Eseményvezérelt programozás
1. beadandó dokumentációja

Kiss-Bartha Nimród

2023. november 9.

Tartalomjegyzék

1. fejezet

Probléma

1.1. Készítette

Név: Kiss-Bartha Nimród
Neptun-kód: AP3558
E-mail: email.kbnim@gmail.com

1.2. Feladat

Gyorsulás (9-es feladat)

Készítsünk programot, amellyel az alábbi motoros játékot játszhatjuk. A feladatunk, hogy egy gyorsuló motorral minél tovább tudjunk haladni. A gyorsuláshoz a motor üzemanyagot fogyaszt, egyre többet. Adott egy kezdeti mennyiség, amelyet a játék során üzemanyagcellák felvételével tudunk növelni.

A motorral a képernyő alsó sorában tudunk balra, illetve jobbra navigálni. A képernyő felső sorában meghatározott időközönként véletlenszerű pozícióban jelennek meg üzemanyagcellák, amelyek folyamatosan közelednek a képernyő alja felé. Mivel a motor gyorsul, ezért a cellák egyre gyorsabban fognak közeledni, és mivel a motor oldalazó sebessége nem változik, idővel egyre nehezebb lesz felvenni őket, így egyszer biztosan kifogyunk üzemanyagból. A játék célja az, hogy a kifogyás minél később következzen be.

A program biztosítson lehetőséget új játék kezdésére, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem mozog semmi a játékban). Ismerje fel, ha vége a játéknak, és jelenítse meg, mennyi volt a játékidő. Ezen felül szüneteltetés alatt legyen lehetőség a játék elmentésére, valamint betöltésére.

1.3. Elemzés

- A játék főszereplője a **motorkerékpár** (vagy motoros). Rendelkezik **gyorsaság**gal és **üzemanyagtartálly**al. A motoros csak jobbra vagy balra tud mozdulni a képernyőn a megfelelő nyílbillentyűk lenyomásával. A „fel” és „le” billentyűkkel képes felgyorsulni és lelassulni.
- Az ablak tetején szabályos időközönként (önkényesen legyen 2 mp-ként) megjelennek ún. **üzemanyagcellák**, véletlenszerű x koordinátában. Ezek a motor gyorsaságával egyenes arányosságban haladnak lefelé. Ha a motor és a cella „*elég közel vannak egymáshoz*”, a cella eltűnik és az általa hordozott mennyiséggel megnő a motor tartályának töltöttségi szintje. Ha nem sikerül elkapnia időben, a képernyő alján eltűnik.

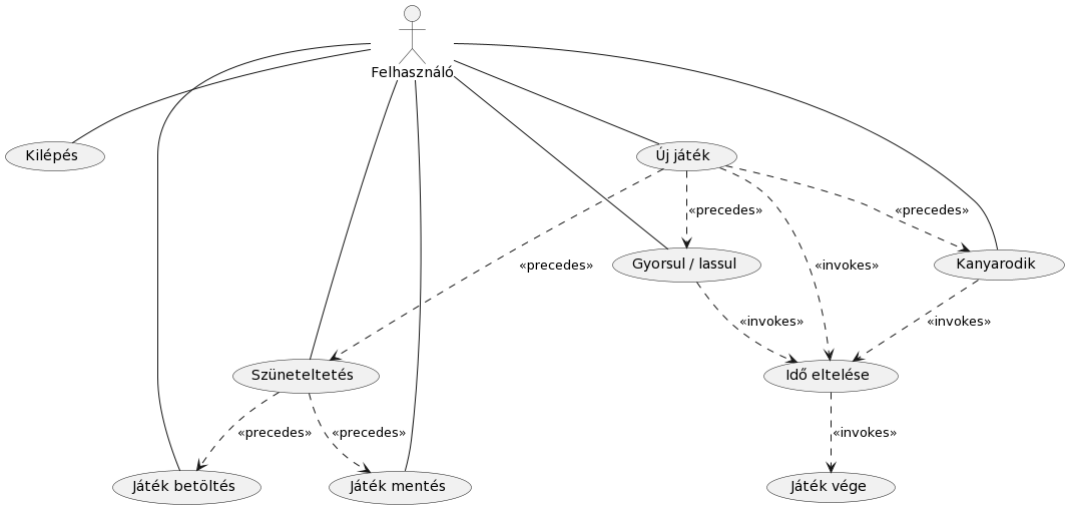
- Rögzítsük, hogy három sebességet vehet fel a motor: *lassú* (`Slow`), *közepes* (`Medium`) és *gyors* (`Fast`). Minél gyorsabban hajt, annál több üzemanyagot fogyaszt. Ha kiürül a tartály, a játék véget ér.

