МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «ООП»
Тема: Полиморфизм.

Студентка гр. 3385 _____ Тараканова А.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы.

Изучить принципы полиморфизма в объектно-ориентированном программировании (ООП) на примере реализации игровых способностей для игры «Морской бой».

Задание.

a. Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:

i. Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).

ii. Сканер - позволяет проверить участок поля 2x2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.

iii. Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.

b. Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.

c. Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.

d. Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):

. Попытка применить способность, когда их нет

i. Размещение корабля вплотную или на пересечении с другим кораблем

ii. Атака за границы поля

- **Примечания:**
- Интерфейс события должен быть унифицирован, чтобы их можно было единообразно использовать через интерфейс
- Не должно быть явных проверок на тип данных

Выполнение работы.

Класс Ability (интерфейс)

Назначение: определяет общий контракт для всех игровых способностей. Позволяет использовать полиморфизм для применения различных способностей персонажа.

Реализация: имеет два базовых метода, один для печати названия способности и второй для ее применения, который возвращает булевое значение в зависимости от того был ли корабль уничтожен в ходе использования способности.

Класс DoubleDamage

Назначение: представляет способность, которая удваивает урон, наносимый цели.

Реализация: после прохождения необходимых проверок дважды вызывает атаку по выбранным координатам.

Связи: наследует интерфейс Ability, имеет связь с AbilityManager, принимает объект класса GameStatus за счет чего имеет связь с полем и менеджером кораблей.

Класс ScannerAbility

Назначение: представляет способность, которая проверяет определённую область на наличие врагов или объектов.

Реализация: проверяет квадрат 2*2 на наличие кораблей при помощи проверки состояния клетки поля

Связи: наследует интерфейс Ability, имеет связь с AbilityManager, принимает объект класса GameStatus за счет чего имеет связь с полем и менеджером кораблей.

Класс BarrageAbility

Назначение: представляет способность, которая наносит урон случайному сегменту случайного корабля.

Реализация: выбирает случайный корабль, если он не уничтожен, выбирает в нем случайный сегмент, если он не уничтожен, производит выстрел.

Связи: наследует интерфейс Ability, имеет связь с AbilityManager, принимает объект класса GameStatus за счет чего имеет связь с полем и менеджером кораблей.

Класс AbilityManager

Управляет доступными способностями персонажа, применяет их и обновляет список способностей по мере уничтожения кораблей.

Реализация: два вектора- один для хранения способностей, доступных игроку; второй с перечнем всех возможных способностей, необходимый для получения новой случайной способности. Генератор.

AbilityManager::AbilityManager() конструктор

void AbilityManager::initializeAbilityPool() инициализирует вектор всех возможных способностей.

void AbilityManager::applyFirstAvailableAbility(GameStatus& status) позволяет применить первую из доступных способностей и добавить новую в случае, когда был уничтожен корабль, также выводит сообщение о невозможности применить способность или их отсутствии.

void AbilityManager::addAbility() добавляет случайную из доступных возможностей, для чего инициализирует пул способностей, после чего при помощи генератора случайных чисел добавляет способность, пул очищается.

Связи: содержит список объектов типа Ability, имеет связь с классом GameStatus

Класс GameStatus

Назначение: хранение информации об игре

Реализация: указатель на поле и менеджер кораблей.

Связи: поле и менеджер кораблей.

Исключения

Класс GameFieldException необходим для отлавливания ошибок, связанных с классом поля.

Класс ShipManagerException необходим для отлавливания ошибок, связанных с классами менеджера кораблей и самих кораблей.

Класс AbilityManagerException необходим для отлавливания ошибок связанных с классом способностей.

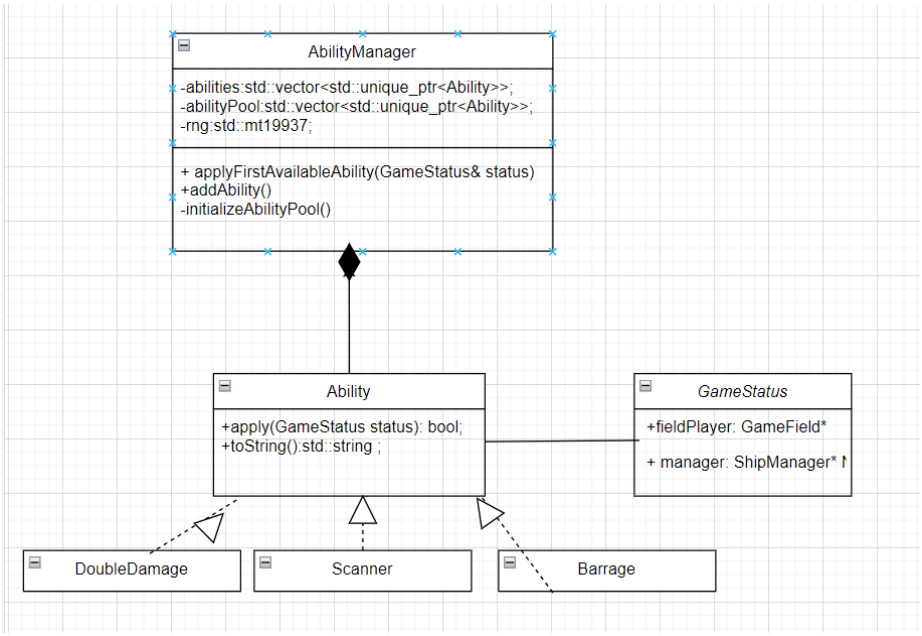

Рис.1 — UML-диаграмма классов

Выводы.

В ходе выполнения лабораторной работы были изучены принципы полиморфизма в объектно-ориентированном программировании (ООП) на примере реализации игровых способностей для игры «Морской бой».

.

