 기타 변수들을 출력한 뒤 재시작 여부를 입력받는다. 승리 시 앞서 언급한 보너스 점수를 가산해 출력한다. 대소문자 구분 없이 y, n이 아닌 경우 잘못된 입력으로 간주, 다시 입력받는다. N을 입력하면 동적 할당받았던 모든 메모리를 반환한 후 프로그램을 종료하고, y를 입력하면 동적 할당받았던 모든 메모리를 반환한 후 초기 루프의 맨 처음으로 돌아가 초기 메뉴를 출력한다.

**<Object.cpp>**

이 파일에서는 move 메소드에 대한 설명은 자명하므로 생략한다. 단 map의 번호 부여는 왼쪽 위가 (0, 0)이고 오른쪽으로 이동하면 x좌표가 증가하며 아래로 이동하면 y좌표가 증가한다. 이들 외에는 default 생성자와 두 정수변수를 입력받는 생성자가 있는데, 인스턴스의 좌표를 map 범위 내에서 rand 함수를 이용해 임의로 지정한다.

**<Torpedo.cpp>**

move 메소드는 moveUp 메소드를 호출한다. 이들 외에는 default 생성자와 두 정수변수를 입력받는 생성자가 있는데, 기본적으로 torpedoStatus를 Unavailable로 설정한다.

**<Boat.cpp>**

Default 생성자와 두 정수변수를 입력받는 생성자가 있는데, 기본적으로 visibility와 boatStatus를 Visible과 Unavailable로 설정한다.

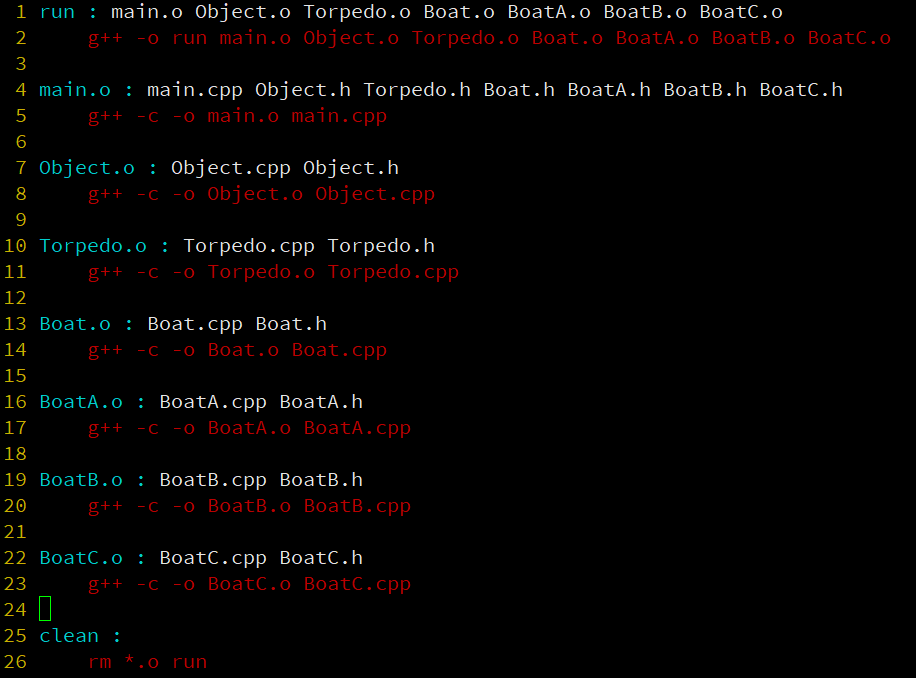
**<BoatA.cpp / BoatB.cpp / BoatC.cpp>**

move 메소드는 각 보트의 상태를 확인하고 상태에 맞게 다음 위치로 보트를 이동시킨다. 가령 A보트가 루프에서 맨 왼쪽에 있으면 오른쪽으로 이동시키고, C보트가 다음 차례에 가라앉아야 하면 visibility를 Invisible로 바꾼다. 이들 외에는 default 생성자와 두 정수변수를 입력받는 생성자가 있는데, boatStatus를 rand 함수를 이용해 임의로 설정한다.

1. **프로그램 실행**

리눅스 환경은 학교 실습 서버를 이용했다. 이번 과제에 사용한 소스 파일들은 모두 /home/std/eric9709/CSED232/ASSN2 위치에 저장되어 있다.