- A felületet Windows Form grafikus felülettel készítjük el. A program vezérlése elsődlegesen billentyűzet-alapú, minimális egérműveletet igényel a használata.

- Egyetlen főablakból fog állni a program, melyen különböző panelek fognak megjelenni a különböző fázisaiban a játéknak. A fázisok az alábbiak.

    1. Első indítás vagy új játék kezdése.
    Feltünteti a játék nevét (címke), kiírja az elvárt utasítást az indításhoz (címke), alatta meg a „*New*”, a „*Load*”, a „*Help*” és a „*Quit*” gombok helyezkednek el rendre. A „*Load*” egy korábbi mentésünket tölti be, egy ennek megfelelő dialógusablakot jelenít meg. A „*Help*” egy új dialógusablakban megjeleníti a program billentyűkombinációit. A szóköz lenyomásával új menetet tudunk indítani.
    2. Játék szüneteltetése.
    Ezt a viselkedést a szóköz újbóli lenyomásával válthatjuk ki és oldhatjuk fel ismét. Megjelenik 5 gomb: „*Resume*”, „*Load*”, „*Save*”, „*Help*” és „*Quit*”. Szüneteltetés közben el lehet menteni az aktuális játékmenetet a megfelelő dialógusablakkal.
    3. Játék vége (kifogy az üzemanyag).
    A panel megjeleníti az aktuális menet időtartamát, a legjobb (betöltött) menet idejét, valamint új játék indítása mellett betöltésre, mentésre, súgóra és kilépésre ad lehetőséget.

- Maga a játék felülete (panelek nélkül) a pálya hátteréből, a motorból, a megjelenő üzemanyagcellákból és az aktuális statisztikákat megjelenítő címkékből áll. A szebb megjelenést a szöveges leírás helyett ikonok, valamint a tartály szintjére vonatkozó folyamatjelző sáv segíti.



1.1. ábra. Felhasználói esetek diagramja

2. fejezet

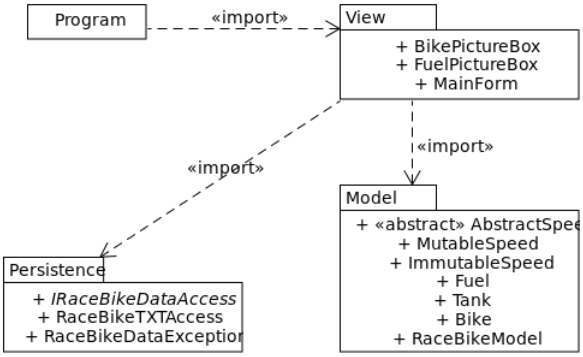
Megvalósítás

2.1. Tervezés

Az alkalmazást három rétegben valósítjuk meg. Lesz egy **modell** réteg, ami a háttérben lévő játékmechanikát biztosítja. Az adatok betöltéséért és mentéséért a **perzisztencia** réteg felel (ennek nem kell ismernie a modell működését, tőle függetlenül működik). Végül a **nézet** rétege grafikus felületet nyújt az alábbi két réteghez.

A rétegekre szabdalás követi az objektumorientált programozás SOLID-elvei közül a „*single responsibility*” elvet, valamint megkönnyíti az alkalmazás karbantarthatóságát és lehetővé teszi, hogyha kedvünk úgy kívánja, a grafikus felületét lecseréljük. Mi ezt a Windows Forms technológiával kivitelezzük.

Az alkalmazás két projektből áll: az első a `Model` és a `Persistence` rétegeket, névtereket tartalmazza, mappákba elkülönítve[1], a második meg a felhasználói felület implementációját foglalja magába.



2.1. ábra. Az alkalmazás csomagdiagramja

[1] A `Model`-ben van még egy `Classes` mappa, ami a kisebb komponenseket tartalmazza.

