MŰEGYETEM 1782

A PROGRAMOZÁS ALAPJAI 2.

HÁZI FELADAT DOKUMENTÁCIÓ

DRAG RACING

KÉSZÍTETTE: BUJTOR BÁLINT, G3M1AW
bujtorbalint1998@gmail.com

KÉSZÍTÉS FÉLÉVE: 2017/18/2

TARTALOMJEGYZÉK

Felhasználói dokumentáció

Az alkalmazás használata rendkívül egyszerű. Némi angol nyelvtudás és minimális felhasználói logika szükséges csupán hozzá. A program indulásakor, egy nyitófelület fogad, ahol kiválaszthatjuk az első versenyjárművünket. A bemeneten meg kell adni a kiválasztott jármű sorszámát. FONTOS, hogy minden esetben számot adjunk meg, ugyanis ezekre működik helyesen a program. A felhasználói élményt és a hasznosan töltött időt növeli, ha a felhasználó a kijelzőn megjelenő bemenetek közül választ, ekkor ugyanis nem aktiválódik a hibakezelés az érvénytelen bemenetre, így gyorsabb lesz a játék működése. Miután sikeresen megvásároltuk az első járművünket a program automatikusan „nevez” minket az első versenyünkre, melynek eredményéről tájékoztat is minket.

Ezután bekerülünk a menübe, ahol több lehetőség közül is választhatunk:

1 – Race: Logikusan ezt a menüpontot választva újabb versenyen veszünk részt. Egy verseny során a felhasználó benzintartálya eggyel csökken (kezdetben öt, negatív nem lehet és nem is lesz) illetve győzelem esetén a Budget-je 20 egységgel nő.

2 – PrintData: Szintén egyértelmű menüpont, itt van lehetőség a játékos adatainak közzétételére.

3 – Shop: a Shopban van lehetőség új jármű, upgrade, illetve benzin vásárlására. Mindnek megvan az ára és ha a usernek nincs rá pénze, akkor nem is fogja tudni megvásárolni azt. Járművek vásárlásakor további megkötés, hogy legfeljebb minden járműtípusból maximum egyet-egyet vásárolhatunk (tehát maximum 3 járművünk lehet). Ennek az az oka, hogy nincs értelme egy adott típusból több példányra, hiszen az adott típusok nem különböznek csak abban, hogy mennyi idő alatt teljesítik a negyedmérföldes távot. Ha pedig van egy már tuningolt pl. Car-unk, akkor butaság lenne egy lassabb ugyanolyan Car-ra költeni a pénzünket. (hiszen nincs Köenigsegg és Trabant, minden autó alaphelyzetben ugyanolyan gyors, csak fejlesztéssel javítható az időeredménye.)

0 – Exit: ha a felhasználó abba akarja hagyni a játékot. Fontos, hogy ekkor az eddigi progress elveszik.

Osztályok statikus leírása

Vehicle

Felelőssége

Egy általános járművet reprezentáló absztrakt osztály. Ezzel a járművel (ezzel nem hiszen absztrakt) lehet negyedmérföldes versenyeken indulni.

Attribútumok

Protected (hogy a leszármazottak is elérhessék)

- maxSpeed: a jármű maximális sebessége m/s-ban
- acceleration: a jármű gyorsulása 0-100-ra (km/h-ra) s-ban
- QUARTER_MILE: statikus konstans, egy negyedmérföld hossza méterben mérve.

Metódusok

Publikus

- Vehicle(double=10, double=30): konstruktor, alapértelmezett paraméterekkel. Az első az acceleration-é, a második a maxSpeed-é.
- void upgradeAcceleration(): a nevéből adódóan a gyorsulást javítja. (*=0.9)
- void upgradeMaxSpeed(): a maximális sebességet növeli (+=10)
- double getAcceleration() const: visszaadja a gyorsulást.
- double getMaxSpeed() const: visszaadja a maximális sebességet.

- virtual void print() const = 0: diagnosztikai kiíratót függvény, ami tisztán virtuális.
- virtual double time() const = 0; // a negyedmérföldes időeredményt számító függvény, tisztán virtuális, mert mindegyik leszármazottnál másképp generálódik a számított idő.
- virtual void upgradeCoefficient() = 0: szintén tisztán virtuális függvény ami az adott osztály koefficensét javítja. Mindegyik osztályban máshogy.
- virtual double getCoefficient() const = 0: a leszármazott osztály koefficensét adja vissza.

