**2017년 2학기**

**[CSED 232] 객체지향프로그래밍**

**ASSN 2 보고서**

제목: 어뢰 게임

POVIS ID: dj1122

학과: 산업경영공학과

학번: 20130220

이름: 이다진

“나는 이 프로그래밍 과제를 다른  사람의 부적절한 도움 없이 완수하였습니다.”

목차

Ⅰ. 개요1

Ⅱ. 전체 조와 알고리즘1

1. 전체 구조1

2. 알고리즘1

3. 클래스 구성4

Ⅲ. 프로그램 실행 방법8

Ⅳ. 예제 및 실행 결과9

Ⅴ. 실행 결과에 대한 토의14

1. 토론14

2. 결론15

3. 개선 방향15

Ⅰ. 개요

본 프로그램은 C++의 상속 기능을 이용하여 “어뢰 게임”을 실행하고 콘솔 화면에 출력한다. 프로그램 구현을 통해 C++의 상속 기능 이용 방법을 습득한다. 리눅스 및 유사환경에서의 프로그램 실행을 통해 Text User Interface 환경에 익숙해진다. 이러한 과정을 통해 객체 지향 프로그래밍과 게임 개발의 연관성을 인식한다.

Ⅱ. 전체 구조와 알고리즘

1. 전체 구조

(1) 메인 메뉴를 출력한다.

(2-1) 1 입력: 바다 너비 5, 바다 길이 10, 배의 개수 5로 설정한다.

(2-2) 2 입력: 바다 너비, 바다 길이, 배의 개수를 입력한 대로 설정한다.

(3) 사용자, BoatA, BoatB, BoatC의 객체를 생성한다.

(4) 게임 화면을 출력한다.

(5) 사용자 입력 값을 받는다.

(6) BoatA, BoatB, BoatC, Torpedo를 각각 이동시킨다.

(7) Torpedo와 BoatA, BoatB, BoatC의 위치가 일치하는지 확인하고 일치하면 각각 삭제한다.

(8) 승패 조건을 검사하여 승패 결과를 출력하고 승패가 결정되지 않았다면 (4)로 돌아간다.

(9-1) y 입력: (1)로 돌아간다.

(9-2) n 입력: 메인 프로그램을 종료한다.

2. 알고리즘

(1) 사용자, BoatA, BoatB, BoatC의 객체를 생성한다.

make\_boats {

1) user에 Boat를 동적할당한다.

2) rand()를 이용하여 생성할 BoatA, BoatB, BoatC의 개수를 정한다.

3) BoatAList, BoatBList, BoatCList에 정해진 개수 만큼 BoatA, BoatB, BoatC를 동적할당하여 추가한다.

4) 주어진 턴의 개수만큼 보트를 움직여보고 위치가 겹치는 경우가 있다면 make\_boats()를 재실행한다.

}

(2) 사용자 입력 값을 받는다.

user->move {

1) 사용자 입력 값을 받는다.

2) swith {

‘a’ 입력: user를 왼쪽으로 이동

‘d’ 입력: user를 오른쪽으로 이동

‘w’ 입력: hasShot <= true

‘s’ 입력: 아무 것도 실행하지 않음

}

}

(3) BoatA, BoatB, BoatC, Torpedo를 각각 이동시킨다.

1) BoatAList에 있는 모든 BoatA에 대해 move()를 호출한다.

2) BoatBList에 있는 모든 BoatB에 대해 move()를 호출한다.

3) 3턴마다 BoatCList에 있는 모든 BoatC에 대해 move()를 호출한다.

4) 모든 Torpedo에 대해 move()를 호출한다.

(4) Torpedo와 BoatA, BoatB, BoatC의 위치가 일치하는지 확인하고 일치하면 각각 삭제한다.

checkHit {

1) BoatA, BoatB, BoatC를 모두 Boat 타입 포인터 변수 ptr에 저장한다.

2) ptr+i와 torpedo+j의 위치가 일치한다면 ptr+i와 torpedo+j를 삭제한다.

(0 <= i <= 남은 보트의 개수, 0<= j <= 발사한 Torpedo의 개수)

}

(5) 승패 조건을 검사하여 승패 결과를 출력하고 승패가 결정되지 않았다면 (4)로 돌아간다.

1) 남은 보트의 개수가 0 이하라면 승리

2) 남은 턴의 수가 0이하 또는 발사 가능한 어뢰의 수가 0이하라면 패배

3) 모두 해당하지 않는다면 user->move()로 되돌아간다.

(6) 게임을 종료한다.

1-1) y 입력: 게임을 재실행한다.

1-2) n 입력: 메인 프로그램을 종료한다.

1-3) 재입력 받는다.

3. 클래스 구성

(1) Game

멤버 변수

1) SCORE: 점수를 상수화한 것.

2) user: 사용자를 의미하는 Boat 형 포인터 변수. (보트의 수는 100을 넘지 않는다고 가정)

3) b\_ptr[100]: BoatA, BoatB, BoatC를 통합적으로 관리하는 Boat형 포인터 변수.

4) boatAList, boatBList, boatCList: 각각 BoatA, BoatB, BoatC를 관리하는 BoatAList, BoatBList, BoatCList형 포인터 변수.

5) t\_ptr[100]: Torpedo를 관리하는 Torpedo형 포인터 변수. (어뢰의 수는 300을 넘지 않는다고 가정)

6) n\_trun: 남은 턴 수

7) init\_boat, init\_boatA, init\_boatB, init\_boatC, init\_torpedo: 처음 결정된 총 보트, BoatA, BoatB, BoatC, 발사 가능한 어뢰의 개수.