아래 그림처럼 Makefile을 만들고, make 명령어를 통해서 컴파일을 진행하면 컴파일이 완료된다.



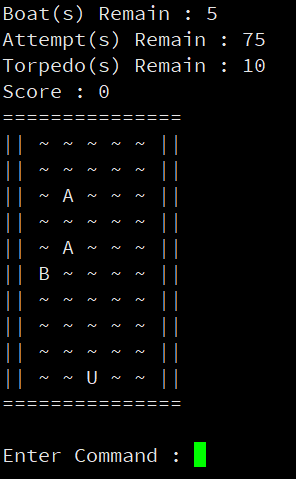
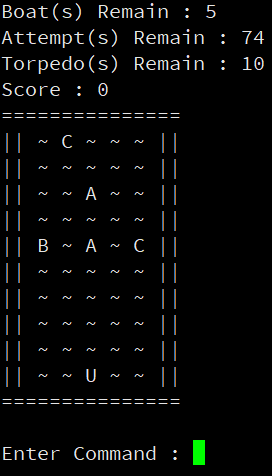
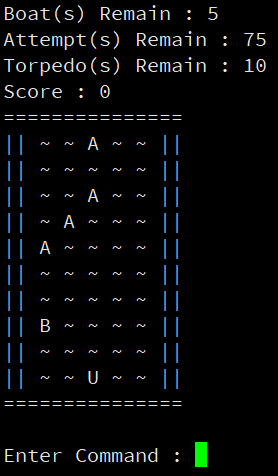
<makefile의 모습>

run 파일을 실행하면, 이미지와 같은 화면이 출력된다.



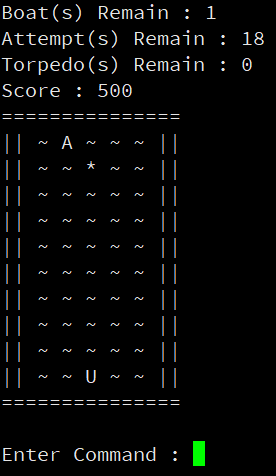
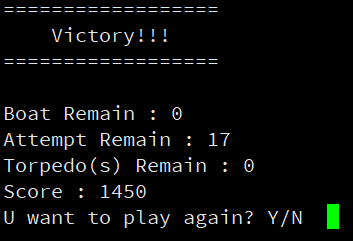
<초기 메뉴 화면>

초기 메뉴 화면에서 1을 입력하면 미리 정해진 map의 크기와 보트의 수로 게임을 시작한다. 이미지에서 볼 수 있듯이 출력 때마다 생성된 보트의 위치가 다른 것을 알 수 있다. 보이지 않는 보트가 있는 것은 C보트가 잠수 중인 상황이다.

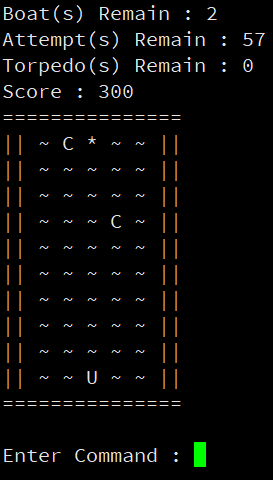
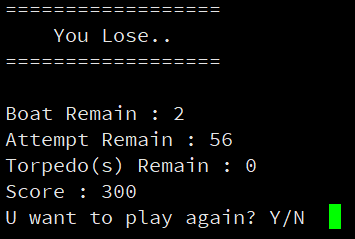
<왼쪽부터 게임 시작 직후와 한 턴이 지난 뒤의 모습, 그리고 아예 다른 게임을 시작한 모습>

게임이 종료되면 이후 출력되는 화면은 아래와 같다. 아래 이미지에서 A는 오른쪽으로 이동할 예정이다. 우선순위에 따라 승리로 판정되었다. 또한 보너스 점수가 가산된 모습을 볼 수 있다.

