독어교육과 정은주
2014-19498

computer programming Project1_ Report

*Unexpected Situation handled

1. Developement/Inplementation Environment

Mac OS, Visual Code

2. code description

1) Vehicle, Car, Airplane, Submarine Class (Inheritance)

먼저 본 project는 header file과 cpp file로 나누어 작성하였으나, 편의상 파일들을 하나의 cpp 파일로 합쳤다.

<Class Vehicle>

```
private:
    int speed;
    int temperature;
    int humidity;
    //** all the resources

    int plusEnergy;
    int oxygenLoss;
    int speedLoss;
    int energyLoss;
    //** loss and plus

    string status;
    bool firstMode;
    bool obstacleMode;
    //** status checking:
```

```
class Vehicle {
    public:
        Vehicle(int speed, int temperature, int humidity)
        {       //** initialization **//
                energyLoss = 0;
                speedLoss = 0;
                oxygenLoss = 0;
                plusEnergy = 0;
                firstMode = true;
                status = "";

            if(speed == 0) //** X & Y (speed 0) else (speed
                this -> obstacleMode = true;
            else {
                this -> obstacleMode = false;
                set_speed(speed);
                set_temperature(temperature);
                //** submarine(speed 20) has no effect of h
                if(speed != 20)
                    set_humidity(humidity);
            }
```

Class Vehicle (member variables & constructor)

1. 공통 variable (speed, temperature, humidity) + loss & plus varialbles

먼저 Vehicle이라는 class를 Car, Airplane, Submarine이 상속하도록 하였다. 더불어 각 클래스에서 member initializer를 통해서 super class를 초기화하도록 하였다. Vehicle들이 공통으로 가지고 있는 speed, temperature, humidity 변수들을 getter와 setter를 통해서 설정하도록 한다. 이때, temperature, humidity의 경우에는 범위에 따라서 energy loss를 고려해 설정해주었다. 더불어 oxygenLoss와 speedLoss 역시 따로 getter와 setter를 설정하였으며, Car class의 solar panel을 통해서 충전을 받는 plusEnergy 역시 getter와 setter를 이용해 설정해 두어 main function에서 값을 이용할 수 있도록 하였다.

```
void set_speed(int speed) { this-> speed = speed; }
void set_temperature(int temperature)
{
    this -> temperature = temperature;
    if(temperature < 40 && temperature > 0)
        set_energyLoss(5);

    else if(temperature >= 40)
        set_energyLoss(10);

    else if(temperature == 0)
        set_energyLoss(8);
}
void set_humidity(int humidity)
{
    this -> humidity = humidity;
    if(humidity < 50)
        set_energyLoss(5);
    else
        set_energyLoss(8);
}
```

Journey의 첫 번째 mode는 항상 road이며, 이때 첫 번째 Car mode에서는 solar charge가 일어나지 않으므로 이를 위해서 따로 first mode check를 설정하도록 하였다. 더불어, current_status() method에서는 status string을 통해서 현재 Vehicle의 상태가 Car, Submarine, Airplane, 혹은 X,Y인지를 알 수 있도록 하였고, X와 Y의 경우에는 obstacle mode 이므로 이 역시 체크할 수 있도록 하였다. 더불어 속력을 0으로 설정하여 다른 variable들의 setter에 영향을 주지 않도록 했다.

submarine mode의 경우에는 Ocean 에서는 humidity가 100이지만, 이로 인한 energy Loss 가 발생하지 않기 때문에, submarine의 speed인 20이 아닌 경우에만, humidity setter가 작동하도록 하였다.

2. status print(status, distance, speed, energy, temperature etc.)

```
void print(int distance, int energy, int oxygen) {
    cout << "Current Status: " << current_status() << endl;
    cout << "Distance: " << distance << " km" << endl;
    cout << "Speed: " << get_speed() << " km/hr" << endl;
    cout << "Energy: " << energy << endl;
    if(current_status() != "Car") cout << "Oxygen Level: " << oxygen << endl;
    cout << "Temperature: " << get_temperature()<< " C" << endl;
    if(current_status() != "Submarine") cout << "Humidity: " << get_humidity() << endl;
}
```

class Vehicle print() method

```
void print(int distance, int energy, int oxygen)
{
    Vehicle::print(distance, energy, oxygen);
    cout << "Depth: " << get_depth() << " m" << endl;
    cout << "Water Flow: " << get_waterflow() << endl;
}
```

class Submarine print() method

```
void print(int distance, int energy, int oxygen)
{
    Vehicle::print(distance, energy, oxygen);
    cout << "Altitude: " << get_altitude() << " m" << endl;
    cout << "Air Density: " << get_airDensity() << endl;
}
```

class Airplane print() method

print method를 통해서 본 current status와 distance, speed, energy 그리고 temperature를 출력한다. car mode가 아닌 경우에 oxygen을, submarine이 아닌 경우에 humidity를 출력하도록 한다. 이때, 움직임에 따라서 값이 변하는 energy, distance, oxygen은 매개 변수로 값을 받도록 한다. 나머지 변수들의 경우에는 private member variable을 그대로 사용하는 것이 아니라, setter를 통해서 설정해둔 값을 getter를 통해서 얻을 수 있도록 하여, encapsulation을 유지하려고 노력하였다.