Приложение

ShipSegment.h

```
#IFNDEF SHIPSEGMENT_H
#DEFINE SHIPSEGMENT_H
#INCLUDE <IOSTREAM>

ENUM CLASS SEGMENTSTATUS{
    WHOLE,
    DAMAGE,
    DESTROYED
};

CLASS SHIPSEGMENT{
PRIVATE:
    SEGMENTSTATUS STATUS;
PUBLIC:
    SHIPSEGMENT() : STATUS(SEGMENTSTATUS::WHOLE){}
    VOID SETSTATUS(SEGMENTSTATUS SEGMENT);
    SEGMENTSTATUS GETSTATUS();
    VOID SHOT();

};

#ENDIF
```

ShipSegment.cpp

```
#INCLUDE "SHIPSEGMENT.H"
VOID SHIPSEGMENT::SHOT(){
        IF (STATUS == SEGMENTSTATUS::WHOLE) {
            STATUS = SEGMENTSTATUS::DAMAGE;
            STD::COUT << "DAMAGE"<< STD::ENDL;
        } ELSE IF (STATUS == SEGMENTSTATUS::DAMAGE) {
            STATUS = SEGMENTSTATUS::DESTROYED;
            STD::COUT << "DESTROYED" << STD::ENDL;
        }
        ELSE{
            STD::COUT<<"ПОВТОРНОЕ УНИЧТОЖЕНИЕ!" << STD::ENDL;
        }
    }

VOID SHIPSEGMENT::SETSTATUS(SEGMENTSTATUS SEGMENT) {
    STATUS = SEGMENT; // УСТАНАВЛИВАЕМ СТАТУС СЕГМЕНТА
}

SEGMENTSTATUS SHIPSEGMENT::GETSTATUS() {
    RETURN STATUS; // ВОЗВРАЩАЕМ ТЕКУЩИЙ СТАТУС СЕГМЕНТА
}
```

Ship.h

```
      //КОРАБЛИК
      #IFNDEF SHIP_H
      #DEFINE SHIP_H
      #INCLUDE "SHIPSEGMENT.H"

      CLASS SHIP{
      PRIVATE:
          SHIPSEGMENT* SEGMENTS;
          INT LEN ;
          BOOL VERTICAL;

      PUBLIC:
          SHIP(INT LEN, BOOL VERTICAL);
          ~SHIP();
```

```
        VOID SHOTSEGMENT(INT INDEX);
        VOID SETVERTICAL(BOOL VERTICAL);
        SEGMENTSTATUS GETSTATUS(INT INDEX) CONST;
        INT GETLENGTH() CONST;
        BOOL GETORIENTATION() CONST;
        BOOL ISDESTROYED() CONST;
    };

    #ENDIF
```

Ship.cpp

```
#INCLUDE "SHIP.H"
#INCLUDE "CUSTOMEXCEPTIONS.H"

SHIP::SHIP (INT LEN, BOOL VERTICAL) : LEN(LEN), VERTICAL(VERTICAL){
    IF(LEN < 1 || LEN > 4){
        THROW SHIPMANAGEREXCEPTION("НЕВОЗМОЖНАЯ ДЛИНА");
    }
    SEGMENTS = NEW SHIPSEGMENT[LEN];
}

SHIP::~SHIP(){
    DELETE[] SEGMENTS;
}

VOID SHIP::SHOTSEGMENT(INT INDEX){
    IF(INDEX<0 || INDEX>LEN)
        THROW SHIPMANAGEREXCEPTION("НЕКОРРЕКТНЫЙ ИНДЕКС");
    SEGMENTS[INDEX].SHOT();
}

SEGMENTSTATUS SHIP::GETSTATUS(INT INDEX) CONST {
    IF(INDEX<0 || INDEX>LEN)
        THROW SHIPMANAGEREXCEPTION("НЕКОРРЕКТНЫЙ ИНДЕКС");
    RETURN SEGMENTS[INDEX].GETSTATUS();
}

INT SHIP::GETLENGTH() CONST {
    RETURN LEN;
}

VOID SHIP::SETVERTICAL(BOOL VERTICAL){
    VERTICAL=VERTICAL;
}

BOOL SHIP::GETORIENTATION() CONST {
    RETURN VERTICAL;
}

BOOL SHIP::ISDESTROYED() CONST {
    FOR (INT I = 0; I < LEN; I++) {
        IF (SEGMENTS[I].GETSTATUS() != SEGMENTSTATUS::DESTROYED) {
            RETURN FALSE;
        }
    }
    RETURN TRUE;
}
```

ShipManager.h

```
    //МЕНЕДЖЕР
    #IFNDEF SHIP_MANAGER_H
    #DEFINE SHIP_MANAGER_H

    #INCLUDE "SHIP.H"
    #INCLUDE <VECTOR>
```

```
      CLASS SHIPMANAGER{
      PRIVATE:
          STD::VECTOR <SHIP*> SHIPS;
      PUBLIC:
          SHIPMANAGER(CONST         STD::VECTOR<STD::PAIR<INT,         INT>>&
SHIPSPECIFICATIONS);
          ~SHIPMANAGER();

          INT GETSHIPSCOUNT() CONST ;
          SHIP* GETSHIP(INT INDEX) CONST;
          BOOL ISDESTROYED();
      };

      #ENDIF
```

ShipManager.cpp

```
#INCLUDE "SHIPMANAGER.H"

SHIPMANAGER::SHIPMANAGER(CONST       STD::VECTOR<STD::PAIR<INT,       INT>>&
SHIPSPECIFICATIONS) {
    FOR (CONST AUTO& DATA : SHIPSPECIFICATIONS) {
        INT LEN = DATA.FIRST;
        INT COUNT = DATA.SECOND;
        FOR (INT I = 0; I < COUNT; I++) {
            TRY {
                SHIP* SHIP = NEW SHIP(LEN,TRUE);
                SHIPS.PUSH_BACK(SHIP); // ПО УМОЛЧАНИЮ ВЕРТИКАЛЬНЫЕ
            } CATCH (STD::INVALID_ARGUMENT& E) {
                STD::CERR << E.WHAT() << STD::ENDL;
            }
        }
    }
}

SHIPMANAGER::~SHIPMANAGER() {
    FOR (SHIP* SHIP : SHIPS) {
        DELETE SHIP;
    }
}

SHIP* SHIPMANAGER::GETSHIP(INT INDEX) CONST {
    IF (INDEX >= 0 && INDEX < SHIPS.SIZE()) {
        RETURN SHIPS[INDEX]; // ВОЗВРАЩАЕТ УКАЗАТЕЛЬ НА КОРАБЛЬ
    }
    RETURN NULLPTR; // ЕСЛИ ИНДЕКС ВНЕ ДИАПАЗОНА, ВОЗВРАЩАЕМ NULLPTR
}

INT SHIPMANAGER::GETSHIPSCOUNT() CONST {
    RETURN SHIPS.SIZE(); // ВОЗВРАЩАЕТ КОЛИЧЕСТВО КОРАБЛЕЙ
}

BOOL SHIPMANAGER::ISDESTROYED(){
    FOR(SHIP* SHIP : SHIPS){
        IF(!SHIP->ISDESTROYED())
            RETURN FALSE;
    }
    RETURN TRUE;
}
```