8) n\_boat, n\_boatA, n\_boatB, n\_boatC, n\_torpedo: 남아있는 총 보트, BoatA, BoatB, BoatC, 발사 가능한 어뢰의 개수.

9) indexOfTorpedo: 발사된 어뢰의 인덱스.

10) score: 합산된 점수.

11) boundX, boundY: 처음 결정된 바다 너비, 바다 길이.

12) again: 게임을 다시 시작하는 경우 true.

메소드

1) Game(): 생성자

2) bool isOverlapped(): 생성된 보트의 위치가 겹치는지 확인한다.

3) void checkHit(): 보트와 어뢰의 위치가 겹치는지 확인하고 겹치는 보트와 어뢰를 삭제한다.

4) void show\_initMenu(): 메인 메뉴를 출력하고 입력 값에 따라 게임 옵션을 설정한다.

5) bool make\_boats(): user를 인스턴스화, 보트들을 인스턴스화하여 종류별로 리스트에 추가, 어뢰는 리스트만 생성한다.

6) void shot\_torpedo(): 어뢰를 인스턴스화하여 리스트에 추가한다.

7) void show\_result(): 게임 결과를 출력하고 게임 재시작 여부를 입력받는다.

8) void show\_state(): 게임 진행 상태와 바다에 떠있는 user, 보트들, 어뢰들을 출력한다.

9) void play\_game(): 게임 전체를 순서에 따라 진행한다.

(2) Object

멤버 변수

1) x, y: 현재 객체의 좌표 상의 위치.

2) boundX, boundY: 현재 객체가 이동할 수 있는 좌표 상의 상한선

메소드

protected:

1) void moveUp(): y좌표 상의 위치를 1만큼 감소시킨다. (출력 시 한 칸 위에 출력)

2) void moveDown(): y좌표 상의 위치를 1만큼 증가시킨다. (출력 시 한 칸 아래에 출력)

public:

1) Object(), Object(int startX, int startY): 기본 생성자와 시작 위치를 매개변수로 하는 생성자.

2) ~Object(): 소멸자.

3) void moveLeft(): x좌표 상의 위치를 1만큼 감소시킨다. (출력 시 한 칸 왼쪽에 출력)

4) void moveRight(): x좌표 상의 위치를 1만큼 증가시킨다. (출력 시 한 칸 오른쪽에 출력)

5) void move(): 하위 클래스에서 움직임에 대해 구현할 빈 메소드.

6) void getShape(): 하위 클래스에서 출력될 모양에 대해 구현할 빈 메소드.

7) void collide(Object\* p): 매개변수 객체와 위치가 겹치는지 확인한다.

8) get & set 메소드: 멤버 변수에 대한 get & set 메소드.

(3) Boat

Boat는 Object를 상속받는다.

멤버 변수

1) hasShot: user가 어뢰 발사 시 true.

2) crtX, crtY: BoatC에서 현재 위치를 임시 저장하는 변수.

메소드

1) Boat(), Boat(int startX, int startY): 기본 생성자와 시작 위치를 매개변수로 하는 생성자.

2) ~Boat(): 소멸자

3) bool visibility(): BoatC를 잠수시키거나 밖으로 나오게 하는 메소드.

4) bool get\_hasShot(): 사용자가 어뢰 발사 키 입력 시 true.

5) void move(): 사용자 입력을 받아 user를 좌우로 움직이거나 어뢰 발사가 가능한 상태로 만들 수 있다.

6) char getShape(): user의 출력 모양을 ‘U’로 구현한다.

(4) Torpedo

Torpedo는 Object를 상속받는다.

메소드

1) Torpedo(), Torpedo(int startX, int startY): 기본 생성자와 시작 위치를 매개변수로 하는 생성자.

2) ~Torpedo(): 소멸자

3) void move(): 어뢰를 위로 움직이게 한다.

4) bool get\_hasShot(): 사용자가 어뢰 발사 키 입력 시 true.

5) char getShape(): 어뢰의 출력 모양을 ‘T’로 구현한다.

(5) BoatA

BoatA는 Boat를 상속받는다.

멤버 변수

1) next: BoatAList에서 다음 BoatA를 가리키는 BoatA형 포인터 변수.

2) standardX: 움직임의 기준이 되는 위치.

3) fromLeft: 좌우 움직임의 기준이 되는 불리언 변수.

메소드

1) BoatA(), BoatA(int startX, int startY): 기본 생성자와 시작 위치를 매개변수로 하는 생성자. next를 NULL로, standardX를 startX로, fromLeft는 랜덤으로 설정한다. startX에는 0~1의 variation을 준다.

2) ~BoatA(): 소멸자

3) get\_next(): next를 반환하는 메소드.

4) set\_next(BoatA \* next): next를 설정하는 메소드.

5) void move(): standardX와 fromLeft를 기준으로 BoatA를 좌우로 움직일 수 있다.

6) char getShape(): BoatA의 출력 모양을 ‘A’로 구현한다.

(6) BoatB

BoatB는 Boat를 상속받는다.

멤버 변수

1) next: BoatBList에서 다음 BoatB를 가리키는 BoatB형 포인터 변수.

2) standardX, standardY: 움직임의 기준이 되는 위치.

메소드

1) BoatB(), BoatB(int startX, int startY): 기본 생성자와 시작 위치를 매개변수로 하는 생성자. next를 NULL로, standardX(Y)를 startX(Y)+0.5로 설정한다. startX(Y)에는 0~1의 variation을 준다.