<Class Car>

```
class Car : public Vehicle {
    public :
        Car(int temperature, int humidity, bool firstMode)
        : Vehicle(80, temperature, humidity)
        {   // car speed = 80
            this -> solarEnergy = 0;
            this -> firstMode = firstMode;
            Vehicle::set_firstMode(firstMode);
            Vehicle::set_status("Car");
            check_humidity(humidity);
        }

        void check_humidity(int humidity)
        {
            if(humidity < 50 && firstMode == false)
                solar_recharge();
        }
        void solar_recharge()
        {
            this -> solarEnergy = 200;
            Vehicle::set_plusEnergy(solarEnergy);
        }

    private:
        int solarEnergy;
        bool firstMode;
};
```

Car mode의 경우에는 first Mode check를 통해서 첫 번째 mode인지를 체크할 수 있도록 하였고, solar_recharge를 통해서 첫 번째 mode가 아니면 solar panel recharge가 일어날 수 있도록 했다. 더불어 이때, humidity가 50보다 작다는 조건이 성립할 때에만 solar_recharge function이 작동하여 에너지 충전이 가능하도록 했다. 따라서 super class

인 Vehicle의 set_plusEnergy function을 통해서 추가 에너지를 후에 main function에서 더할 수 있도록 했다.

<Class Airplane>

```
class Airplane : public Vehicle
{
    public:
        Airplane(int temperature, int humidity, int altitude, int airDensity)
        : Vehicle(900, temperature, humidity)
        {   // Airplane speed = 900
            set_altitude(altitude);
            set_airDensity(airDensity);
            Vehicle::set_firstMode(false);
            Vehicle::set_status("Airplane");
        }
        void set_altitude(int altitude)
        {
            this -> altitude = altitude;
            int n = altitude/1000;
            Vehicle::set_oxygenLoss(n*10);
        }
        void set_airDensity(int airDensity)
        {
            this -> airDensity = airDensity;
            if(airDensity >= 30 && airDensity < 50)
                Vehicle::set_speedLoss(200);

            else if(airDensity >=50 && airDensity < 70)
                Vehicle::set_speedLoss(300);

            else if(airDensity >= 70)
                Vehicle::set_speedLoss(500);
        }
        int get_altitude() { return altitude; }
        int get_airDensity() { return airDensity; }
```

Airplane의 경우에는 altitude, airDensity를 private Variable로 가지며, getter와 setter를 통해서 값을 얻을 수 있도록 하였다. 더불어서 air density와 altitude의 값에 의해서 altitude는 oxygen loss가 발생하며, 이는 1000 단위이다. air density의 경우에는 각각 30, 50 그리고 70 이상일 때, 200, 300, 500의 speed loss가 발생함을 알고, super class인 Vehicle에서 speed loss setter를 통해서 스피드 loss를 기록한다. 더불어서 print 함수를 통해서는 super class의 distance, oxygen, energy 등을 매개변수로 보내고 똑같은 부분을 출력하도록 한 이후에 추가로 출력해야하는 altitude와 air density 값이 출력될 수 있도록 한다.

< Class Submarine >

```
class Submarine : public Vehicle
{
    public :
       Submarine(int temperature, int depth, int waterflow)
        : Vehicle(20, temperature, 100)
        {   // Submarine speed = 20, humidity = 100;
            set_depth(depth);
            set_waterflow(waterflow);
            Vehicle::set_firstMode(false);
            Vehicle::set_status("Submarine");
            light_on();
        }
        void set_depth(int depth)
        {
            this -> depth = depth;
            int n = depth/50;
            Vehicle::set_oxygenLoss(n*5);
        }
        void set_waterflow(int waterflow)
        {
            this -> waterflow = waterflow;
            if(this -> waterflow >=30 && this -> waterflow < 50)
                Vehicle::set_speedLoss(3);

            else if(this -> waterflow >= 50 && this -> waterflow < 70)
                Vehicle::set_speedLoss(5);

            else if(this -> waterflow >=70)
                Vehicle::set_speedLoss(10);
        }
        void light_on() { Vehicle::set_energyLoss(30);}
        int get_depth() { return depth;}
        int get_waterflow() { return waterflow;}
```

Submarine mode의 경우에는 depth와 waterflow를 설정하여, 이에 맞게 Vehicle의 speedLoss 혹은 energyLoss, oxygenLoss가 발생하도록 하였다. 더불어서 light_on()이라는 method를 만들어서 초기화 시 energyloss가 발생할 수 있도록 설정해 주었다. 더불어 print() method에서는 부모 class인 Vehicle의 상태를 그대로 출력하고, depth 와 water flow를 추가적으로 출력할 수 있도록 한다.