GameField.h

```
#IFNDEF GAME_FIELD_H
#DEFINE GAME_FIELD_H

#INCLUDE "SHIPMANAGER.H"
```

```
#INCLUDE <MAP>
#INCLUDE <STDEXCEPT>
#INCLUDE <IOSTREAM>

ENUM CLASS CELLSTATUS {
    UNKNOWN, // НЕИЗВЕСТНО
    EMPTY,   // СВОБОДНО
    SHIP     // ЗАНЯТО
};

CLASS GAMEFIELD {
PRIVATE:
    CELLSTATUS** FIELD; // КЛЕТКИ
    INT WIDTH;
    INT HEIGHT;
    STD::MAP<STD::PAIR<INT, INT>, SHIP*> SHIPPOSITIONS;
    STATIC GAMEFIELD* INSTANCE_;

    VOID ALLOCATEMEMORY(INT W, INT H);
    VOID DEALLOCATEMEMORY();

PUBLIC:
    GAMEFIELD(INT WIDTH, INT HEIGHT);
    ~GAMEFIELD();
    GAMEFIELD(CONST GAMEFIELD& OTHER);
    GAMEFIELD& OPERATOR=(CONST GAMEFIELD& OTHER);  //КОПИРОВАНИЕ
    GAMEFIELD(GAMEFIELD&& OTHER)NOEXCEPT;
    GAMEFIELD& OPERATOR=(GAMEFIELD&& OTHER) NOEXCEPT; //ПЕРЕМЕЩЕНИЕ
    VOID PLACESHIP(SHIP* SHIP, INT X, INT Y);
    BOOL ATTACKCELL(INT X, INT Y);
    VOID PRINTFIELD() CONST;
    BOOL CHEKSHIP(INT X, INT Y, INT LEN, BOOL VERTICAL);
    CELLSTATUS GETSTATUS(INT X , INT Y) CONST; // МЕТОД ПОЛУЧЕНИЯ СТАТУСА
КЛЕТКИ
};

#ENDIF
```

