MŰEGYETEM 1782

A PROGRAMOZÁS ALAPJAI 2.

HÁZI FELADAT DOKUMENTÁCIÓ

DUGÓFIGYELŐ RENDSZER (TMC)

KÉSZÍTETTE: NYIRI LEVENTE, ULYHQH
nyiri.levi88@gmail.com

2022. 05. 17.

TARTALOMJEGYZÉK

Felhasználói dokumentáció

A program egy dugófigyelő rendszer, amelyben a felhasználó tetszőleges helyen lekérdezheti a dugó súlyosságát.

Ehhez először szükséges példányosítani minél több Vehicle objektumot, ugyanis ezek nélkül nem sok értelme lenne dugóról beszélni. Ezeket nem célszerű egyesével létrehozni, alapértelmezetten 500 darab fog létrejönni, véletlenszerű értékekkel feltöltve (persze ésszerű határokon belül).

Az állomások hasonló módon jönnek létre, ott viszont futtatási argumentumként kapja meg a függvény, hogy ezek közül hány legyen Station, a többi pedig Moving_Station lesz (alapértelmezetten összesen 100 darab);

Ezek után a felhasználó standard inputról megadja, hogy mely időpillanatban és mely helyen kíváncsi a dugó mértékére, amire a választ a standard outputon fogja megkapni.

Például:

Adjon meg idot(ora, perc, masodperc):

3 45 32

Adjon meg egy x es egy y koordinatat:

653 -344

A dugo a (653 , -344) helyen: Enyhe

A lehetséges kimenetek:

Nagyon durva, Durva, Enyhe, Nincs dugo 😊, Nincs jarmu hatotavolsagon belul.

Osztályok statikus leírása

Vector

Felelőssége

Egy x és egy y koordináta tárolása.

Attribútumok

Védett

double xCoord ez az x koordináta

double yCoord ez az y koordináta

Metódusok

Publikus

Vector(double xCoord = 0, double yCoord = 0) Ez az osztály konstruktora

void setXCoord(double xCoord)

void setYCoord(double yCoord)

double getXCoord()const { return xCoord; }

double getYCoord()const { return yCoord; }

Ezek pedig basic setter és getter függvények.

Velocity

Felelőssége

A Moving_Station osztály tartalmazza, annak a sebességét és sebességének az irányát és kezdeti tartózkodási helyét tartalmazza.

Ősosztályok

A Vector osztályból öröklődik, mivel tudnia kell tárolni koordinátákat.

Attribútumok

Privát

double speed; Az állomás sebessége

Vector startingCoords; A kezdeti koordináták

Vector directionCoords; A haladás irányának a koordinátái

Metódusok

Publikus

Velocity(double xCoord=0,double yCoord=0,string direction ="NOTHING", double speed = 0); Konstruktor

void setSpeed(double speed)

void setStartingCoords(double xCoord, double yCoord)

void setDirectionCoords(double xCoord, double yCoord)

double getStartingCoordsX() const { return startingCoords.getXCoord(); }

double getStartingCoordsY() const { return startingCoords.getYCoord(); }

double getDirectionCoordsX() const { return directionCoords.getXCoord(); }

double getDirectionCoordsY() const { return directionCoords.getYCoord(); }

double getSpeed() { return speed; }

Getter és setter függvények.

Time

Felelőssége

A Moving_Station-ok pillanatnyi helyzetéről nincs értelme beszélni, hogyha idő nincsen definiálva, ez az osztály erre való.

Attribútumok

Privát

unsigned sec; másodperc

unsigned min; perc

unsigned hour; óra

Metódusok

Publikus

Time(unsigned sec = 0, unsigned min = 0, unsigned hour = 0) Konstruktor

unsigned getSec()const { return sec; }

unsigned getMin()const { return min; }

unsigned getHour()const { return hour; }

void setSec(unsigned sec)

void setMin(unsigned min)

void setHour(unsigned hour)

Eddig ezek setter és getter függvények

unsigned convertToSec()

Ezzel át lehet váltani Time típusból int-be(másodpercbe)

Vehicle

Felelőssége

A járműveket képviseli, amelyek adatai természetesen kulcsfontosságúak a dugó meghatározásában.

Attribútumok

Privát

double speed; A jármű sebessége.

Vector position; A jármű pozíciója.

Metódusok

Publikus

Vehicle(double speed = 0, Vector& position = 0) Konstruktor

void setSpeed(double speed)

double getSpeed() { return speed; }

Vector getPosition() { return position; }

void setPosition(double a, double b)

Getter és setter függvények.

double randCoord()

Ez a random adatokkal feltöltéshez fontos függvény, egy random koordinátát generál -1000 és 1000 között.