2) Code Flow - Variables & Funtions

```
void initializer(); // initialization for global variables
void test_case(); // test cases set.
int choose_testCase();
void tokenize(string);
int* unit_tokenize(int reasources[], char* mode, int index, int modeCount);

int next_move(); // choosing next mode 1 or 2
char* get_status(); // current mode
bool percent(int); // for unexpected mode
int unit_set(char* mode); // unit(car, submarine, airplane)
string move_status(int move); // successfully moved.

bool mode_change(int energy, int oxygen, int distance);
void graphic_record(int lastPos, int currentPos);
string final_status(int energy, int oxygen, int distance);
string finish(int energy, int oxygen); // FINISHED!
string blackbox(string mode, string energy, string oxygen, string speed);
```

Funtion Declaration

```
const static int ROADUNIT = 50;
const static int OCEANUNIT = 10;
const static int SKYUNIT = 1000;

static bool extraMode = false;
static bool damaged = false;

static vector<Vehicle*> journey; // all the journeys
static string TC[10]; // test cases
static vector<int> distance_sum; // distance accumulation
static string graphic = "|";
```

Global Variables

먼저 거리, 속력, 단위별 움직임, 산소 등의 값을 처음 road mode의 값에 맞게 설정해두고, graphic을 위한 마지막 vehicle의 위치와 현재 위치 값을 int로 설정한다. 더불어서 car mode를 위해서 만약 태양열 충전을 진행했는지를 체크할 수 있도록 하기 위해서 bool 값 역시 설정해둔다. 더불어, blackbox를 위한 mode, energy, oxygen, speed record string 값 역시 설정해둔다. 이 후 다시 while 문을 통해서 game을 진행한다. 본 코드는 먼저, 흐름 상 extra mode인지를 체크하고, 만약 맞다면, 이에 맞는 확률에 따른 energy loss 혹은 vehicle stop이 발생할 수 있도록 하였다. 더불어 만약 vehicle이 완전히 멈추게 되면 while문을 빠져나오게 된다. 그 후 car mode의 경우에는 solar energy charge check를 진행한다. 만약 이전 mode에서 unexpected event에 의해서 완전히 망가지게 됐는지를 체크하는 damaged bool variable의 true, false를 체크하고, 더불어서, 첫 번째 모드인지 혹은 이미 같은 mode이기 때문에 이미 충전이 이루어졌는지를 체크하는 bool 값의 true false를 체크한다.

먼저 main function의 흐름은 cases들을 array에 담고, 먼저 extra mode 와 normal mode를 선택할 수 있도록 하였다. 그 후 while문을 통해서 진행된다. 먼저, choose() method에서 case를 선택할 수 있도록 하고, 선택한 직후 tokenize를 진행한다. 이 때, initializer()를 통해서 전역으로 설정해둔 container와 variable들의 값들을 초기화한다.

1) initializer()

```
void initializer()
{
    while(!journey.empty())
        journey.pop_back();

    while(!distance_sum.empty())
        distance_sum.pop_back();

    graphic = "|";
    damaged = false;
}
```

2) choose_testCase()

```
int choose_testCase()
{
    cout << "Choose the number of the test case (1~10) : ";
    int n;
    cin >> n;

    if(n >= 1 && n <= 10)
    {
        cout << "Test case #" << n << ".\n" << endl;
        tokenize(TC[n-1]);
        return 1;
    }
    else
        return 0;
}
```

그 이후 next_move() function을 통해서 계속 이동하게 되는데, 만약에 1번을 선택한다면, unit 단위로 이동하게 되고, 각 mode에 해당하는 energy loss oxygen loss를 modes vector container에 담긴 class의 function을 통해서 체크한다. 더불어 2번째 mode를 선택하게 된다면, for 문을 통해서 unit 단위로 energy, oxygen invalid check를 하면서 이동한다. 만약 둘 중에 하나라도 고갈이 일어나게 된다면, 그 때 움직인 unit 의 횟수를 기록해두고, 이에 따라서 move 값을 설정한다. 이렇게 next move 1과 2모두 더 이상 움직일 수 없는 값이 된다면 바로 vehicle을 멈추고, final status와 blackbox를 위한 record string에 값을 저장하게 된다. 더불어서 두 자원의 고갈이 일어나지 않았지만, 만약 해당 mode를 모두 움직인 경우에는 road distance와 mode vector container에서 해당 mode를 삭제한다. 그리고 마지막에 mode container의 사이즈가 0 이 되는 경우에는 해당 게임을 종료하게 되고, 사용자로 하여금 다시 case를 고를 수 있도록 한다.

3) Tokenize – unit_tokenize()

tokenize의 경우에는 하나의 function안에 코드가 복잡해지는 것을 방지하고자, delimiter(“,”)를 통해서 mode 별로 하나씩 쪼개고, argument로 본 mode를 unit_tokenize() function으로 보내어, 각 road, sky, ocean mode에 알맞게 delimiter에 의해서 쪼개어질 수 있도록 하였고, unit 단위역시 미리 static으로 설정해둔 road, sky, ocean에 맞게 설정했다. 더불어서 각 mode에서 필요한 자원들을 resources array에 저장할 수 있도록 하였으며, 이를 return 값으로 전달하였다. 또한 전역으로 설정해둔 distance_sum vector에 거리 값을 저장했다. 본 vector는 거리 값을 accumulation을 통해

서 담도록 하였는데, 이는 본 코드 진행상 처음에 담기는 값만을 이용하여 while문을 돌리기 때문에 계속해서 mode가 끝나면 삭제해서 contaioner의 0번째 요소를 계속 사용하도록 하였기 때문이다. 더불어서 extra credit을 위한 graphic을 위한 값 역시 미리 설정해두도록 하였다. 이렇게 tokenize 과정이 끝나면, 다시 main function으로 돌아와서 mode와 road distance vector에 담긴 값을 통해 while문을 진행한다.