GameField.cpp

```
#INCLUDE "GAMEFIELD.H"
#INCLUDE "CUSTOMEXCEPTIONS.H"

VOID GAMEFIELD::ALLOCATEMEMORY(INT W, INT H) {
    FIELD = NEW CELLSTATUS*[H];
    FOR (INT I = 0; I < H; ++I) {
        FIELD[I] = NEW CELLSTATUS[W];
        STD::FILL(FIELD[I],   FIELD[I]   +   W,   CELLSTATUS::UNKNOWN);   //
ИНИЦИАЛИЗАЦИЯ СТАТУСА КЛЕТОК
    }
}

VOID GAMEFIELD::DEALLOCATEMEMORY() {
    FOR (INT I = 0; I < HEIGHT; ++I) {
        DELETE[] FIELD[I];
    }
    DELETE[] FIELD;
}

GAMEFIELD::GAMEFIELD(INT WIDTH, INT HEIGHT) : WIDTH(WIDTH), HEIGHT(HEIGHT) {
    IF (WIDTH <= 0 || HEIGHT <= 0) {
            THROW   STD::INVALID_ARGUMENT("ДЛИНА   И   ШИРИНА   ДОЛЖНЫ   БЫТЬ
ПОЛОЖИТЕЛЬНЫМИ");
    }
    ALLOCATEMEMORY(WIDTH, HEIGHT);
```

```
}


GAMEFIELD::~GAMEFIELD() {
    FOR (INT I = 0; I < HEIGHT; I++) {
        DELETE[] FIELD[I];
    }
    DELETE[] FIELD;
}

GAMEFIELD::GAMEFIELD(CONST   GAMEFIELD&   OTHER)   :   WIDTH(OTHER.WIDTH),
HEIGHT(OTHER.HEIGHT), FIELD(NULLPTR) {
    ALLOCATEMEMORY(OTHER.WIDTH, OTHER.HEIGHT); // ВЫДЕЛЯЕМ ПАМЯТЬ
    FOR (INT I = 0; I < HEIGHT; ++I) {
        STD::COPY(OTHER.FIELD[I],  OTHER.FIELD[I]  +  WIDTH,  FIELD[I]);  //
ГЛУБОКОЕ КОПИРОВАНИЕ СТАТУСА КЛЕТОК
    }
    SHIPPOSITIONS = OTHER.SHIPPOSITIONS; // КОПИРУЕМ ПОЗИЦИИ КОРАБЛЕЙ
}

 GAMEFIELD& GAMEFIELD::OPERATOR=(CONST GAMEFIELD& OTHER) {
    IF (THIS == &OTHER) RETURN *THIS; // ЗАЩИТА ОТ САМОПРИСВАИВАНИЯ

    // ОСВОБОЖДАЕМ ПАМЯТЬ
    DEALLOCATEMEMORY();

    // КОПИРУЕМ ЗНАЧЕНИЯ
    WIDTH = OTHER.WIDTH;
    HEIGHT = OTHER.HEIGHT;

    ALLOCATEMEMORY(OTHER.WIDTH, OTHER.HEIGHT);
    FOR (INT I = 0; I < HEIGHT; ++I) {
        STD::COPY(OTHER.FIELD[I], OTHER.FIELD[I] + WIDTH, FIELD[I]);
    }
    SHIPPOSITIONS = OTHER.SHIPPOSITIONS;

    RETURN *THIS;
}

GAMEFIELD::GAMEFIELD(GAMEFIELD&& OTHER) NOEXCEPT :
    FIELD(OTHER.FIELD),       WIDTH(OTHER.WIDTH),       HEIGHT(OTHER.HEIGHT),
SHIPPOSITIONS(STD::MOVE(OTHER.SHIPPOSITIONS)) {
    OTHER.FIELD = NULLPTR; // СБРОС УКАЗАТЕЛЯ В NULLPTR
    OTHER.WIDTH = 0;
    OTHER.HEIGHT = 0;
}

GAMEFIELD& GAMEFIELD::OPERATOR=(GAMEFIELD&& OTHER) NOEXCEPT {
    IF (THIS == &OTHER) RETURN *THIS; // ЗАЩИТА ОТ САМОПРИСВАИВАНИЯ

    // ОСВОБОЖДАЕМ ПАМЯТЬ
    DEALLOCATEMEMORY();

    // ПЕРЕМЕЩАЕМ ЗНАЧЕНИЯ
    FIELD = OTHER.FIELD;
    WIDTH = OTHER.WIDTH;
    HEIGHT = OTHER.HEIGHT;
    SHIPPOSITIONS = STD::MOVE(OTHER.SHIPPOSITIONS);

    OTHER.FIELD = NULLPTR; // СБРОС УКАЗАТЕЛЯ В NULLPTR
    OTHER.WIDTH = 0;
    OTHER.HEIGHT = 0;
```

```
    RETURN *THIS;
}

BOOL GAMEFIELD :: CHEKSHIP( INT X, INT Y, INT LEN,  BOOL VERTICAL){
    IF(Y >= HEIGHT || Y < 0 || X >= WIDTH || X < 0 )
        RETURN FALSE;
    INT DY, DX;
    IF (VERTICAL) {
        IF(Y + LEN > HEIGHT)
            RETURN FALSE;
    }
    ELSE{
        IF (X + LEN > WIDTH)
            RETURN FALSE;
    }
    FOR (INT I = -1; I <= LEN; I++) {
        FOR (INT J = -1; J <= 1; J++){
            IF (VERTICAL){
                DY = I;
                DX = J;
            }
            ELSE{
                DY = J;
                DX = I;
            }
            IF((Y+DY >= 0) && (X+DX >= 0) && (Y+DY < HEIGHT) && (X+DX <
WIDTH))
                IF (FIELD[Y + DY][X+DX] != CELLSTATUS::UNKNOWN)
                    RETURN FALSE;
        }
    }
    RETURN TRUE;
}

VOID GAMEFIELD::PLACESHIP(SHIP* SHIP, INT X, INT Y) {
    IF (CHEKSHIP(X, Y, SHIP->GETLENGTH(),  SHIP->GETORIENTATION())) {
        FOR (INT I = 0; I < SHIP->GETLENGTH(); I++) {
            IF (SHIP->GETORIENTATION()){
                FIELD[Y + I][X] = CELLSTATUS::SHIP;
                SHIPPOSITIONS.EMPLACE(STD::MAKE_PAIR(X,  Y  +  I),  SHIP);  //
СОХРАНЯЕМ ПОЗИЦИЮ СЕГМЕНТА
            }
            ELSE{
                FIELD[Y][X + I] = CELLSTATUS::SHIP;
                SHIPPOSITIONS.EMPLACE(STD::MAKE_PAIR(X  +  I,  Y),  SHIP);  //
СОХРАНЯЕМ ПОЗИЦИЮ СЕГМЕНТА
            }
        }
    }
    ELSE{
        THROW GAMEFIELDEXCEPTION("НЕВЕРНОЕ РАСПОЛОЖЕНИЕ КОРАБЛЯ");
    }
}

BOOL GAMEFIELD::ATTACKCELL(INT X, INT Y) {
    IF (X < 0 || X >= WIDTH || Y < 0 || Y >= HEIGHT) {
        THROW GAMEFIELDEXCEPTION("ВЫСТРЕЛ ЗА ГРАНИЦЫ ПОЛЯ");
    }

    IF (FIELD[Y][X] == CELLSTATUS::SHIP) { // ЕСЛИ КЛЕТКА СОДЕРЖИТ КОРАБЛЬ
        // СОЗДАЕМ ПАРУ КЛЮЧЕЙ КООРДИНАТ
        STD::PAIR<INT, INT> COORDS(X, Y);
```

```
        // ПРОВЕРЯЕМ, ЕСТЬ ЛИ КОРАБЛЬ ПО ЭТИМ КООРДИНАТАМ
        AUTO IT = SHIPPOSITIONS.FIND(COORDS);

        IF (IT != SHIPPOSITIONS.END()) {
            SHIP* TARGETSHIP = IT->SECOND; // ПОЛУЧАЕМ УКАЗАТЕЛЬ НА АТАКУЕМЫЙ
КОРАБЛЬ
            IF(TARGETSHIP->ISDESTROYED())
                RETURN TRUE;
            // ОПРЕДЕЛЯЕМ ИНДЕКС СЕГМЕНТА В КОРАБЛЕ
            INT SEGMENTINDEX = -1;
            WHILE(FIELD[Y][X] == CELLSTATUS::SHIP ) {
                IF (TARGETSHIP->GETORIENTATION())
                    Y--;
                ELSE
                    X--;
                SEGMENTINDEX++;
                IF(Y<0 || X<0)
                    BREAK;
            }
            IT->SECOND->SHOTSEGMENT(SEGMENTINDEX);

            IF(TARGETSHIP->ISDESTROYED()){
                STD::COUT << "КОРАБЛЬ УНИЧТОЖЕН!"<< STD::ENDL;
                RETURN TRUE;
            }
        }
    }ELSE
        STD::COUT << "ПРОМАХ!";
    RETURN FALSE;
}

VOID GAMEFIELD::PRINTFIELD() CONST {
    FOR (INT Y = 0; Y < HEIGHT; ++Y) {
        FOR (INT X = 0; X < WIDTH; ++X) {
            SWITCH (FIELD[Y][X]) {
                CASE CELLSTATUS::UNKNOWN:
                    STD::COUT << "? ";
                    BREAK;
                CASE CELLSTATUS::EMPTY:
                    STD::COUT << "~ ";
                    BREAK;
                CASE CELLSTATUS::SHIP:
                    INT I=X;
                    INT J=Y;
                    STD::PAIR<INT, INT> COORDS(I, J);

                    AUTO IT = SHIPPOSITIONS.FIND(COORDS);
                    IF (IT != SHIPPOSITIONS.END()) {
                        SHIP* TARGETSHIP = IT->SECOND; // ПОЛУЧАЕМ УКАЗАТЕЛЬ
НА АТАКУЕМЫЙ КОРАБЛЬ
                         // ОПРЕДЕЛЯЕМ ИНДЕКС СЕГМЕНТА В КОРАБЛЕ
                        INT SEGMENTINDEX = -1;
                        WHILE(FIELD[J][I] == CELLSTATUS::SHIP ) {
                            IF (TARGETSHIP->GETORIENTATION())
                                J--;
                            ELSE
                                I--;
                            SEGMENTINDEX++;
                            IF(J<0 || I<0)
                                BREAK;
                        }
                        AUTO STATUS = IT->SECOND->GETSTATUS(SEGMENTINDEX);
                        IF(STATUS == SEGMENTSTATUS::DESTROYED){
```

```
                            STD::COUT << "X ";
                            BREAK;
                        }
                        IF(STATUS == SEGMENTSTATUS::DAMAGE){
                            STD::COUT << "D ";
                            BREAK;
                        }
                        ELSE{
                            STD::COUT << "S ";
                            BREAK;
                        }
                    }
                }
            }
            STD::COUT << "\N";
        }
}
```

```
// МЕТОД ПОЛУЧЕНИЯ СТАТУСА КЛЕТКИ
CELLSTATUS GAMEFIELD::GETSTATUS(INT X, INT Y) CONST {
    IF (X < 0 || X >= WIDTH || Y < 0 || Y >= HEIGHT) {
        THROW GAMEFIELDEXCEPTION("КООРДИНАТЫ ВНЕ ПОЛЯ");
    }
    RETURN FIELD[Y][X];
}
```