2) ~BoatB(): 소멸자

3) get\_next(): next를 반환하는 메소드.

4) set\_next(BoatB \* next): next를 설정하는 메소드.

5) void move(): standardX, standardY를 기준으로 BoatB를 좌우상하로 움직일 수 있다.

6) char getShape(): BoatB의 출력 모양을 ‘B’로 구현한다.

(7) BoatC

BoatC는 Boat를 상속받는다.

멤버 변수

1) next: BoatCList에서 다음 BoatC를 가리키는 BoatC형 포인터 변수.

2) loopCnt: visibility() 실행의 기준이 될 변수.

3) variation: 랜덤으로 시작 시점을 결정할 때 사용되는 변수.

메소드

1) BoatC(), BoatC(int startX, int startY): 기본 생성자와 시작 위치를 매개변수로 하는 생성자. next를 NULL로, variation은 0~2 사이의 랜덤 값으로 설정한다.

2) ~BoatC(): 소멸자

3) get\_next(): next를 반환하는 메소드.

4) set\_next(BoatC \* next): next를 설정하는 메소드.

5) set\_loopCnt(int loopCnt): 게임의 loop 횟수를 입력받아 저장하는 메소드.

5) void move(): loopCnt+variation을 기준으로 BoatC를 잠수시키거나 밖으로 나오게 할 수 있다.

6) char getShape(): BoatC의 출력 모양을 ‘C’로 구현한다.

(8) BoatAList, BoatBList, BoatCList

멤버 변수

1) head: 리스트의 첫번째 객체를 가리키는 BoatA(B, C)형 포인터 변수.

2) size: 리스트에 있는 객체의 개수.

메소드

1) BoatA(B, C)List(): 매개변수가 없는 생성자. head를 NULL로, size를 0으로 설정한다.

2) get & set 메소드: 멤버 변수에 대한 get & set 메소드.

3) void add(BoatA(B,C) \* boatA(B,C)): 리스트에 객체를 추가하는 메소드.

Ⅲ. 프로그램 실행 방법

본 프로그램은 MacOS의 터미널에서 실행된다. Makefile의 내용은 다음과 같다.

① main.cpp 파일을 제외한 나머지 10개 .cpp 파일을 컴파일하고 각각의 .hpp 파일과 링킹하여 각각의 .o 파일을 생성한다.

② main.cpp 파일을 컴파일하고 6개의 .hpp 파일과 링킹하여 main.o 파일을 생성한다.

③ main.o와 10개의 .o 파일을 링킹하여 assn2 실행 파일을 생성한다.

(1) 터미널을 실행한다.

(2) 소스 파일과 Makefile이 모두 있는 디렉토리로 이동한다.

$cd <디렉토리명>

(3) 터미널 입력 라인에서 “make”를 입력하면 “make all”과 같은 효과가 나타나서 assn2 실행 파일이 생성된다.

$make

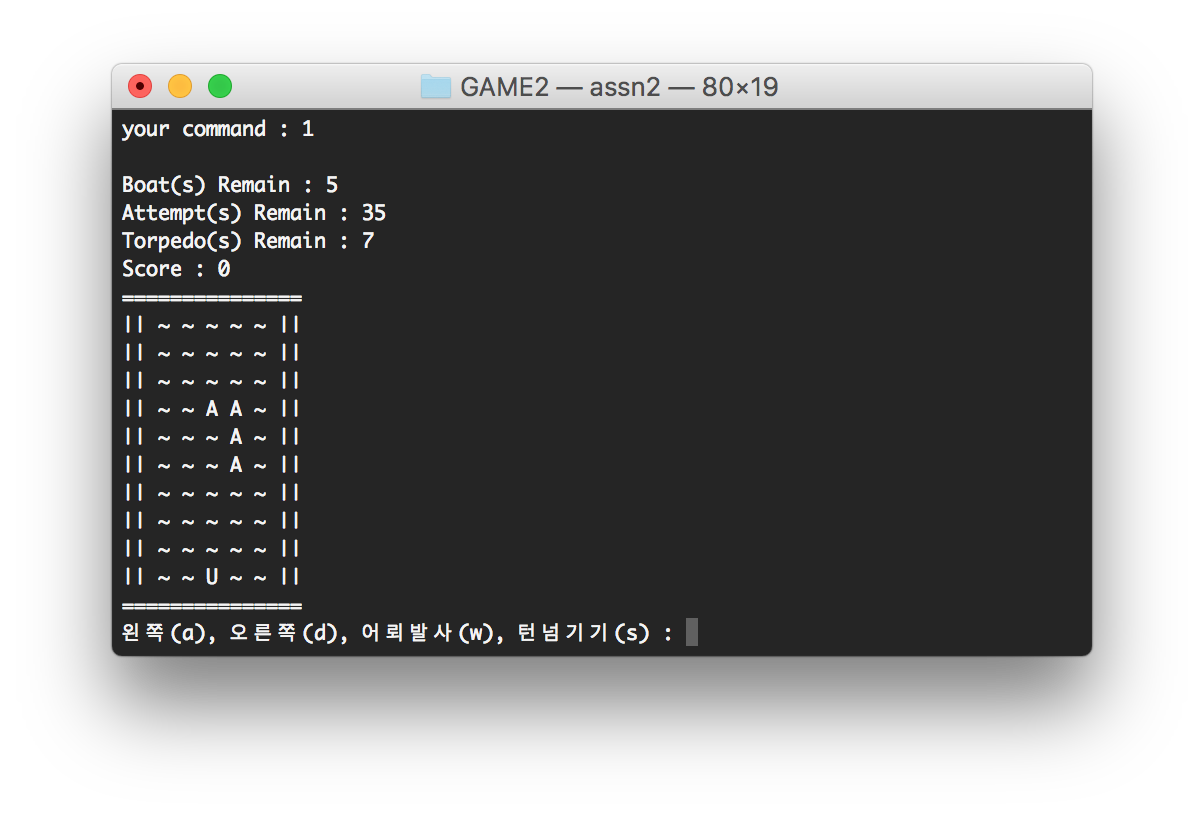
(4) assn2 를 실행한다.