<Tokenize> - each mode by mode Road(R), Sky(S), Ocean(O), X, Y

```
for(int i = 0; i < modeCount + obstacle; i++)
{
    char* nextMode = a.at(i);
    if(strstr(nextMode, "R") != NULL)
    {
        int a[3]; //road, temperature, humidity
        int *r = unit_tokenize(a, nextMode, index++, modeCount);
        Vehicle* car = new Car(r[1], r[2], (i==0)? true : false);
        journey.push_back(car);
    }
    else if(strstr(nextMode, "S") != NULL)
    {
        int a[5]; // sky, temperature, humidity, altitude, density
        int *s = unit_tokenize(a, nextMode, index++, modeCount);
        Vehicle* airplane = new Airplane(s[1], s[2], s[3], s[4]);
        journey.push_back(airplane);
    }
    else if(strstr(nextMode, "O") != NULL)
    {
        int a[4]; // ocean, temperature, depth, waterflow
        int *o = unit_tokenize(a, nextMode, index++, modeCount);
        Vehicle* submarine = new Submarine(o[1], o[2], o[3]);
        journey.push_back(submarine);
    }
    else
    {
        if(strstr(nextMode, "X") != NULL && extraMode == true)
        {
            Vehicle* X = new Vehicle(0, -100, -100);
            X -> set_energyLoss(100);
            X -> set_status("X");
            journey.push_back(X);
        }
        else if(strstr(nextMode, "Y") != NULL && extraMode == true)
        {
            Vehicle* Y = new Vehicle(0, -100, -1);
            Y -> set_oxygenLoss(30);
            Y -> set_status("Y");
            journey.push_back(Y);
        }
    }
```

tokenize를 진행한 후에 각각 mode에 알맞게 Vehicle을 만들고, 그 값을 각각의 class에 맞게 resources를 constructor를 통해서 값들을 초기화할 수 있도록 한다. 더불어 각각을 journey vector에 push back을 통해서 담도록 한다.

extramode의 경우에는 각각 energy와 oxygen loss를 설정하고, status 역시 설정해준다. 그 후 vector 값에 담도록 한다.

<unit_tokenize> - each mode by mode(Road, Sky, Ocean)

```
int* unit_tokenize(int resources[], char* mode, int index, int modeCount)
{
    int k = 0;
    int unit;
    string tok;
    string shape;
    //** setting for delimiter, unit distance, graphic string **//
    if(strstr(mode, "R") != NULL)
    {   // Road
        tok = "[]RTH";
        unit = ROADUNIT;
        shape = "=";
    }
    else if(strstr(mode, "S") != NULL)
    {   // Sky
        tok = "[]STHAD";
        unit = SKYUNIT;
        shape = "^";
    }
    else
    {   // Ocean
        tok = "[]OTDW";
        unit = OCEANUNIT;
        shape = "~";
    }
    //** resourses tokenize  **//
    char* delimiter = new char[tok.length() + 1];
    strcpy(delimiter, tok.c_str());
    char* pch = strtok(mode, delimiter);
```

** Resources, distance, mode, graphic shape etc. setting **

```
    //** resourses saving **//
    while(pch != NULL)
    {
        resources[k++] = atoi(pch);
        pch = strtok(NULL, delimiter);
    }
    //** distance accumulation **//
    for(int j = index; j < modeCount; j++)
        distance_sum[j] += resources[0];

    //** graphic string **//
    for(int j = 0; j < resources[0] / unit; j++)
        graphic += shape;

    return resources;
}
```

4) Unexpexted Events

1. 난수 구하기 (percent)

```
bool percent(int n)
{
    int random[100];
    srand((unsigned int)time(NULL));
    int i, j;

    //** 1~100 random number array without duplicate **//
    for(i = 0; i < 100 ; i++) {
        random[i] = rand() % 100+1;
        for(j = 0; j < i; j++) {
            if(random[i] == random[j]) {
                i--;
                break;
            }
        }
    }
    srand((unsigned int)time(NULL));
    int m = rand() % 100;
    int percent = random[m];

    if(percent >=0 && percent <= n)
        return true;
    else
        return false;
}
```

먼저, random 함수를 이용하여서, 1부터 100까지의 수를 임의로 중복되는 값이 없도록 하여서 array에 저장한다. 또한 임의로 srand를 이용하여서 현재시간을 null로 설정하여서, percent를 구할 때마다 새로운 값이 나올 수 있도록 한다. 그 후 매개 변수로 받은 값이 percent 값과의 범위를 따졌을 때, 해당하면, true를 아니면 false를 return 할 수 있도록 한다.

2. X mode

```
if(strstr(status, "X") != NULL)
{   if(percent(20) == false)
    {   energy -= mode -> get_energyLoss();
        energy = (energy < 0) ? 0 : energy;
        if(energy == 0)
        {
            cout << final_status(energy, oxygen, distance);
            cout << graphic << "\n" << finish(energy, oxygen);
            cout << blackbox(modeRecord, energyRecord, oxygenRecord, speedRecord);
            break;
        }
    }
    else { cout << newBlackBox; break;}
}
```

X의 경우에는 percent가 80%의 확률로 살아남게 되므로, 20%가 false 일 때와 같고, 이때 에너지가 100 감소하게 된다. 더불어, 만약 에너지가 0보다 작게 된다면, 바로 final status와 graphic 더불어 죽은 이유를 출력하고, 블랙박스 역시 출력할 수 있도록 한다. 나머지 20퍼센트의 경우에는 미리 설정해둔 new Blackbox 값에 의해 블랙박스가 표시되고, 죽게 된다.