GameStatus.h

```
#IFNDEF GAME_STATUS_H
#DEFINE GAME_STATUS_H

#INCLUDE "GAMEFIELD.H"

CLASS GAMESTATUS{
PUBLIC:
    GAMEFIELD* FIELDPLAYER;
    SHIPMANAGER* MANAGER;

    GAMESTATUS(GAMEFIELD* FIELDPLAYER, SHIPMANAGER* MANAGER);
};

#ENDIF
```

GameStatus.cpp

```
      #INCLUDE "GAMESTATUS.H"


      GAMESTATUS::GAMESTATUS(GAMEFIELD*  FIELDPLAYER,SHIPMANAGER*  MANAGER)  :
FIELDPLAYER(FIELDPLAYER), MANAGER(MANAGER){}
```

Ability.h

```
#IFNDEF ABILITY_H
#DEFINE ABILITY_H

#INCLUDE <VECTOR>
#INCLUDE <MEMORY>
#INCLUDE <RANDOM>
#INCLUDE "GAMESTATUS.H"

CLASS ABILITY {
PUBLIC:
    VIRTUAL ~ABILITY() {}
    VIRTUAL BOOL APPLY(GAMESTATUS STATUS) = 0;
```

```
    VIRTUAL STD::STRING TOSTRING() CONST = 0;
};
#ENDIF
```

AbilityManager.h

```
#IFNDEF ABILITYMANAGER_H
#DEFINE ABILITYMANAGER_H

#INCLUDE <VECTOR>
#INCLUDE <MEMORY>
#INCLUDE <RANDOM>
#INCLUDE "ABILITY.H" // ПРЕДПОЛАГАЕТСЯ, ЧТО У ВАС ЕСТЬ БАЗОВЫЙ КЛАСС ABILITY
#INCLUDE "GAMESTATUS.H" // ПРЕДПОЛАГАЕТСЯ, ЧТО У ВАС ЕСТЬ КЛАСС GAMESTATUS

CLASS ABILITYMANAGER {
PUBLIC:
    ABILITYMANAGER();

    VOID APPLYFIRSTAVAILABLEABILITY(GAMESTATUS& STATUS);
    VOID ADDABILITY();


PRIVATE:
    VOID INITIALIZEABILITYPOOL();

    STD::VECTOR<STD::UNIQUE_PTR<ABILITY>> ABILITIES; // СПОСОБНОСТИ ИГРОКА
    STD::VECTOR<STD::UNIQUE_PTR<ABILITY>> ABILITYPOOL; // ПУЛ СПОСОБНОСТЕЙ
    STD::MT19937 RNG; // ГЕНЕРАТОР СЛУЧАЙНЫХ ЧИСЕЛ
};

#ENDIF // ABILITYMANAGER_H
```

AbilityManager.cpp

```
      #INCLUDE "ABILITYMANAGER.H"
      #INCLUDE "BARRAGEABILITY.H"
      #INCLUDE "DOUBLEDAMAGEABILITY.H"
      #INCLUDE "SCANNERABILITY.H"
      #INCLUDE "CUSTOMEXCEPTIONS.H"
      #INCLUDE <ALGORITHM>
      #INCLUDE <IOSTREAM>
      #INCLUDE <STDEXCEPT>

      ABILITYMANAGER::ABILITYMANAGER()
          : RNG(STD::RANDOM_DEVICE{}()) {
          ABILITIES.PUSH_BACK(STD::MAKE_UNIQUE<DOUBLEDAMAGE>());
          ABILITIES.PUSH_BACK(STD::MAKE_UNIQUE<SCANNER>());
          ABILITIES.PUSH_BACK(STD::MAKE_UNIQUE<BARRAGE>());

          STD::SHUFFLE(ABILITIES.BEGIN(), ABILITIES.END(), RNG);

      }

      VOID ABILITYMANAGER::INITIALIZEABILITYPOOL() {

          // ИНИЦИАЛИЗАЦИЯ ПУЛА СПОСОБНОСТЕЙ (ПО ОДНОЙ КАЖДОГО ВИДА)
          ABILITYPOOL.PUSH_BACK(STD::MAKE_UNIQUE<DOUBLEDAMAGE>());
          ABILITYPOOL.PUSH_BACK(STD::MAKE_UNIQUE<SCANNER>());
          ABILITYPOOL.PUSH_BACK(STD::MAKE_UNIQUE<BARRAGE>());
      }

      VOID ABILITYMANAGER::APPLYFIRSTAVAILABLEABILITY(GAMESTATUS& STATUS) {
          IF (!ABILITIES.EMPTY()) {
              AUTO&   ABILITY   =   ABILITIES.FRONT();   //   ПОЛУЧАЕМ   ПЕРВУЮ
СПОСОБНОСТЬ
              IF (ABILITY){
```

```
                TRY {
                    IF(ABILITY->APPLY(STATUS))
                        ADDABILITY();
                } CATCH (ABILITYMANAGEREXCEPTION& E) {
                    STD::CERR << E.WHAT() << STD::ENDL;
                }
                // УДАЛЯЕМ СПОСОБНОСТЬ ИЗ СПИСКА
                ABILITIES.ERASE(ABILITIES.BEGIN());
            } ELSE {
                THROW     ABILITYMANAGEREXCEPTION("НЕВОЗМОЖНО     ПРИМЕНИТЬ
СПОСОБНОСТЬ");
            }
        } ELSE {
            THROW    ABILITYMANAGEREXCEPTION("У    ИГРОКА    НЕТ    ДОСТУПНЫХ
СПОСОБНОСТЕЙ");
        }
    }

    VOID ABILITYMANAGER::ADDABILITY() {
        INITIALIZEABILITYPOOL();
        IF (!ABILITYPOOL.EMPTY()) {
            INT       INDEX       =       STD::UNIFORM_INT_DISTRIBUTION<>(0,
ABILITYPOOL.SIZE() - 1)(RNG);
            ABILITIES.PUSH_BACK(STD::MOVE(ABILITYPOOL[INDEX]));
            STD::COUT  <<  "ДОБАВЛЕНА  СПОСОБНОСТЬ:  "  <<  ABILITIES.BACK()-
>TOSTRING() << STD::ENDL; // ВЫВОДИМ ИНФОРМАЦИЮ О СПОСОБНОСТИ
            ABILITYPOOL.CLEAR();
        } ELSE {
            THROW ABILITYMANAGEREXCEPTION("ПУЛ СПОСОБНОСТЕЙ ПУСТ");
        }
    }
```