Car

Felelőssége

Egy tuningolható versenyautót megvalósító osztály.

Ősosztály

Vehicle – mert az autó is egy jármű, van végsebessége, gyorsulása, használja a virtuális függvényeit.

Attribútumok

Privát:

- nitro: a Car osztály koefficense, ami a gyorsulást javítja, a time függvényben. Pozitív szám, sosem lesz kisebb, mint nulla, alapértelmezetten egy.

Metódusok

Publikus:

- Car(double = 1, double = 10, double = 30) : konstruktor, a nitro koefficenst 1-nek állítja alapértelmezetten.
- void upgradeCoefficient(): a nitrot szorozza 0.9-el → jobb lesz a gyorsulása
- double getCoefficient() const: visszaadja a nitrot
- void print() const: diagnosztikai kiíratót függvény
- double time() const: negyedmérföldes időt számító függvény
- Car& operator=(const Car&): assignment operator túlterhelés, a debuggoláshoz kellett, kellhet.

Boat

Felelőssége

Egy tuningolható versenycsónakot megvalósító osztály.

Ősosztály

Vehicle – mert a csónak is jármű, használja a virtuális függvényeit.

Attribútumok

Privát:

- blade: a hajó koefficense, a végleges negyedmérföldes időt javítja

Metódusok

Publikus:

- Boat(int = 1, double = 10, double = 30): konstruktor, a blade-t alapértelmezetten egynek állítja, az ős konstrukorát a maradék paraméterekkel hívja.
- double getCoefficient() const: visszaadja a blade-t
- void upgradeCoefficient(): 0.9-el szorozza az időeredményt, javítva azt

- void print() const: diagnosztikai kiíratő függvény
- double time() const: negyedmérföldes időeredményt számoló függvény
- Boat& operator=(const Boat&): operator= a debuggoláshoz kellett/kellhet.

Motorbike

Felelőssége

Versenymotort megvalósító osztály, amit lehet fejleszteni.

Ősosztály

Vehicle – mert a motor is egy jármű, örökli annak tulajdonságait, használja annak függvényeit.

Attribútumok

Privát:

- weight: a maximális sebességet javító koefficens.

Metódusok

Publikus:

- Motorbike(int = 0, double = 10, double = 30): konstrukort, a weight koefficenst alapértelmezetten 0-val inicializálja, a többi az ős konstruktorának hivásához kell, megfelelően
- double getCoefficient() const: visszaadja a koefficenst
- void upgradeCoefficient(): noveli a koefficenst(weight += 20)
- void print() const: diagnosztikai kiíratő fv.
- double time() const: negyedmérföldes időeredményt számoló fv.
- Motorbike& operator=(const Motorbike&): operator= a debuggoláshoz használtam

Garage

Felelőssége

A járművek heterogén kollekcióját megvalósító osztály.

Attribútumok

Privát:

- int vehicleCount: a tömbben levő járművek száma
- Vehicle* garage[3]: heterogén kollekció, aminek három eleme lehet. Az, hogy miért ennyi már ismertettem.

Metódusok

Publikus:

- void printVehicles() const: egy for ciklusban meghívja minden járműnek a print() fv-ét
- int getVehicleCount() const: visszaadja a tömbben levő járművek számát.
- Vehicle* operator[](int) const: visszaadja az adott járműre mutató pointert.
- ~Garage(): destruktor, felszabadítja a memóriát.
- Garage(): konstruktor, vehicleCountot inicializálja 0-val
- template <class T> bool buy(T pvehicle): template fv., ami lehetővé teszi egy tetszőleges Vehicle típus megvásárlását és annak betételét a tömbbe, ha már van adott jármű a tömbben akkor false-al tér vissza, ekkor nem volt sikeres a várárlás

User

Felelőssége

Versenyzőt megvalósító osztály.

Attribútumok

Privát:

- double budget: tőke, ebből tud vásárolni.
- int fuel: üzemanyagegységek száma.