3. Y mode

```
else if(strstr(status, "Y") != NULL)
{   if(percent(35) == false)
    {   if(percent(50) == true)
        {   if(graphic.substr(lastPos-1, lastPos).compare("="))
                damaged = true; // solar penal damaged until the end.
            else
            {
                oxygen -= mode -> get_oxygenLoss();
                oxygen = (oxygen < 0) ? 0 : oxygen;
                if(oxygen == 0)
                {
                    cout << final_status(energy, oxygen, distance);
                    cout << graphic << "\n" << finish(energy, oxygen);
                    cout << blackbox(modeRecord, energyRecord, oxygenRecord, speedRecord);
                    break;
                }
            }
        }
    } else { cout << newBlackBox; break; }
}
```

Y 모드의 경우에는 65%의 확률로 살아남게 되고, 35%의 확률이 맞다면, 바로 블랙박스를 출력하도록 한다. 살아남았다면 다시 50%의 확률로 살아남는데, 미리 설정해둔 last position int 값으로 만약 이전 mode가 Car("=")에 해당한다면 solar panel이 damaged가 되고, 후에 Car mode가 나온다면 이때, 손상된 solar panel은 damaged가 global 변수이며, 프로그램이 종료 될 때까지 값은 변하지 않는다. 이게 아니라면, 다른 ocean, sky mode에서는 oxygen이 감소하게 되며 만약 oxygen이 부족해서 다음 unit으로 갈 수 없다면, 그 자리에서 현재 상태와 블랙박스를 출력하고, 다음 case를 선택할 수 있도록 한다. 그 이외의 50%는 아무 일도 일어나지 않는다.

```
//** for the car mode solar energy recharge **//
char* status = get_status();
if(strstr(status, "Car") != NULL)
{
    if(solarChange == false && damaged == false
        && journey.at(0) -> is_firstMode() == false)
    {
        energy += journey.at(0) -> get_plusEnergy();
        energy = (energy > 1000) ? 1000 : energy;
        solarChange = true; //until mode change
    }
    oxygen = 100;
}
```

*** soalr panel ***

extra mode 의한 손상 없고, 첫 번째 mode가 아니고, 현재 mode에서 이미 충전이 진행된 경우가 아닐 때만 충전 가능!

5) Next Move - first vs second

```
switch(next_move())
{
    case(1):
    {   //** first Move: unit movement **//
        move = unit_set(status);
        energy -= journey.at(0) -> get_energyLoss();
        oxygen -= journey.at(0) -> get_oxygenLoss();
        speed = journey.at(0) -> get_speed();
        currentPos++;
        break;
    }
    case(2):
    {   //** second Move: at once movement  **//
        int unit = unit_set(status);
        int totalCount = (distance_sum.at(0) - distance)/unit;
        int realCount = 0;

        //** the Vehicle stops when there is not enough energy
        //** or oxygen, counting the times how far it can move.
        for(int i = 0; i < totalCount; i++)
        {
            energy -= journey.at(0) -> get_energyLoss();
            oxygen -= journey.at(0) -> get_oxygenLoss();
            realCount++;
            if(energy <= 0 || oxygen <= 0) break;
        }
        speed = journey.at(0) -> get_speed();
        move = unit * realCount;
        currentPos += realCount;
        break;
    }
}
```

1. next move - FISRT

첫 번째 모드에서는 unit 단위로 움직이게 된다. 이는 Road의 경우는 50, Sky의 경우는 1000, 그리고 Ocean의 경우는 10km 씩 움직이게 되며, 각각 energy Loss를 더해줄 수 있도록 한다. 더불어서 그래픽을 위한 current position variable을 한 칸 이동해준다.

2. next move - SECOND

두 번째 모드의 경우에는 전체 이동거리는 남아있는 거리 모두이므로, unit 단위로 나누어서 for 문을 통해 이동하도록 한다. 만약 energy나 oxygen이 부족한 경우는 break를 통해 빠져나오고, 이동한 만큼만 current position을 옮겨주고, switch문을 빠져나올 수 있도록 한다.

```
//** invalid range for distance, energy, oxygen check **//
distance += move; // total movement record
Vehicle* lastMode = journey.at(0);
string finish_reason;

if(mode_change(energy, oxygen, distance) == true)
{
    finish_reason = finish(energy, oxygen);
    solarChange = false; //initialization for next mode
    energy = (energy < 0) ? 0 : energy;
    oxygen = (oxygen < 0) ? 0 : oxygen;

    modeRecord += lastMode -> current_status() + " > ";
    energyRecord += to_string(energy) + " > ";
    oxygenRecord += to_string(oxygen) + " > ";
    speedRecord += to_string(speed) + " > ";
}
graphic_record(lastPos, currentPos);
lastPos = currentPos; // for the graphic movement
```