$./assn2

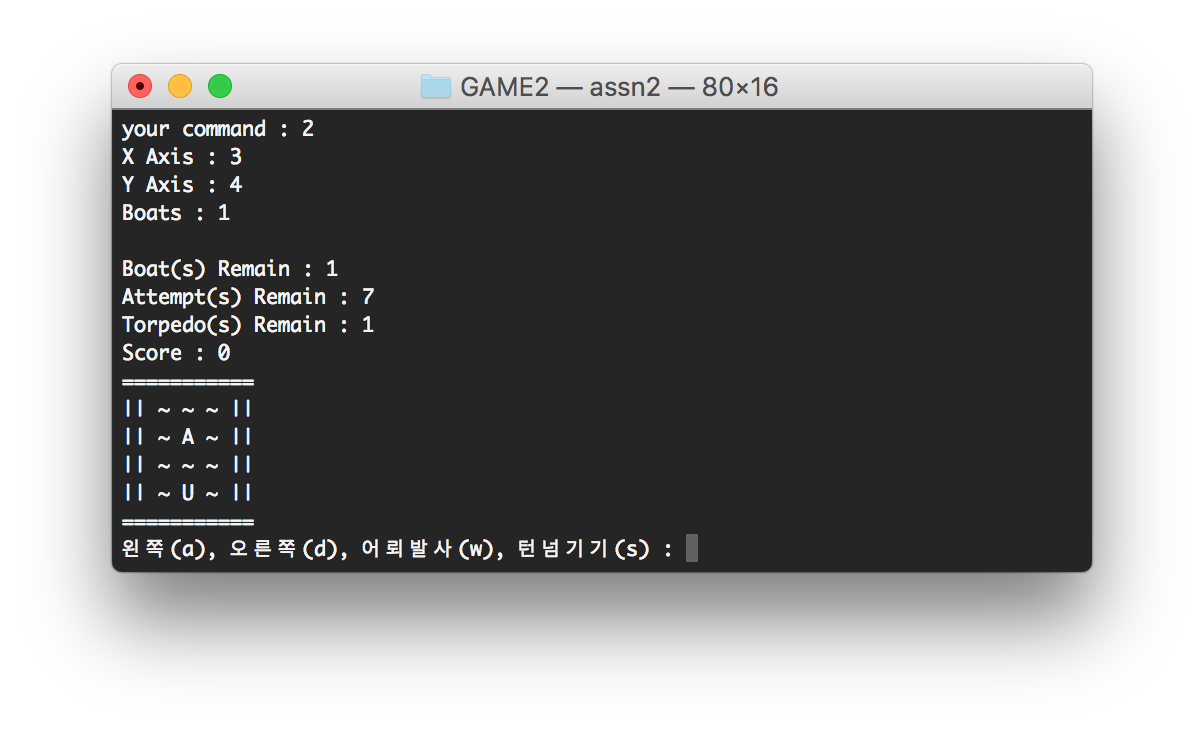
Ⅳ. 예제 및 실행 결과

0. 메인 프로그램 시작

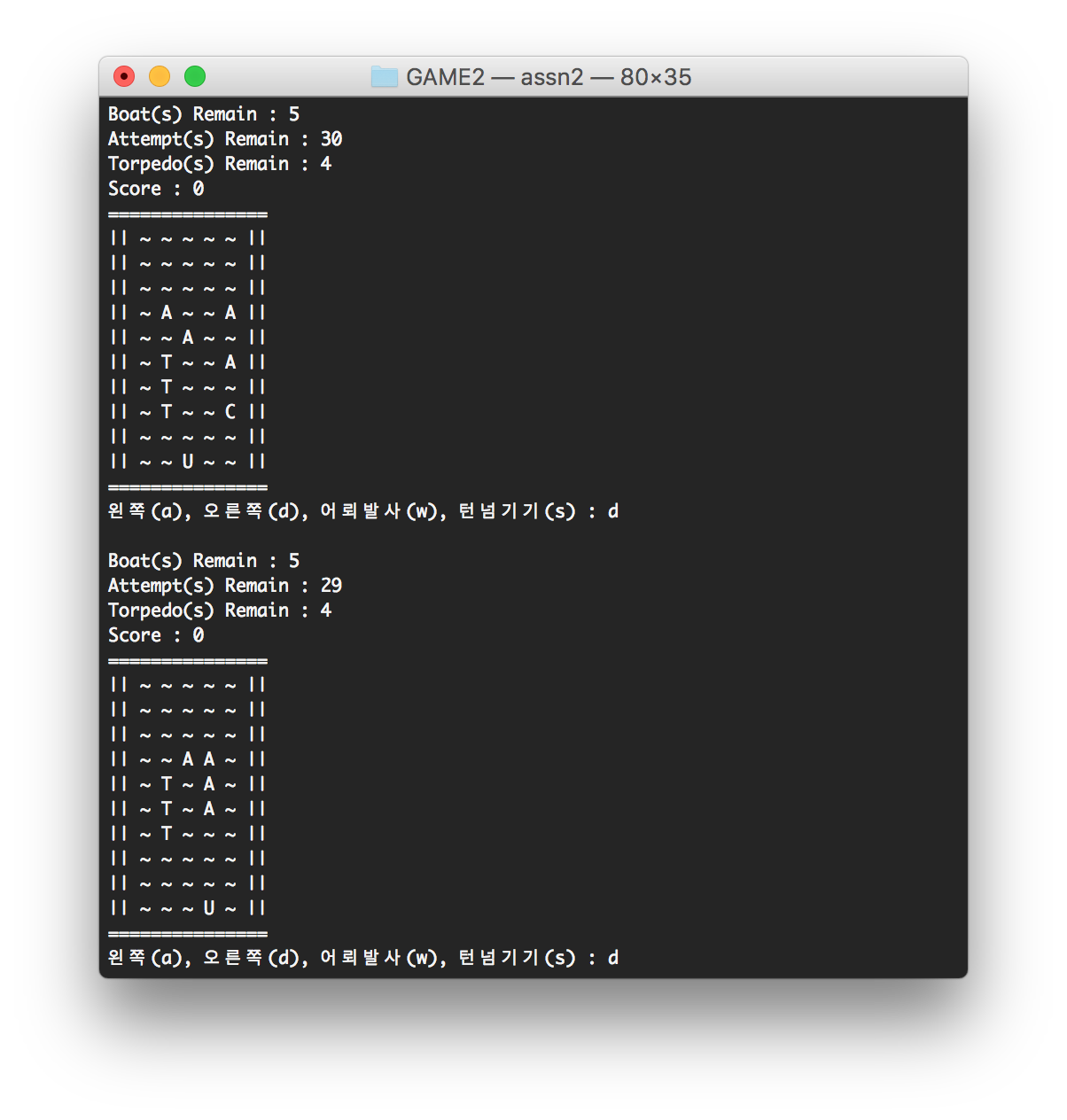
1-1. Default game