2.1.1. Perzisztencia

- A játék `TimeSpan` típusú adatokból olvassa be a korábbi mentéseinket, valamint ugyanilyen formátumban menti el nekünk. Mindkét esetben `*.txt` fájlformátumú a szóban forgó fájl.

  Példa bementi / kimeneti fájlra

  ```
  00:01:20.0209835
  ```

- Létrehoztunk egy `IRaceBikeDataAccess` interfészt, amely tartalmazza a két alapvető funkcionalitást. A betöltés aszinkron módon zajlik, a mentés a folyamat rövidsége révén nem. Ennek előnye, hogy a későbbiekben bővíthetjük a programot és akár más fájlformátumokat és képes lehet kezelni.

- Eme interfészből származtatjuk a `RaceBikeTXTAccess` osztályt, ami implementálja a metódusokat úgy, hogy `*.txt` fájlokat kezelni képes legyen.

- Abban az esetben, ha a fájl hibás formátumú, fájlkiterjesztésű vagy nem létező helyre mutat az elérése, akkor a program egy `RaceBikeDataException` formájában hibaüzenetet dob a felhasználónak.

- A szöveges fájl egyetlen sorból áll, amely az időt tárolja. Ha utána új sorban saját magunk írunk tetszőleges szöveget, a program at ignorálja. Ha a módosításunk ugyanabban a sorban történik, amelyben az időtartam található, azzal korrumpáljuk a formátumot, így futás idejű hibát kapunk.

2.1.2. Modell

- A modell réteget a `RaceBikeModel` osztály valósítja meg.
  - Privát adattagok: `_bike`, `_fuels`, `_dataAccess`, `_stopwatch`
  - Tulajdonságok: `LatestBestTime`, `CurrentTime`, `CurrentSpeed`, `CurrentTankLevel`, `IsPaused`, `IsGameOver`
  - Események: `GameContinues`, `GameOver`. Nem igényelnek saját eseménykezelő-argumentumtípust.
  - A konstruktor függőségi befecskendezéssel kap hozzáférést a perzisztencia réteghez.
  - A cél az, hogy a modell belső állapotát csak a metódusokon keresztül tudjuk módosítani. Éppen ezért közvetlenül nem hívható meg pl. a `_model._bike.SpeedUp()` vagy a `_model._bike.Speed.SpeedUp()` sem. Pontosan ezért kétféle sebesség típus van (róluk később lesz szó).
  - A `LoadGameAsync()` és `SaveGame()` műveletekkel érjük el a perzisztencia réteget.
  - Mivel a játék idővel garantáltan véget ér, így a `Reset()`-tel tudunk új játékot kezdeni. A `GameTimePauseResume()` szünetelteti és folytatja a játékmenetet. A `GameTimeElapsing()` csökkenti az üzemanyagszintet (ha nem üres az), különben leállítja a stoppert és véget ér a játék.
  - Manuális intervencióra az alábbi metódusokon keresztül van lehetőségünk: `IncreaseTankLevel()`, `GenerateNewFuelItem()`, `LoseFuelItem()`, `SpeedUp()`, `SlowDown()`.

- A `Bike` osztály rögzíti a pillanatnyi sebességét és a tartálya állapotát, emelllett azokkal a műveletekkel rendelkezik, amiket az elemzésben meg a modellnél leírtam.
- A `Tank` osztálynak a maximális kapacitása 1000 egység alapértelmezetten, ami a konstruktorban paraméterezhető. Feltölteni `Fuel` típusú objektumokkal lehet, amik alapértelmezetten 100 egységet tárolnak.
- A sebesség osztályok az `AbstractSpeed` osztályból származnak, ami rendelkezik egy felülírt `ToString()` metódussal meg egy kasztoló operátorral, ami `Int32`-vé konvertálja azt.
  - A `MutableSpeed` 3 sebességet enged meg, lehet gyorsítani, lassítani, alapértelmezett értékre visszaállítani. A konstruktora paraméter nélküli. Ezt a típust használja a motorháztető alatt a `Bike` és a `RaceBikeModel`.
  - Az `ImmutableSpeed` annyiban tér el, hogy nem rendelkezik a sebességmódosító metódusokkal. Ezt kapjuk meg a modell osztály `CurrentSpeed` tulajdonságának eredményeként.