빠져나온 이후에는 mode_change를 해야 하는지 check하도록 하고, 만약 해야한다면, solarChange bool값을 다시 초기화해주고, energy 혹은 oxygen의 값이 0보다 작다면 다시 0으로 설정해주도록 한다. 더불어서, 현재 mode, energy, oxygen, speed 값들을 블랙박스를 위해서 다시 string들에 기록해둔다. 더불어 graphic 역시 기록하도록 한다.

```
bool mode_change(int energy, int oxygen, int distance)
{
    if(energy <= 0 || oxygen <= 0 || distance == distance_sum.at(0))
    {
        if( distance == distance_sum.at(0))
        {   // if the mode ends, change the mode in vectors!
            distance_sum.erase(distance_sum.begin());
            journey.erase(journey.begin());
        }
        return true;
    }
    return false;
}
```

만약에 distance의 값이 vector에 저장된 값과 같게 된다면, 본 mode가 끝난다는 것과 같은 의미이므로, distance Vector container와 journey vector container의 값을 삭제하도록 한다. 그리고 energy, oxygen, distance 값을 확인하고, mode change가 필요하면 true 아니면 false 값을 return 한다.

```
void graphic_record(int lastPos, int currentPos)
{
    graphic = graphic.substr(0, lastPos) + graphic.substr(lastPos + 1);
    graphic = graphic.substr(0, currentPos)+ "@" +graphic.substr(currentPos);
}
```

<graphic record>

6) Final status & Blackbox record

```
string final_status(int energy, int oxygen, int distance)
{
    string finalStatus = "Final Status: \n";
    finalStatus += "Distance: " + to_string(distance) + " km" + "\n"
                + "Energy: " + to_string(energy) + "\n"
                + "Oxygen Level: " + to_string(oxygen) + "\n";
    return finalStatus;
}
```

**final status - energy, oxygen, distance

```
string finish(int energy, int oxygen)
{
    string line = "!FINISHED : ";
    if(energy >= 0 && oxygen >= 0 && journey.size() == 0)
        line.append("Arrived");

    else if(energy <= 0) line += "Energy Failure";
    else if(oxygen <= 0) line += "Oxygen Failure";
    return line;
}
```

**finish reason - arrived || energy failure || oxygen failure

```
string blackbox(string mode, string energy, string oxygen, string speed)
{
    string blackbox = "Blackbox:\n";
    blackbox += mode.substr(0, mode.length()-2) + "\n"
            +  energy.substr(0, energy.length()-2) + "\n"
            +  oxygen.substr(0, oxygen.length()-2) + "\n"
            +  speed.substr(0, speed.length()-2) + "\n"
            +  "----------------------------------------" + "\n";
    return blackbox;
}
```

** blackbox record - mode, energy, oxygen, speed

```
if(finished == false && journey.size() != 0) {
    cout << move_status(move);
    lastMode -> print(distance, energy, oxygen);
}

if(finished == true) // print final statements
{
    cout << move_status(move);
    cout << final_status(energy, oxygen, distance);
    cout << graphic << "\n\n" << finish_reason << endl;
    cout << blackbox(modeRecord, energyRecord,
                     oxygenRecord, speedRecord);
    delete lastMode;
    break;
}
```

7) mode continue or next move

finished bool 값을 체크하여서, 만약 환경의 change가 필요하다면, 마지막 status와 블랙박스를 출력하도록 하고, journey가 끝나지 않았거나, 아직 환경의 change가 필요하지 않다면, 현재 몇 km를 갔는지를 보여주고, 현재 상태를 출력할 수 있도록 한다.

3, Troubleshooting points

1. Class - 상속, 초기화

먼저, variable 초기화를 적절히 하지 않아서 처음에 제대로 된 결과 값이 나오지 않았었다. 우선, class내에서도 member initializer를 사용하여 초기화를 하였으며, 자식 class만의 고유한 값의 경우에는 setter를 사용하거나 바로 값을 0으로 초기화를 하였는데, 처음에 setter와 본 초기화 값들의 순서를 바꾸어 적어서 값이 초기화되지 않은 채 setter 내에서 그 값이 사용되어서 오류가 나는 것을 알 수 있었다. 따라서 본 값 순서와 setter 순서를 바꿈으로써 본 오류를 해결 할 수 있었다.

상속과 다형성을 이용할 때, 어느 부분을 부모 class에 두고 어느 부분을 자식 class에 두어야 하는지에 대해서 많은 고민을 할 수 있었다. 실제로 부모 class는 추상 클래스의 특성을 가지며, 자식 클래스들이 공통으로 가지는 갑을 설정해야 하는데, 본 설정 시에 oxygen의 경우에는 submarine은 이 값을 가지지 않기 때문에 본 값을 설정을 해야하는지 아닌지에 대해서도 고민이 많았고, 중복을 없애기 위해서 자식클래스만이 가지는 특성이 아니라면 넣어주려고 했다.

2. print function의 구현

Arrived 값을 설정할 때, 본 코드 논리상 print 값을 어떻게 설정해 주어야 하는지에 대해서도 처음에 오랜 시간이 걸리는 것을 알 수 있었다. 최대한 중복되는 값들을 겹치지 않도록 하기 위해서 고민을 하였고, class 안에 print 함수를 넣도록 하여서 본 문제를 해결할 수 있었다.