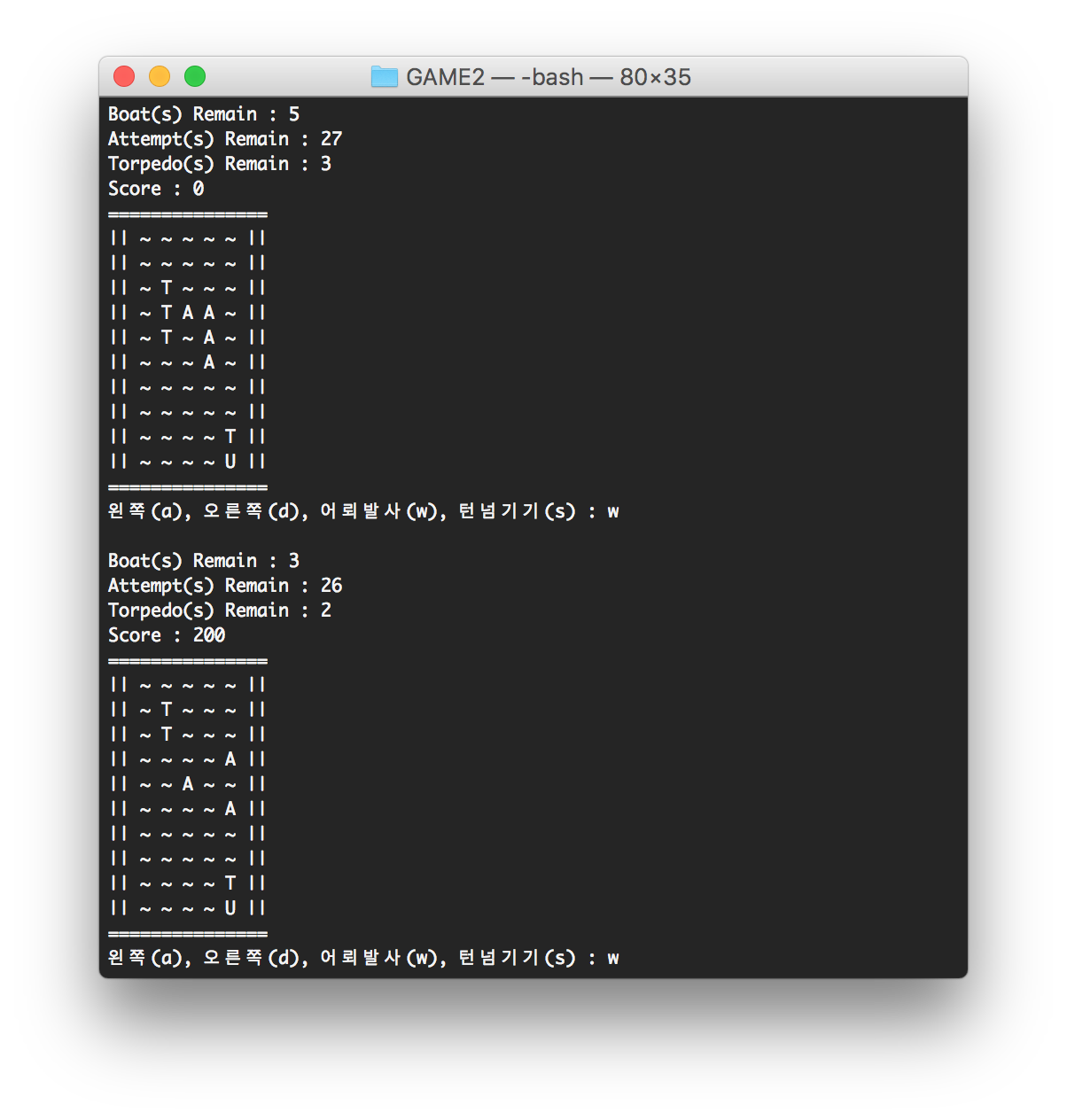
1-2. Custom game



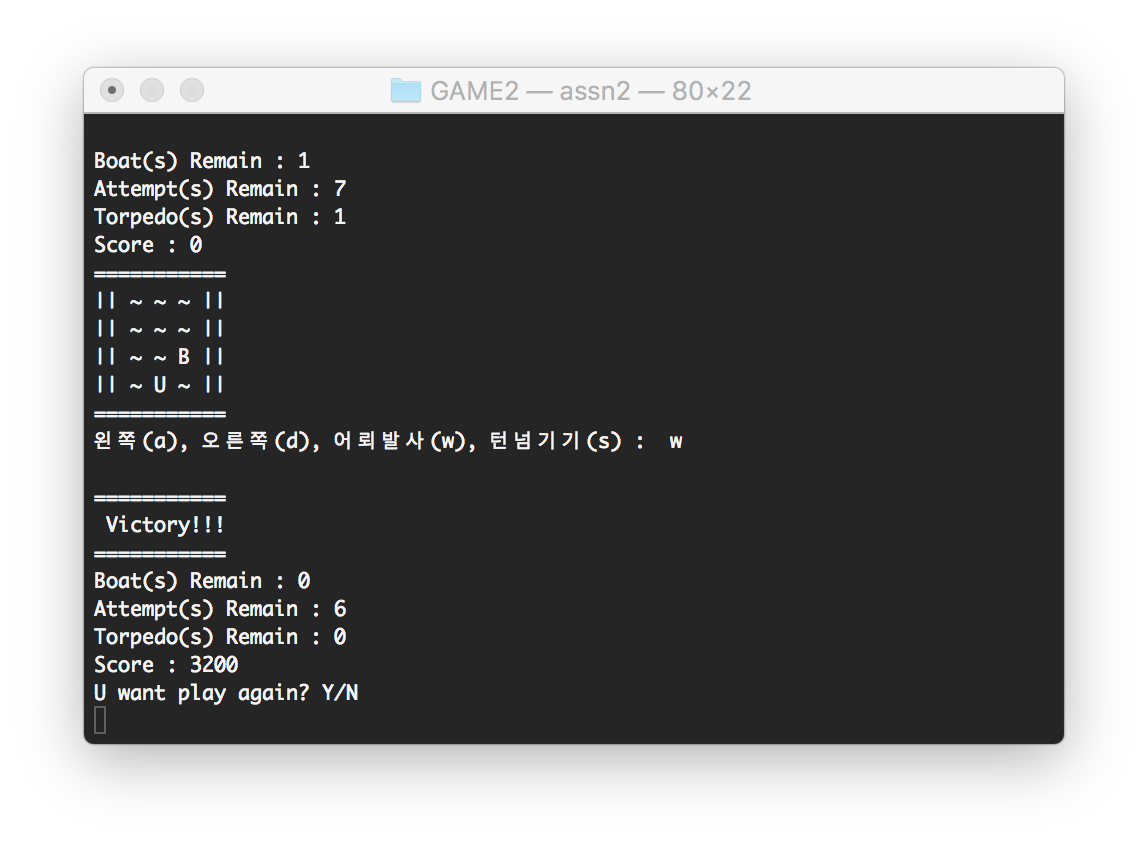
2-1. 게임 실행 화면 (User 오른쪽 