Publikus:

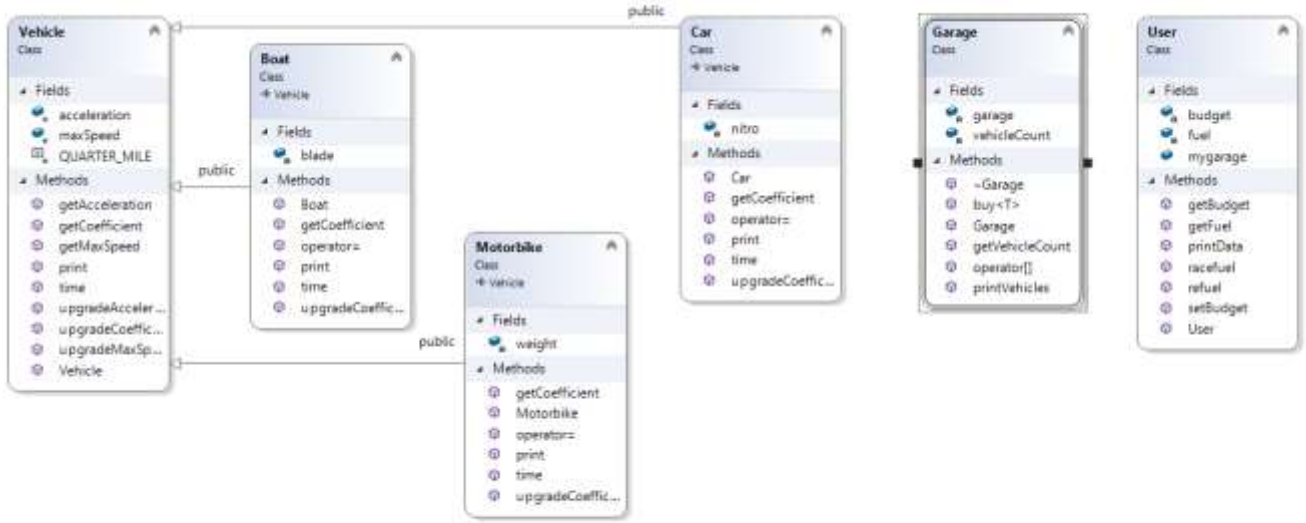
- Garage mygarage: a user járműveinek heterogén kollekciója. Privátként nem tudták elérni, problémás lett volna a getterek, setterek stb. függvények megírása.

Metódusok

Publikus:

- User() : konstruktor – budget = 120, fuel = 5
- double getBudget() const: visszaadja a budget-t
- int getFuel() const: visszaadja a fuelt
- void setBudget(double): a paraméterként kapott értéket hozzáadja a budgethez. Negatív érték kezelve van.
- void refuel(): tankolo fv: -50 egység pénz +5 egység fuel
- void racefuel(): csökken eggyel a fuel – versenyzéskor egy egységet használ el
- void printData() const: diagnosztikai kiíró fv.

UML osztálydiagramm



Összegzés

Mit sikerült és mit nem sikerült megvalósítani a specifikációból?

A fame implementálására sajnos nem jutott időm, pontosabban már a program írásának az elején rájöttem, hogy ez sokszorosára növelné a program komplexitását és méretét, hiszen figyelembe kellett volna vennie a winstreak-et a user fame-ét, továbbá az éppen versenyzett jármű fame-ét. Ebből lehetett volna kikalkulálni a fame alapú versenynyeremény rendszert. A többi funkció implementálása azt gondolom, hogy sikeres volt.