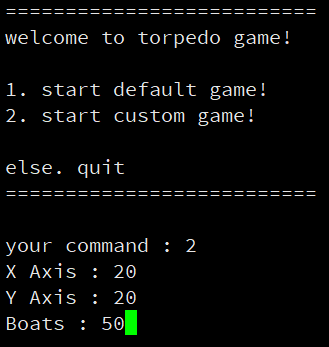
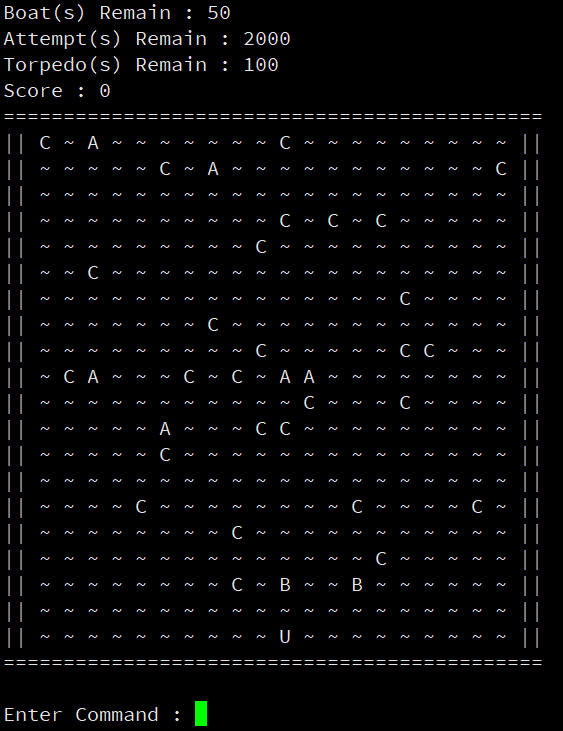
<승리한 경우의 출력>

게임에서 패배하면 아래 이미지와 같은 그림이 출력된다. 아래는 어뢰가 모두 소진되고 map에 어뢰가 없는 경우의 패배조건이다.

<패배한 경우의 출력>

초기 메뉴에서 2를 입력해 custom 게임을 만드는 경우, 출력은 아래 이미지와 같다. 아래의 경우는 너비, 높이가 20이며 보트의 수는 50척이다. 잠수한 C보트 역시 존재한다.

<custom 게임에 대한 출력의 모습>

1. **토론**

* 메모리의 동적할당을 진행하는 과정에서 heap 쪽 문제가 발생했다. 처음에는 할당 과정에서 너비의 값을 넣어야 하는 곳에 높이의 값을 넣어서 문제가 발생했고, 이후에도 할당한 메모리를 반환하지 않고 재할당하려 해서 문제가 발생하기도 했다. 다행히 해당 문제는 전부 해결할 수 있었다.
* Game에 해당하는 클래스를 새로 정의하면 좋겠다는 생각이 구현 중반쯤 들기 시작했지만 늦었다고 판단해 계속 강행한 점이 조금 아쉬웠다. 만약 클래스를 새로 만들었다면 아마 main 함수의 줄 수가 많이 줄었을 것이다.
* 보트와 어뢰를 어떻게 지도 위에 대응시키고 격침 여부와 보이는지를 판단할 수 있을까 고민하다 멤버 변수를 추가했고 상수를 enum으로 별도로 지정했다. 또한 \*\* 형태로 보트의 수에 맞게 한 줄짜리 2차원 배열을 만들어 구현했다. 어뢰의 경우도 마찬가지였는데, 어뢰는 쏘는 횟수에 해당하는 위치의 어뢰를 보이게/유효하게 설정함으로서 구현을 진행했다. 가령 어뢰가 4번째로 발사되는 상황이었다면 배열의 [3] 위치의 어뢰를 활성화했다.

1. **결론**

* 이번 과제에서 요구한 메뉴 구현과 메뉴 처리를 모두 수행할 수 있었다.
* Class의 상속 개념을 이용해 코딩을 진행할 수 있었다.
* 주어진 내용 외에도 추가적인 class 정의와 메소드 및 멤버 변수의 선언을 통해 구현을 더 효율적으로 진행할 수 있다.

1. **개선 방향**

* Game에 대응되는 큰 클래스 하나를 정의해서 사용했다면 main 함수의 가독성이 늘어났을 것이고, 유지보수성도 증가시킬 수 있었을 것이다.
* 배를 배치하는 과정에 있어서 시간이 오래 걸릴 수 있다. 가령 20\*20 짜리 map 위에 50개의 배를 배치하려면 수 초가 걸린다. 이는 보트의 재할당 과정을 생략하고 좌표만 바꿔주는 방법으로 코드를 수정하면 동적할당 및 반환 과정이 생략되므로 시간이 좀 더 단축될 것이다.