이동, BoatA 이동, BoatC 잠수)



2-2. 게임 실행 화면 (User 어뢰 발사, BoatA, BoatC 1개씩 격추)



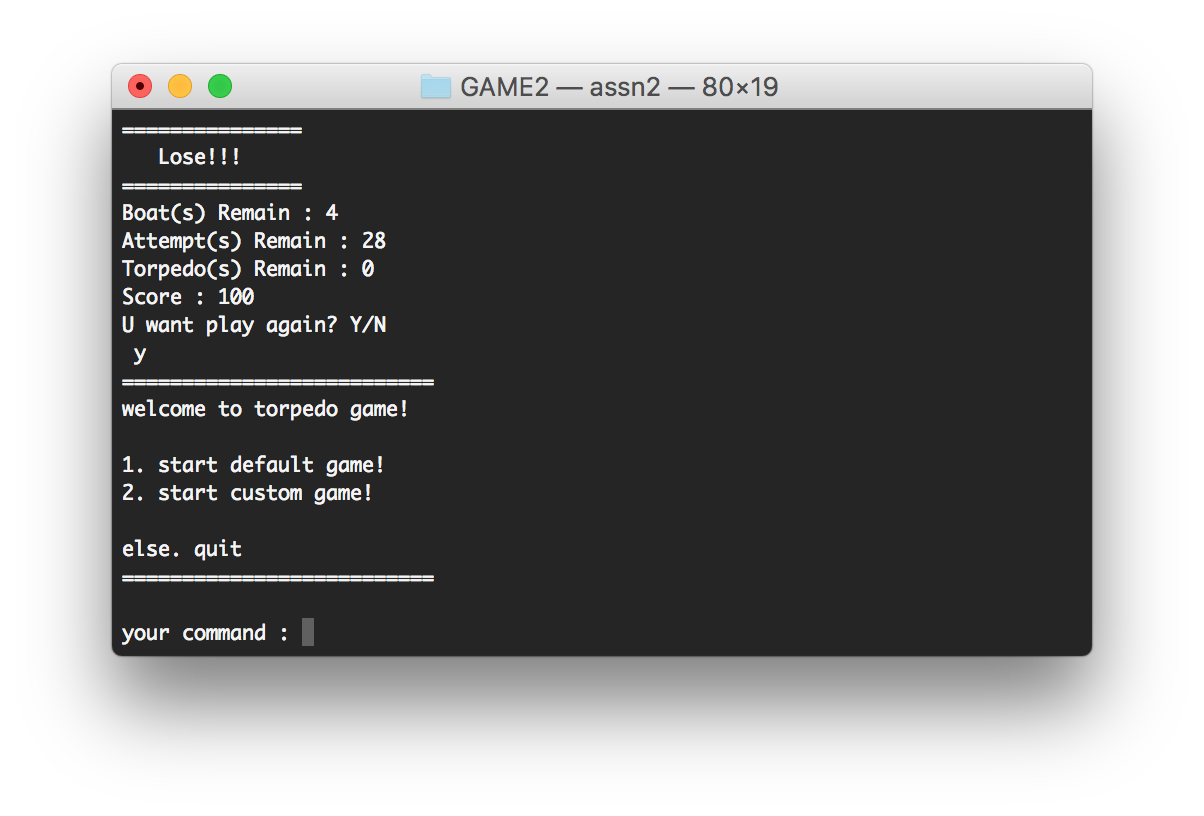
3-1. 게임 결과 (승리)