BarrageAbility.h

```
    #IFNDEF BARRAGE_ABILITY_H
    #DEFINE BARRAGE_ABILITY_H

    #INCLUDE "ABILITY.H"

    CLASS BARRAGE : PUBLIC ABILITY {
    PUBLIC:
        BOOL APPLY(GAMESTATUS STATUS) OVERRIDE;
        STD::STRING TOSTRING() CONST OVERRIDE;
    };


    #ENDIF
```

BarrageAbility.cpp

```
#INCLUDE "BARRAGEABILITY.H"
#INCLUDE "CUSTOMEXCEPTIONS.H"

BOOL BARRAGE::APPLY(GAMESTATUS STATUS) {
    STD::COUT << "ОБСТРЕЛ АКТИВИРОВАН!" << STD::ENDL;
    BOOL CHEK =TRUE;
    SHIP* SHIP;
    INT RANDOMSEGMENT;
    STD::MT19937 GEN(STD::RANDOM_DEVICE{}());
    WHILE(CHEK){
        STD::UNIFORM_INT_DISTRIBUTION<INT>    SHIPDIST(0,    STATUS.MANAGER-
>GETSHIPSCOUNT() - 1);
        SHIP = STATUS.MANAGER->GETSHIP(SHIPDIST(GEN));
        IF(! SHIP->ISDESTROYED())
            CHEK=FALSE;
    }
    STD::UNIFORM_INT_DISTRIBUTION<INT> SEGMENTDIST(0, SHIP->GETLENGTH() - 1);
```

```
    WHILE(!CHEK){
        RANDOMSEGMENT = SEGMENTDIST(GEN);
        TRY {
            IF(SHIP->GETSTATUS(RANDOMSEGMENT) != SEGMENTSTATUS::DESTROYED)
                CHEK=TRUE;
            } CATCH (SHIPMANAGEREXCEPTION& E) {
                STD::CERR << E.WHAT() << STD::ENDL;
            }
    }

    TRY {
        SHIP->SHOTSEGMENT(RANDOMSEGMENT);
        } CATCH (SHIPMANAGEREXCEPTION& E) {
            STD::CERR << E.WHAT() << STD::ENDL;
        }

    IF(SHIP->ISDESTROYED()){
                STD::COUT << "КОРАБЛЬ УНИЧТОЖЕН!"<< STD::ENDL;
                RETURN TRUE;
            }
    RETURN FALSE;
}

STD::STRING BARRAGE::TOSTRING() CONST {
    RETURN "BARRAGE"; // ОПИСАНИЕ СПОСОБНОСТИ
}
```

DoubleDamageAbility.h

```
      #IFNDEF DOUBLE_DAMAGE_ABILITY_H
      #DEFINE DOUBLE_DAMAGE_ABILITY_H

      #INCLUDE "ABILITY.H"

      CLASS DOUBLEDAMAGE : PUBLIC ABILITY {
      PUBLIC:
          BOOL APPLY(GAMESTATUS STATUS) OVERRIDE;
          STD::STRING TOSTRING() CONST OVERRIDE;
      };
      #ENDIF
```

DoubleDamageAbility.cpp

```
      #INCLUDE "DOUBLEDAMAGEABILITY.H"
      #INCLUDE "CUSTOMEXCEPTIONS.H"


      BOOL DOUBLEDAMAGE::APPLY(GAMESTATUS STATUS) {
          STD::COUT << "ДВОЙНОЙ УРОН АКТИВИРОВАН!" << STD::ENDL;
          INT X, Y;
          BOOL CHEK = TRUE;
          WHILE(CHEK){
              STD::COUT << "\NВВЕДИТЕ КООРДИНАТУ ДЛЯ ВЫСТРЕЛА (X Y): ";
              STD::CIN >> X >> Y;
              IF(STD::CIN.FAIL()){
                  STD::CIN.CLEAR();

STD::CIN.IGNORE(STD::NUMERIC_LIMITS<STD::STREAMSIZE>::MAX(), '\N');
                  STD::COUT << "КООРДИНАТЫ ПРОИГНОРИРОВАНЫ "<< STD::ENDL;
                  CONTINUE;
              }
              CHEK=FALSE;
          }
          TRY {
              STATUS.FIELDPLAYER->ATTACKCELL(X, Y);
              IF(STATUS.FIELDPLAYER->ATTACKCELL(X, Y))
                  RETURN TRUE;
```

```
        } CATCH (GAMEFIELDEXCEPTION& E) {
            STD::CERR << E.WHAT() << STD::ENDL;
        }

        RETURN FALSE;
    }


    STD::STRING DOUBLEDAMAGE:: TOSTRING() CONST{
        RETURN "ДВОЙНОЙ УРОН"; // ОПИСАНИЕ СПОСОБНОСТИ
    }
```

ScannerAbility.h

```
    #IFNDEF SCANNER_ABILITY_H
    #DEFINE SCANNER_ABILITY_H

    #INCLUDE "ABILITY.H"

    CLASS SCANNER : PUBLIC ABILITY {
    PUBLIC:
        BOOL APPLY(GAMESTATUS STATUS) OVERRIDE;
        STD::STRING TOSTRING() CONST OVERRIDE;
    };

    #ENDIF
```