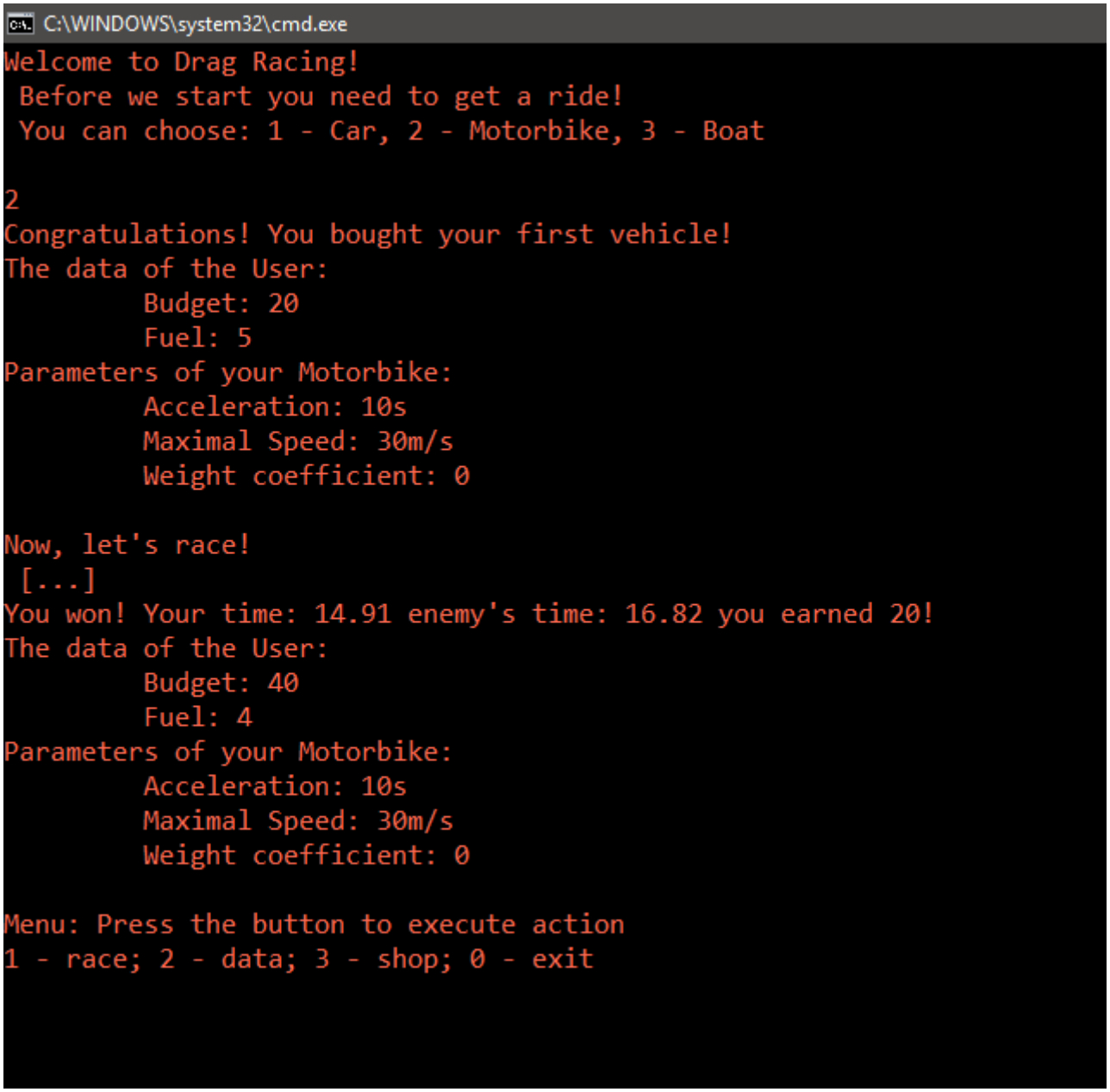
Mit tanultál a megvalósítás során?