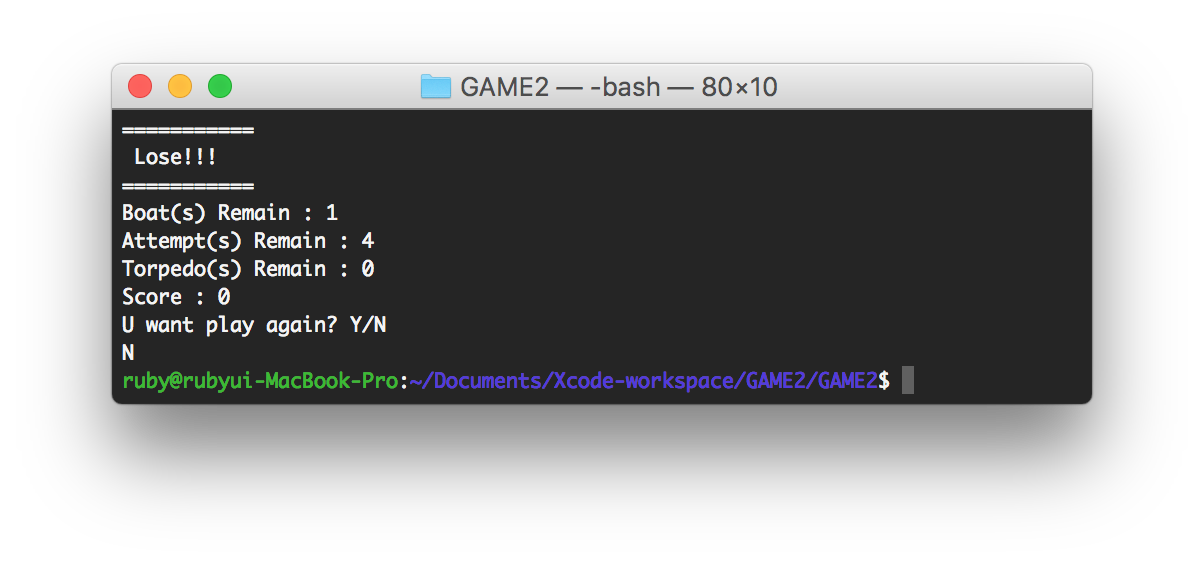
3-1. 게임 결과 (패배)