ScannerAbility.cpp

```
    #INCLUDE "SCANNERABILITY.H"
    #INCLUDE "CUSTOMEXCEPTIONS.H"

    BOOL SCANNER::APPLY(GAMESTATUS STATUS) {
        INT X, Y;
        BOOL CHEK= TRUE;
        WHILE(CHEK){
            STD::COUT << "\NВВЕДИТЕ КООРДИНАТУ ДЛЯ НАЧАЛА СКАНИРОВАНИЯ (X
Y): ";
            STD::CIN >> X >> Y;
            IF(STD::CIN.FAIL()){
                STD::CIN.CLEAR();

STD::CIN.IGNORE(STD::NUMERIC_LIMITS<STD::STREAMSIZE>::MAX(), '\N');
                STD::COUT << "КООРДИНАТЫ ПРОИГНОРИРОВАНЫ "<< STD::ENDL;
                CONTINUE;
                }
            CHEK=FALSE;
        }
        STD::COUT << "СКАНЕР АКТИВИРОВАН!" << STD::ENDL;
        FOR (INT I = 0; I <= 1; ++I) {
            FOR (INT J = 0; J <= 1; ++J) {
                CELLSTATUS STAT;
                TRY {
                    STAT = STATUS.FIELDPLAYER->GETSTATUS(Y+I,X+J);
                } CATCH (GAMEFIELDEXCEPTION& E) {
                    STD::CERR << E.WHAT() << STD::ENDL;
                    RETURN FALSE;
                }
                IF (STAT == CELLSTATUS::SHIP){
                    STD::COUT << "КОРАБЛЬ НАЙДЕН" << STD::ENDL;
                    RETURN FALSE;
                }
            }
        }
        STD::COUT << "КОРАБЛЬ НЕ НАЙДЕН" << STD::ENDL;
        RETURN  FALSE;
    }
```

```
      STD::STRING SCANNER:: TOSTRING() CONST{
          RETURN "СКАНЕР"; // ОПИСАНИЕ СПОСОБНОСТИ
      }
```

CustomException.h

```
      #IFNDEF CUSTOM_EXCEPTIONS_H
      #DEFINE CUSTOM_EXCEPTIONS_H

      #INCLUDE <STDEXCEPT>

      CLASS GAMEFIELDEXCEPTION : PUBLIC STD::RUNTIME_ERROR {
      PUBLIC:
          GAMEFIELDEXCEPTION(CONST CHAR* MESSAGE) : RUNTIME_ERROR(MESSAGE) {}
      };

      CLASS SHIPMANAGEREXCEPTION : PUBLIC STD::RUNTIME_ERROR {
      PUBLIC:
          SHIPMANAGEREXCEPTION(CONST CHAR* MESSAGE) : RUNTIME_ERROR(MESSAGE)
{}
      };

      CLASS ABILITYMANAGEREXCEPTION : PUBLIC STD::RUNTIME_ERROR {
      PUBLIC:
          ABILITYMANAGEREXCEPTION(CONST        CHAR*        MESSAGE)        :
RUNTIME_ERROR(MESSAGE) {}
      };

      #ENDIF // CUSTOM_EXCEPTIONS_H
```

CustomException.cpp

```
#INCLUDE "CUSTOMEXCEPTIONS.H"

GAMEFIELDEXCEPTION::GAMEFIELDEXCEPTION(CONST      CHAR*      MESSAGE)      :
RUNTIME_ERROR(MESSAGE) {}

SHIPMANAGEREXCEPTION::SHIPMANAGEREXCEPTION(CONST     CHAR*     MESSAGE)     :
RUNTIME_ERROR(MESSAGE) {}

ABILITYMANAGEREXCEPTION::ABILITYMANAGEREXCEPTION(CONST   CHAR*   MESSAGE)   :
RUNTIME_ERROR(MESSAGE) {}
```