Vehicles

Felelőssége

A járművek tömbjének a kezelését könnyíti.

Attribútumok

Privát

Vehicle* p; Egy Vehicle-re mutató pointer.

unsigned elementNum; A járművek száma.

Metódusok

Publikus

Vehicles(unsigned elementNum = 0); Konstruktor

Vehicles(const Vehicles& other); Másoló konstruktor

~Vehicles() Destruktor, kell mivel a konstruktorban van dinamikus memóriafoglalás

void setElementNum(unsigned elementNum)

unsigned getElementNum()const { return elementNum; }

Vehicle* getP() const { return p; }

Setter és getter függvények.

Station

Felelőssége

A környezetében meg kell tudnia határozni a dugó súlyosságát.

Attribútumok

Védett

Vector position; Az állomás pozíciója.

double range; A hatótávolsága.

int speedLimit; Az állomás környezetében a sebesség korlát.

Metódusok

Publikus

Station(double range = 0, double xCoord = 0, double yCoord = 0, int speedLimit = 0) Konstruktor

Station(const Station& other) Másoló konstruktor

void setRange(double range)

void setSpeedLimit(int speedLimit)

double getRange() { return range; }

double getSpeedLimit() { return speedLimit; }

Vector getPosition() { return position; }

Setter és getter függvények.

double averageSpeed(const Vehicles& p);

A hatótávolságán belül eső járművek átlagsebességét számolja ki.

string severityOfJam(double average);

A hatótávolságába eső járművek átlagsebessége és a környezetében a sebesség korlát alapján meghatározza a dugó súlyosságát.

virtual string type() { return "Station"; }

Az osztály típusát adja vissza, heterogén kollekciónál van jelentősége.

Moving_Station

Felelőssége

A környezetében meg kell tudnia határozni a dugó súlyosságát , bárhol is legyen éppen.

Attribútumok

Védett

Velocity velocity; Az állomás sebesség-vektora.

static Time time; Az osztály összes objektumához tartozó idő.

Time timestamp; Az osztály „saját ideje”, az aktuális pozíciójának megvizsgálása után egyenlő lesz a time-al.

Ősosztályok

Mivel a Moving_Station a Station viselkedését bővíti ki, ezért belőle örököltetem.

Ennek a heterogén kollekció miatt kiemelt fontossága van.

Metódusok

Publikus

Moving_Station(const Velocity& velocity=0, const Time& timestamp=0, double range = 0, int speedLimit = 0) Konstruktor

Velocity getVelocity() { return velocity; }

Time getTime() { return time; }

```
void setVelocity(const Velocity& V)
```

```
static void setTime(unsigned hour, unsigned min, unsigned sec)
```

```
void setTimestamp(const Time& time)
```

Getter és setter függvények.

```
void actualPositon();
```

Megnézi, hogy a létrehozásának ideje óta eltelt idő alatt mennyit haladt és milyen irányba, ezt hozzáadja a kiindulási koordinátáihoz.

```
double averageSpeed(const Vehicles& p);
```

Mint a Station-nél, kiszámolja a hatótávolságába eső járművek átlagsebességét, itt viszont belül meghívja az actualPosition()-t, hogy a megfelelő helyen számolja ezeket.

```
string type() { return "Moving_Station"; }
```

Az osztály típusát adja vissza, heterogén kollekciónál van jelentősége.

```
double randCoord()
```

Egy random koordinátát generál -1000 és 1000 között

```
string teszt_randomolo();
```

Egy random irányt generál(8 lehetőség).