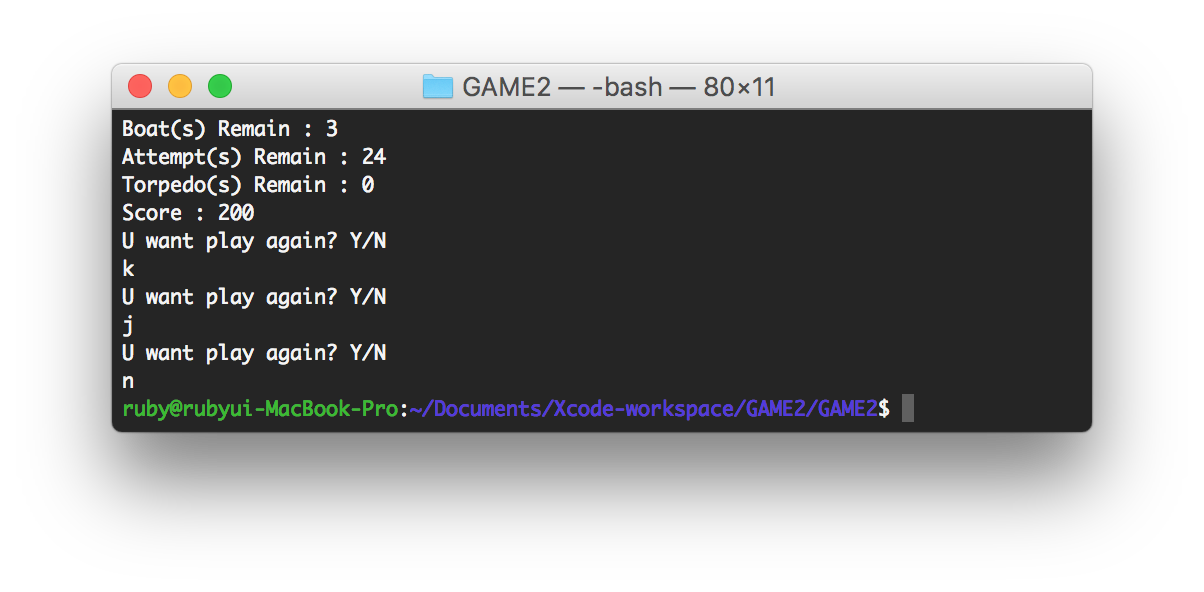
4-1. 게임 재시작



4-2. 게임 종료



4-3. 유효하지 않은 키 값 처리



Ⅴ. 실행 결과에 대한 토의

1. 토론

(1) BoatA, BoatB, BoatC를 한꺼번에 관리하기

user 입력을 받기 전 생성된 보트끼리 위치 중복 여부를 검사하거나 어뢰와 보트의 위치 중복(격추) 여부를 확인하기 위해 모든 조합(Combination)을 확인하는 반복문이 필요하다. 그런데 포인터 타입이 다른 BoatA, BoatB, BoatC를 따로 관리하면 여러 조합의 반복문을 작성해야 하므로 공통의 부모 클래스 타입인 Boat 형 포인터 변수로 한꺼번에 관리하였다.

(2) Boat가 Torpedo에 맞췄는지 검사하는 메소드 작성 시 발생한 문제 해결

// 수정 전 코드

void Game::checkHit() {

    for (int i = 0; i < n\_boat; i++) {

        for (int j = 0; j <= indexOfTorpedo; j++) {

            if(b\_ptr[i]->collide(t\_ptr[j])) {

                b\_ptr[i]->setX(-200); b\_ptr[i]->setY(-200);

                n\_boat--;

                if (i < n\_boatA)

                    n\_boatA--;

                else if (i < n\_boatA+n\_boatB)

                    n\_boatB--;

                else

                    n\_boatC--;

                t\_ptr[j]->setX(-100); t\_ptr[j]->setY(-100);

            }

        }

    }

}

위의 메소드는 1)모든 보트들에 대하여 2)모든 발사한 어뢰 중 위치 좌표가 일치하는 것이 있다면 맞춘 것으로 간주하여 해당 보트와 어뢰를 삭제하는 메소드이다.

해당 메소드에 대한 테스트 중 처음 격추는 성공하지만, 그 이후에는 위치 좌표가 동일해도 격추가 이루어지지 않는 문제가 발생하였다. 이때 격추 시 바로 n\_boat(남아있는 보트의 수)를 감소시키는 것이 문제 원인이었다. n\_boat이 반복문의 조건식에 포함되어있기 때문에 n\_boat를 바로 감소시키면 반복문에서 모든 보트를 검사하지 않게 된다. 따라서 처음에 생성된 n\_boat의 값을 저장하는 init\_boat를 만들어 조건식에 사용하였다.

2. 결론

(1) 소스 파일 분리와 클래스 작성의 연관성

ASSN1 작성 시 1)클래스마다 존재 이유와 기능이 명확히 구분되며 2)코드의 양이 많았기 때문에 소스 파일 분리가 필수적이었다. 그런데 본 프로그램 작성 시 BoatA(B, C)List 클래스는 코드 내용이 너무 적어 소스 파일 분리가 무의미해 보였다.

그런데 완성된 코드를 검토해보니 BoatA(B, C)List 클래스가 불필요하며 BoatA(B, C)만으로도 리스트를 구현할 수 있다는 것을 알게 되었다. 만약 소스 파일을 분리하지 않았다면 이러한 새로운 클래스 타입이 불필요하다는 점을 찾지 못했을 것이다.

(2) 상속의 유용함

상위 클래스에 공통되는 멤버변수, 메소드가 존재하므로 하위 클래스에서 작성해야 할 코드의 양이 줄었다. 또한 Object라는 최상위 클래스가 하위 클래스 객체의 절대 위치 좌표를 갖고 있기 때문에 하위 클래스에서 새로운 기준의 위치 좌표를 도입하여 move() 작성 시 이용할 수 있었다.

(3) ReadMe 문서의 유용함

본 프로그램의 컴파일 및 실행파일 생성은 터미널에서 makefile을 통해 이루어져야 한다. 그런데 통합개발도구에서 실행하는 것에 익숙하여 makefile 사용법이 기억나지 않았다. 이때 Assn1ReadMe 문서에 이용법을 기록해두었기 때문에 문서를 보고 바로 makefile을 이용할 수 있었다.

3. 개선 방향

(1) BoatA(B, C)List 클래스없이 BoatA(B, C) 리스트 관리

BoatA(B, C)List 클래스에는 add 기능 하나만 있기 때문에 클래스를 따로 작성하지 않고 BoatA(B, C) 클래스에 필요한 멤버 변수와 메소드를 추가하는 것이 바람직할 것이다.

(2) BoatA(B, C)List 클래스 개선

BoatA(B, C)List 클래스를 없애지 않고 del 기능을 추가한다면 리스트에서 격추된 보트 객체를 바로바로 삭제하고 할당해제 할 수 있다. BoatA(B, C)List 클래스가 이러한 복합적인 기능을 구현한다면 단일 메소드가 아닌 클래스로써의 존재 가치가 충분할 것이다.