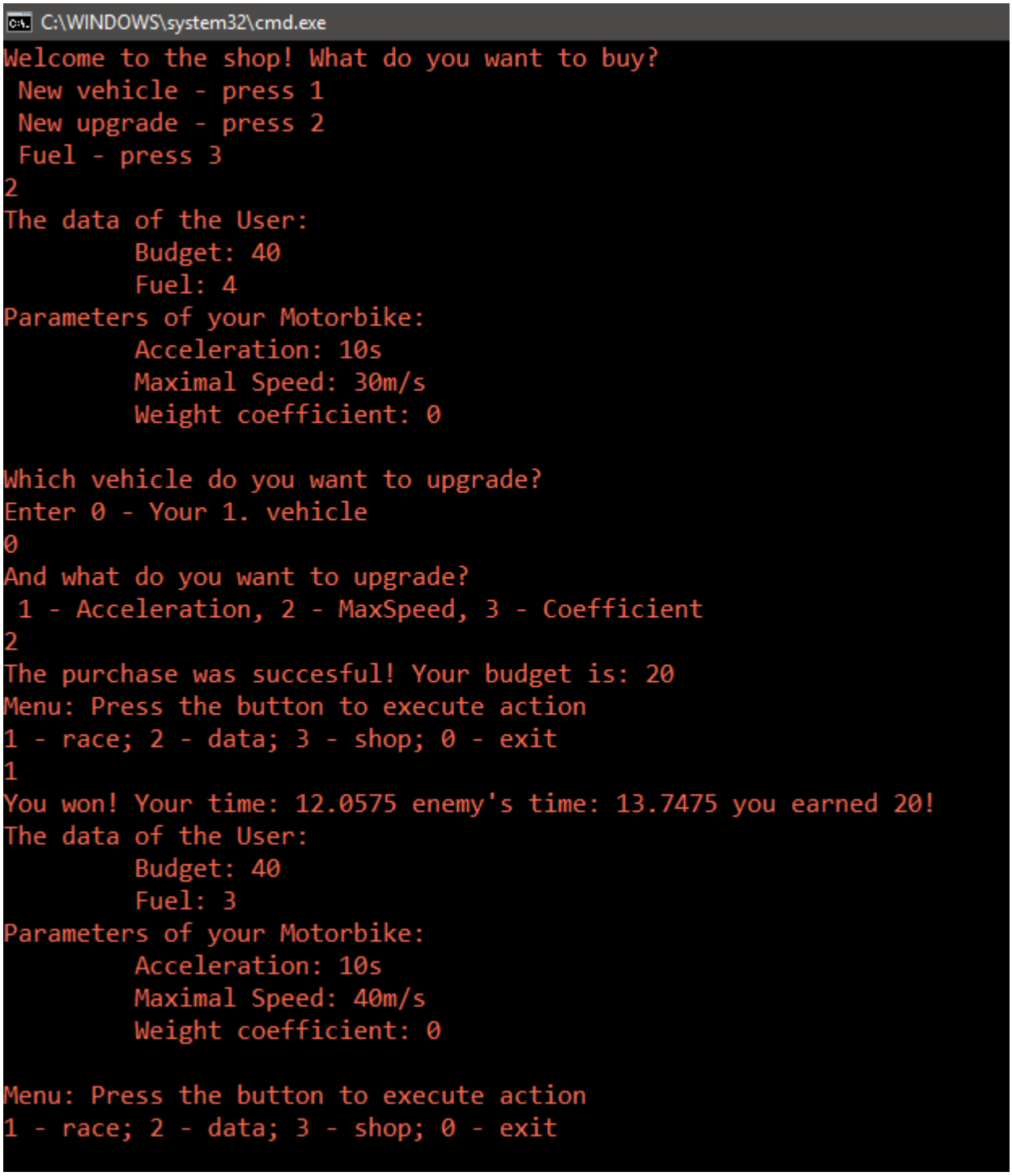
A heterogén kollekció valóban hasznos, de rendkívül nehéz megcsinálni működőképesre. A program írására fordított időnek az 1/3-át, felét ez tette ki. Ezután nagyobb problémába nem ütköztem, izgalmas volt látni, hogy a tanultaknak valóban használt is veszem a gyakorlatban is. (operator overloading, templatek, exception handling, pure virtual function, öröklés) Az viszont kevésbé tetszett, hogy miután összeállt a program és csak a működést kellett megírni, akkor nagyon unalmassá vált a switch case-k és a do-while-ok megírása.

Továbbfejlesztési lehetősége

Lehetőség lehetne különböző kezdeti szintű járművek vásárlására is (pl. level 1 car vagy level 3 car ~ Fiat vagy Ferrari). Ekkor már lenne értelme annak is, hogy egy adott típusból több példány is álljon a user garázsában. További fejlesztési lehetőség lehet, ha elmenthető a progress egy szöveges fileba és onnan következő használatkor vissza lehessen állítani. Továbbá a menü struktúráját is lehetne fejleszteni, optimalizálni.

Képernyőképek a futó alkalmazásról




C:\WINDOWS\system32\cmd.exe
Welcome to the shop! What do you want to buy?
New vehicle - press 1
New upgrade - press 2
Fuel - press 3
2
The data of the User:
Budget: 40
Fuel: 4
Parameters of your Motorbike:
Acceleration: 10s
Maximal Speed: 30m/s
Weight coefficient: 0
Which vehicle do you want to upgrade?
Enter 0 - Your 1. vehicle
0
And what do you want to upgrade?
1 - Acceleration, 2 - MaxSpeed, 3 - Coefficient
2
The purchase was succesful! Your budget is: 20
Menu: Press the button to execute action
1 - race; 2 - data; 3 - shop; 0 - exit
1
You won! Your time: 12.0575 enemy's time: 13.7475 you earned 20!
The data of the User:
Budget: 40
Fuel: 3
Parameters of your Motorbike:
Acceleration: 10s
Maximal Speed: 40m/s
Weight coefficient: 0
Menu: Press the button to execute action
1 - race; 2 - data; 3 - shop; 0 - exit