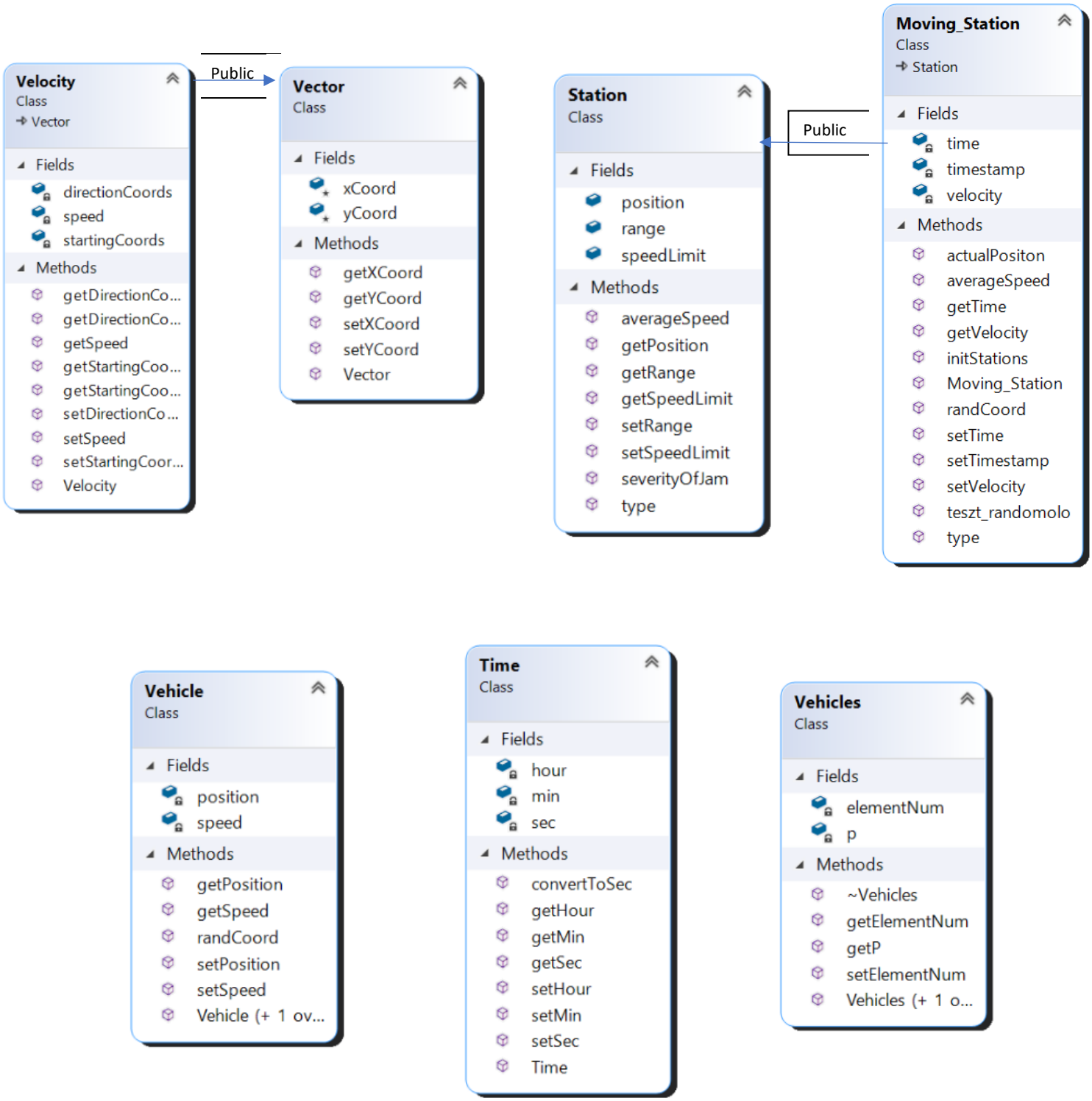
```
Station** initStations(Station** tomb, int n);
```

Stationoket és Moving_Stationoket hoz létre(100 darabot) és feltölti őket véletlenszerű adatokkal, majd az ezekre mutató pointerekkel feltölti a Station** tomb-öt és visszaadja.

Ha például függvény hívásnál n helyére 34-et írunk, akkor 34 Station-t és 66 Moving_Station-t fog létrehozni.

UML osztálydiagramm

(Nem találtam üres nyilat öröklléshez.)



Összegzés

Mit sikerült és mit nem sikerült megvalósítani a specifikációból?

Mindent sikerült megvalósítani a specifikációhoz képest, ami változott, hogy a kérdéses időt és pozíciót nem futattatási argumentumként várja a felhasználótól, hanem standard bemeneten olvassa.

Az állomásokhoz végül nem külön classt csináltam, amely tud többet is tárolni, hanem heterogén kollekcióval oldottam meg.

Mit tanultál a megvalósítás során?

A legnagyobb nehézséget a heterogén kollekció megvalósítása és a rand() függvény okozta. Egy rosszul elhelyezett srand(time(0)) rengeteg fejfájást okozott, de most már legalább tudom pontosan, hogy, hogyan működik.

Továbbfejlesztési lehetőségek

Ki lehetne bővíteni térképekkel, konkrét útvonalakkal, azok mentén is le lehetne kérdezni a dugót. Meg lehetne csinálni, hogy a dugó okát is megmondja.

Grafikus megjelenítést is lehetne hozzá készíteni, amely az utakon különböző színekkel jelezné a dugó súlyosságát. Ilyenek persze már léteznek.

Képernyőképek a futó alkalmazásról