2.1.3. Nézet

- A nézetet a `MainForm` osztály biztosítja, amely tárolja a modell egy példányát (`_model`), valamint az adatelérés konkrét példányát (`_dataAccess`).
- A motorost egy `BikePictureBox` reprezentálja, ami rendelkezik az üzemanyagot elkapó metódussal. Az üzemanyagcellának `FuelPictureBox` a típusa, ami az objektum valódi középpontjával (`MidPoint` tulajdonsággal) is rendelkezik.
- A pályaelemek felett egy menü jelenik meg első indításkor (`_mainMenuPanel`) és szüneteltetéskor. Amíg el nem indítottuk a játékot vagy be nem töltöttünk egy mentést, addig a `Save` gomb nem elérhető.
- A menü aktív játékmenet közben a szóközzel hívható meg.
- A `_gameTimer` figyeli a játék haladását, ami szünetel, ha a menü meg van jelenítve.
- A `_fuelTimer` rendszeres időközönként generál egy új üzemanyagot a modellben és a képernyőn lerak egyet belőle. Fontos, hogy a grafikus üzemanyag és a modellbeli üzemanyag valójában nem ismeri egymást.
- A felületen dinamikusan jelennek meg az üzemanyagcellák, amik a sebességgel egyenesen arányosan haladnak lefelé. Ha elkapjuk őket vagy az ablak aljára kerülnek, lefut a `Dispose()` metódusuk.
- Az üzemanyag elkapása a következőképp zajlik:
  A képernyő koordinátájának x iránya megegyezik a matematikaiéval, de az y iránya pontosan az ellentettje. Lefelé haladva növekszik az y értéke.
  Legyen $b \in BikePictureBox$, $f \in FuelPictureBox$. Jelölje L a b bal fölső sarkának koordinátáját. Legyen w a szélessége, h a magassága. Továbbá jelölje M az f középpontját.

Definíció. Az üzemanyag elkaphatósága a motor által

Egy $f \in FuelPictureBox$ akkor és csak akkor elkapható a $b \in BikePictureBox$ által, ha

$$f_M \in \{(x, y) \mid p_x \leq x \leq q_x \ \wedge \ p_y \leq y \leq q_y\} =: S_c$$

ahol

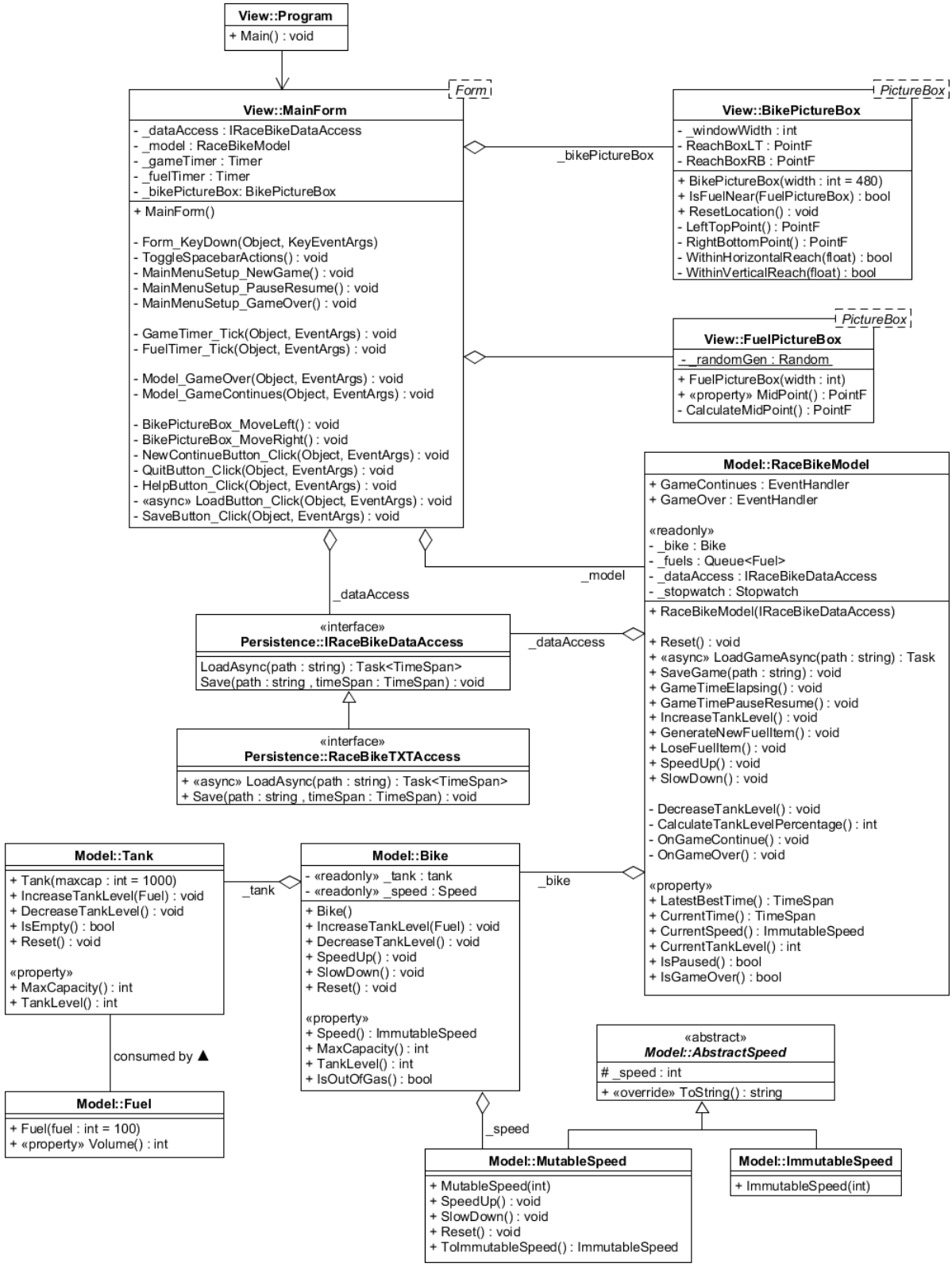
$$p := \left(L_x - \frac{w}{4}, L_y - \frac{h}{4}\right) \text{ és } q := \left(L_x + \frac{5}{4} \cdot w, L_y + h\right)$$