3. tokenize

처음 과제 스펙이었던, file input을 통해서 tokenize를 할 때에도 처음에 if, else I, else 등을 이용해서 Road, Ocean, Sky의 경우를 모두 나누어서 설정을 했었는데, 그러다보니 코드 길이가 너무 길어지고 복잡해져서 code clarity가 떨어지는 것을 알 수 있었다. 따라서 본 코드를 수정하기 위해서 겹치는 부분을 최대한 간소화해서 축소하려는 노력을 하였다.

4. Devide & Conquer

생각보다 본 프로젝트 과제를 수행할 때 코드 길이가 많이 길어져서 어떻게 간소화하고 어떻게 쪼개어서 function을 고려해야할 지에 많은 시간을 쏟았다. 생각보다, 겹치는 코드가 많았고, 오류가 나지 않으면서 본 코드를 간소화하기 위해 노력하였다. 더불어서, main 함수 내에서도 많은 코드들이 있었기 때문에 이러한 점들을 보완하기 위해서 많은 시간을 소요했다. environment가 끝나거나, 혹은 각각 상황에 맞는 블랙박스와 final status를 출력하기 위해서 역시 생각보다 많은 변수를 고려해야 했다. 더불어서 계산의 실수 혹은 bool 값의 오류가 생기지 않도록 하기 위해서 많은 노력을 했다.

5. unexpected events

먼저, 퍼센트를 구하는 부분에서 random으로 어떻게 percent를 출력할 수 있는지에 대한 생각을 오래하였다. 먼저, 랜덤으로 값을 넣되, 중복되지 않도록 100까지의 숫자를 넣어서 실제로 array 값을 이용할 때에도 그 값들이 고른 percent가 될 수 있도록 하려고 노력하였다.

6. Code Flow - main

main 함수 내에서 while 문을 통해서 진행하였는데, 이 while 문을 break하거나 continue 할 때에도 조건을 여럿 따져야 했으며, 논리 전개상의 오류가 없도록 하고, 복잡해지지 않도록 하기 위해서 여러 번의 수정을 거쳤다. 일단 next move choice가 2개라는 점 즉, 한 번에 가거나 unit 단위로 갈 수 있다는 점을 고려해야 했으며, 각각에 따라서 energy loss 혹은 oxygen, speed loss를 고려해야했기 때문에 조금이라도 계산의 실수가 있으면 안되는 것을 알 수 있었다.

4. Code Screenshots

```
TC[0] ="[R500T20H20],[S3000T10H5A2000D30],[O80T0D100W100]";// first test case
```

```
Eunjooui-MacBook:2014_19498 project_1 eumjo_o$ g++ project_1_2014_19498.cpp
Eunjooui-MacBook:2014_19498 project_1 eumjo_o$ ./a.out
PJ1. 정은주  CP-2014-19498
Mode Select(1 for EXTRA, 2 for NORMAL) :1
Choose the number of the test case (1~10) : 1
Test case #1.

Current Status: Car
Distance: 0 km
Speed: 0 km/hr
Energy: 1000
Temperature: 20 C
Humidity: 20
|@=========^^^~~~~~~~~|
Next move? (1, 2)
CP-2014-19498>1
Successfully moved to next 50 km
Current Status: Car
Distance: 50 km
Speed: 80 km/hr
Energy: 990
Temperature: 20 C
Humidity: 20
|=@========^^^~~~~~~~~|
Next move? (1, 2)
CP-2014-19498>2
Successfully moved to next 450 km
Current Status: Car
Distance: 500 km
Speed: 80 km/hr
Energy: 900
Temperature: 20 C
Humidity: 20
|=========@^^^~~~~~~~~|
Next move? (1, 2)
CP-2014-19498>1
Successfully moved to next 1000 km
Current Status: Airplane
Distance: 1500 km
Speed: 700 km/hr
Energy: 890
Oxygen Level: 80
Temperature: 10 C
Humidity: 5
```