Main.cpp

```
#INCLUDE "ABILITYMANAGER.H"
#INCLUDE "CUSTOMEXCEPTIONS.H"

// ОСНОВНАЯ ФУНКЦИЯ ПРОГРАММЫ
INT MAIN() {
    SYSTEM("CHCP 65001");

    // СОЗДАНИЕ ОБЪЕКТА GAMEFIELD
    INT WIDTH, HEIGHT;
    STD::COUT << "ВВЕДИТЕ РАЗМЕРЫ ПОЛЯ (ШИРИНА ДЛИНА): ";
    STD::CIN >> WIDTH >> HEIGHT;
    GAMEFIELD GAMEFIELD(WIDTH, HEIGHT);


    // СОЗДАНИЕ ОБЪЕКТА SHIPMANAGER
    //STD::VECTOR<STD::PAIR<INT, INT>> SHIPSPECS = {{1,1}};
    STD::VECTOR<STD::PAIR<INT, INT>> SHIPSPECS = {{4, 1}, {3, 2}, {2, 3}, {1,
4}};
    SHIPMANAGER SHIPMANAGER(SHIPSPECS);

    // РАССТАНОВКА КОРАБЛЕЙ НА ПОЛЕ
    INT X, Y;
    CHAR ORIENTATION;
```

```
    BOOL VERTICAL = TRUE;

    STD::COUT << "РАССТАНОВКА КОРАБЛЕЙ:" << STD::ENDL;
    FOR (INT I=0; I<SHIPMANAGER.GETSHIPSCOUNT(); I++) {
        BOOL PLACED = FALSE;
        WHILE (!PLACED) {
            STD::COUT << "ВВЕДИТЕ КООРДИНАТЫ ДЛЯ " << SHIPMANAGER.GETSHIP(I)-
>GETLENGTH() << "-СЕГМЕНТНОГО КОРАБЛЯ (X Y): ";
            STD::CIN >> X >> Y;
            IF(STD::CIN.FAIL()){
            STD::CIN.CLEAR();
            STD::CIN.IGNORE(STD::NUMERIC_LIMITS<STD::STREAMSIZE>::MAX(),
'\N');
            STD::COUT << "КООРДИНАТЫ ПРОИГНОРИРОВАНЫ "<< STD::ENDL;
            CONTINUE;
            }
            IF(SHIPMANAGER.GETSHIP(I)->GETLENGTH()!= 1){
                STD::COUT << "ВВЕДИТЕ ЕГО ОРИЕНТАЦИЮ: 1 - ДЛЯ ВЕРТИКАЛЬНОГО,
0 - ДЛЯ ГОРИЗОНТАЛЬНОГО ";
                STD::CIN >> ORIENTATION;
                IF(ORIENTATION=='0' || ORIENTATION=='1')
                    VERTICAL=(BOOL)ORIENTATION;
                ELSE{

STD::CIN.IGNORE(STD::NUMERIC_LIMITS<STD::STREAMSIZE>::MAX(), '\N');
                    STD::COUT << "КООРДИНАТЫ ПРОИГНОРИРОВАНЫ "<< STD::ENDL;
                    CONTINUE;
                }
            }
            SHIPMANAGER.GETSHIP(I)->SETVERTICAL( VERTICAL);
             TRY {
                GAMEFIELD.PLACESHIP(SHIPMANAGER.GETSHIP(I), X, Y);
                PLACED = TRUE;
            } CATCH (GAMEFIELDEXCEPTION& E) {
                STD::COUT << E.WHAT() << STD::ENDL;
            }

        }
    }

    STD::COUT << "\NТЕКУЩЕЕ СОСТОЯНИЕ ПОЛЯ:" << STD::ENDL;
    GAMEFIELD.PRINTFIELD();
    // ИГРА
    CHAR INPUT;
    INT CHEKING;
    ABILITYMANAGER MANAGER;
    GAMESTATUS STATUS(&GAMEFIELD, &SHIPMANAGER);
    WHILE (TRUE) {
        STD::COUT << "\NВЫБЕРИТЕ ДЕЙСТВИЕ: 2 - ДЛЯ ВЫВОДА ТЕКУЩЕГО СОСТОЯНИЯ
ПОЛЯ, 1 - ДЛЯ ВЫСТРЕЛА, 0 - ДЛЯ ПРИМЕНЕНИЯ СПОСОБНОСТИ ";
        STD::CIN >> CHEKING;
        SWITCH (CHEKING){
        CASE 0:
            TRY {
                MANAGER.APPLYFIRSTAVAILABLEABILITY(STATUS);
                BREAK;
            } CATCH (ABILITYMANAGEREXCEPTION& E) {
                STD::COUT << E.WHAT() << STD::ENDL;
            }
        CASE 1:
            STD::COUT << "\NВВЕДИТЕ КООРДИНАТУ ДЛЯ ВЫСТРЕЛА (X Y): ";
            STD::CIN >> X >> Y;
            IF(STD::CIN.FAIL()){
```

```
            STD::CIN.CLEAR();
            STD::CIN.IGNORE(STD::NUMERIC_LIMITS<STD::STREAMSIZE>::MAX(),
'\N');
            STD::COUT << "КООРДИНАТЫ ПРОИГНОРИРОВАНЫ "<< STD::ENDL;
            CONTINUE;
            }
            TRY{
                IF(GAMEFIELD.ATTACKCELL(X, Y)==TRUE){
                        MANAGER.ADDABILITY();
                }
            } CATCH (GAMEFIELDEXCEPTION& E) {
                STD::COUT << E.WHAT() << STD::ENDL;
            }
            BREAK;
        CASE 2:
            STD::COUT << "\NТЕКУЩЕЕ СОСТОЯНИЕ ПОЛЯ:" << STD::ENDL;
            GAMEFIELD.PRINTFIELD();
            BREAK;
        DEFAULT:
            STD::COUT << "\NНЕИЗВЕСТНАЯ КОМАНДА!" << STD::ENDL;
            BREAK;
        }
        IF (SHIPMANAGER.ISDESTROYED()) {
            STD::COUT << "\NПОЗДРАВЛЯЮ! ВЫ ПОБЕДИЛИ!" << STD::ENDL;
            BREAK;
        }
    }

    RETURN 0;
}
```

Makefile.mak

```
      ALL: PLAY

      PLAY:    MAIN.O    GAMEFIELD.O    SHIPMANAGER.O    SHIP.O    SHIPSEGMENT.O
GAMESTATUS.O    ABILITYMANAGER.O    BARRAGEABILITY.O    DOUBLEDAMAGEABILITY.O
SCANNERABILITY.O
           G++     MAIN.O  GAMEFIELD.O  SHIPMANAGER.O  SHIP.O  SHIPSEGMENT.O
GAMESTATUS.O    ABILITYMANAGER.O    BARRAGEABILITY.O    DOUBLEDAMAGEABILITY.O
SCANNERABILITY.O -O PLAY
      MAIN.O: MAIN.CPP
            G++ -C MAIN.CPP
      SHIPMANAGER.O: SHIPMANAGER.CPP SHIPMANAGER.H
            G++ -C SHIPMANAGER.CPP
      SHIP.O: SHIP.CPP SHIP.H CUSTOMEXCEPTIONS.H
            G++ -C SHIP.CPP
      SHIPSEGMENT.O: SHIPSEGMENT.CPP SHIPSEGMENT.H
            G++ -C SHIPSEGMENT.CPP
      GAMESTATUS.O: GAMESTATUS.CPP GAMESTATUS.H
            G++ -C GAMESTATUS.CPP
      GAMEFIELD.O: GAMEFIELD.CPP GAMEFIELD.H CUSTOMEXCEPTIONS.H
            G++ -C GAMEFIELD.CPP
      ABILITYMANAGER.O:    ABILITYMANAGER.CPP    ABILITYMANAGER.H    ABILITY.H
CUSTOMEXCEPTIONS.H
            G++ -C ABILITYMANAGER.CPP
      BARRAGEABILITY.O:          BARRAGEABILITY.CPP          BARRAGEABILITY.H
CUSTOMEXCEPTIONS.H
            G++ -C BARRAGEABILITY.CPP
      DOUBLEDAMAGEABILITY.O:   DOUBLEDAMAGEABILITY.CPP   DOUBLEDAMAGEABILITY.H
CUSTOMEXCEPTIONS.H
            G++ -C DOUBLEDAMAGEABILITY.CPP
      SCANNERABILITY.O: SCANNERABILITY.CPP SCANNERABILITY.H
            G++ -C SCANNERABILITY.CPP
      CLEAN:
```

```
REMOVE-ITEM *.O -FORCE
```