A halmaz jele S_c, azaz *set of catchable points* és p, q ennek a ponthalmaznak rendre a bal felső és jobb alsó végpontjai.

A q pont y koordinátáját szándékosan választottam úgy, hogy a kapott alakzat ne legyen egybevágó az eredeti képpel, ugyanis ha $q_y := L_y + \frac{5}{4} \cdot h$ lenne, akkor lenne rá esély, hogy úgy kapja el a motor az üzemanyagot, hogy gyakorlatilag lejjebb van nála. Ezt a csalási lehetőséget meg el szeretnénk kerülni.

2.2. Tesztelés

- Az alkalmazás működését egységteszteléssel ellenőriztem, melyeket a `RaceBikeModelTest` osztályban helyeztem el. A *Moq* csomag felhasználásával oldottam meg a modell és a perzisztencia szimulációját.
  - `Initialize()` – inicializálja a tesztkörnyezetet.
  - `LoadFileContents()` – privát metódus, mely a fájlbetöltést szimulálja (mokkolja). A nagyobb tesztelő metódusokban hívjuk meg.
  - `ValidInputFileTest` – fájl beolvasása, melynek tartalma `00:00:30.8101404` → helyes eredmény.
  - `InvalidInputFileTest01()` – fájl beolvasása, melynek tartalma `00:00:30.8101404ű` → kivételt dob.
  - `InvalidInputFileTest02()` – fájl beolvasása, melynek tartalma üres → kivételt dob.
  - `GamePauseResumeTest01()` – játék elindítása és megállítása, fájlbeolvasás nélkül.
  - `GamePauseResumeTest02()` – játék elindítása és megállítása, korábbi fájlbeolvasással.
  - `ChangeSpeedTest()` – a sebességváltozást ellenőrzi.
  - `RunningOutOfGasTest()` – az üzemanyagból való kimerülést teszteli.
  - `FuelConsumption()` – tankolás után hogyan változik a telítettsége.
  - `FuelConsumptionEmptyQueue()` – ha üres az üzemanyagot tároló sor, nem tud miből tankolni.
  - `LoseFuelTest()` – egyet elkap, de a másikat elveszti.
  - `LoseEachFuelTest()` – egyiket sem kapja el.
  - `TestReset()` – az alapállapotra való visszaállítást teszteli.



2.2. ábra. Az alkalmazás osztálydiagramja