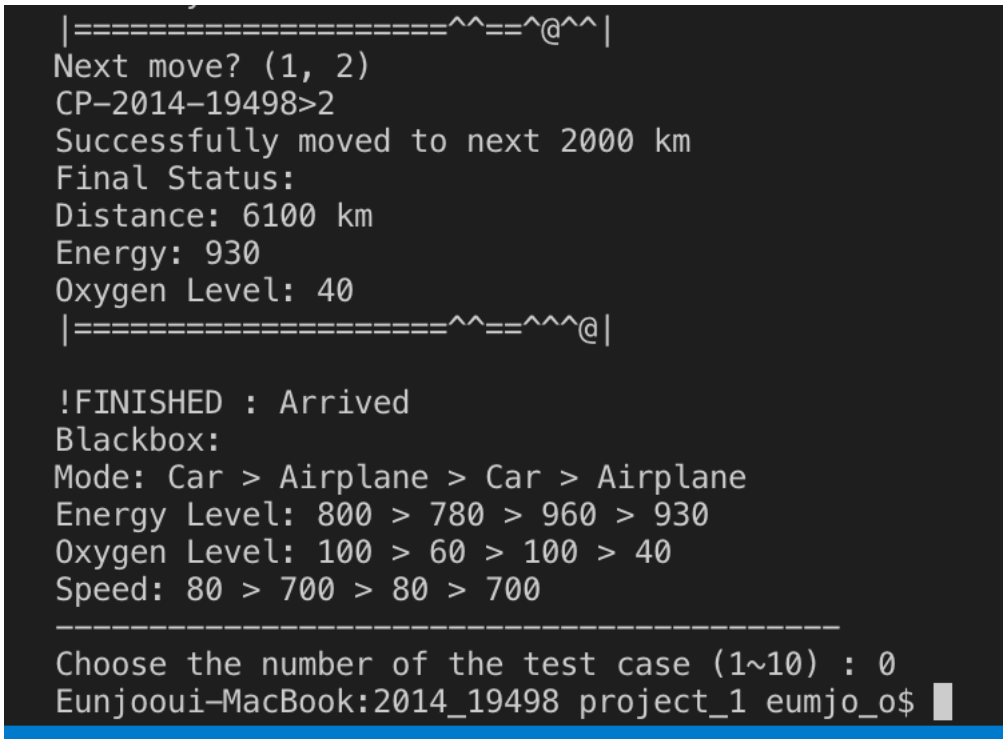
```
TC[6] ="[R500T20H20],[Y],[S3000T10H5A2000D30],[R500T20H20],
[O80T0D100W100],[S3000T10H5A2000D30]";// 7th test cases
```

```
Humidity: 5
|==========^@^^~~~~~~~|
Next move? (1, 2)
CP-2014-19498>2
Successfully moved to next 2000 km
Current Status: Airplane
Distance: 3500 km
Speed: 700 km/hr
Energy: 870
Oxygen Level: 40
Temperature: 10 C
Humidity: 5
|==========^^^@~~~~~~~|
Next move? (1, 2)
CP-2014-19498>1
Successfully moved to next 10 km
Current Status: Submarine
Distance: 3510 km
Speed: 10 km/hr
Energy: 832
Oxygen Level: 30
Temperature: 0 C
|==========^^^~@~~~~~~~|
Next move? (1, 2)
CP-2014-19498>2
Successfully moved to next 30 km
Final Status:
Distance: 3540 km
Energy: 718
Oxygen Level: 0
|==========^^^~~~~@~~~~|

!FINISHED : Oxygen Failure
Blackbox:
Mode: Car > Airplane > Submarine
Energy Level: 900 > 870 > 718
Oxygen Level: 100 > 40 > 0
Speed: 80 > 700 > 10
----------------------------------------
Choose the number of the test case (1~10) :
```

```
TC[2] ="[R1000T20H20],[S2000T10H5A2000D30],[R100T20H20],[X],
[S3000T10H5A2000D30]"; // 3rd test case
```

```
----------------------------------------
Choose the number of the test case (1~10) : 3
Test case #3.

Current Status: Car
Distance: 0 km
Speed: 0 km/hr
Energy: 1000
Temperature: 20 C
Humidity: 20
|@==================^^==^^^|
Next move? (1, 2)
CP-2014-19498>1
Successfully moved to next 50 km
Current Status: Car
Distance: 50 km
Speed: 80 km/hr
Energy: 990
Temperature: 20 C
Humidity: 20
|=@==================^^==^^^|
Next move? (1, 2)
CP-2014-19498>2
Successfully moved to next 950 km
Current Status: Car
Distance: 1000 km
Speed: 80 km/hr
Energy: 800
Temperature: 20 C
Humidity: 20
|==================@^^==^^^|
Next move? (1, 2)
CP-2014-19498>1
Successfully moved to next 1000 km
Current Status: Airplane
Distance: 2000 km
Speed: 700 km/hr
Energy: 790
Oxygen Level: 80
Temperature: 10 C
Humidity: 5
```

```
Temperature: 10 C
Humidity: 5
|====================^@^==^^^|
Next move? (1, 2)
CP-2014-19498>2
Successfully moved to next 1000 km
Current Status: Airplane
Distance: 3000 km
Speed: 700 km/hr
Energy: 780
Oxygen Level: 60
Temperature: 10 C
Humidity: 5
|====================^^@==^^^|
Next move? (1, 2)
CP-2014-19498>1
Successfully moved to next 50 km
Current Status: Car
Distance: 3050 km
Speed: 80 km/hr
Energy: 970
Temperature: 20 C
Humidity: 20
|====================^^=@=^^^|
Next move? (1, 2)
CP-2014-19498>2
Successfully moved to next 50 km
Current Status: Car
Distance: 3100 km
Speed: 80 km/hr
Energy: 960
Temperature: 20 C
Humidity: 20
|====================^^==@^^^|
Next move? (1, 2)
CP-2014-19498>1
Successfully moved to next 1000 km
Current Status: Airplane
Distance: 4100 km
Speed: 700 km/hr
Energy: 950
Oxygen Level: 80
Temperature: 10 C
Humidity: 5
```


|====================^^==^@^^|
Next move? (1, 2)
CP-2014-19498>2
Successfully moved to next 2000 km
Final Status:
Distance: 6100 km
Energy: 930
Oxygen Level: 40
|====================^^==^^^@|
!FINISHED : Arrived
Blackbox:
Mode: Car > Airplane > Car > Airplane
Energy Level: 800 > 780 > 960 > 930
Oxygen Level: 100 > 60 > 100 > 40
Speed: 80 > 700 > 80 > 700
Choose the number of the test case (1~10) : 0
Eunjooui-MacBook:2014_19498 project_1 eumjo_o$