Eseményvezérelt programozás
2. beadandó dokumentációja

Kiss-Bartha Nimród

2023. november 27.

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 **üzemanyagtartály**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.

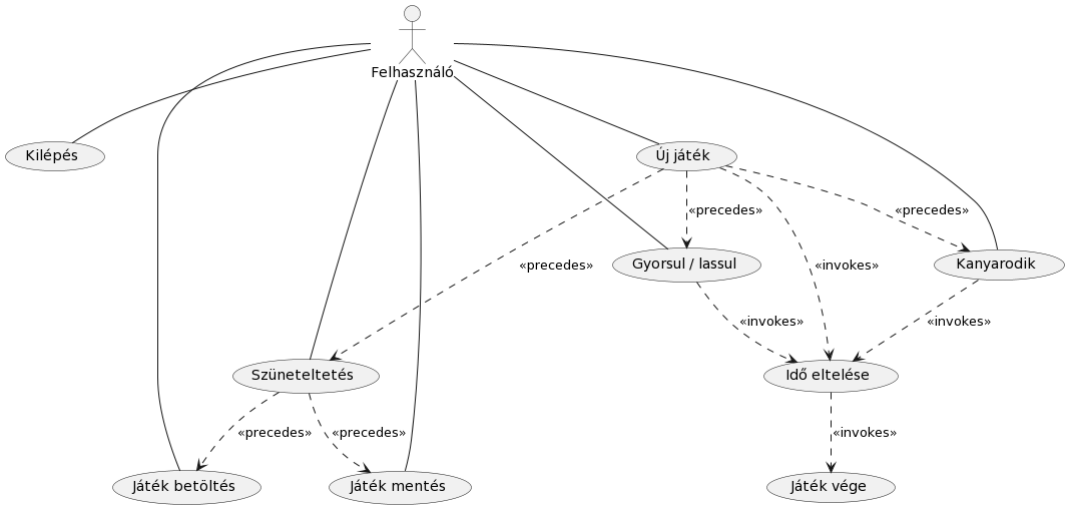
- Rögzítsük, hogy három sebességet vehet fel a motor: *lassú* (`Slow`), *közepes* (`Medium`) és *gyors* (`Fast`). Minél gyorsabban hajt, annál több üzemanyagot fogyaszt. Ha kiürül a tartály, a játék véget ér.

- A felületet Windows Presentation Foundation (WPF) grafikus felülettel készítjük el.

- 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

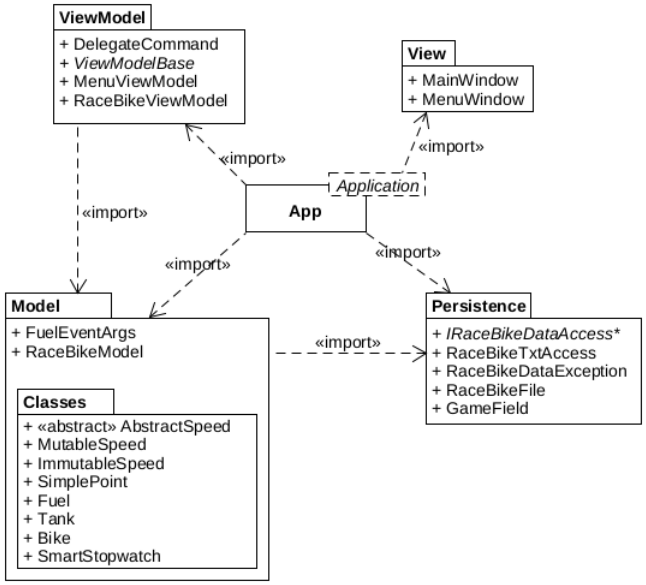
Megvalósítás

2.1. Tervezés

Az alkalmazást az MVVM (*model–view–view-model*) architektúra szerint 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). A **nézet-modell** rétege fogja közvetíteni az eseményeket a modell felé és a modell állapotváltozásait a **nézet** felé.

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 WPF 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, benne a `ViewModel` névtérrel.



2.1. ábra. Az alkalmazás csomagdiagramja

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

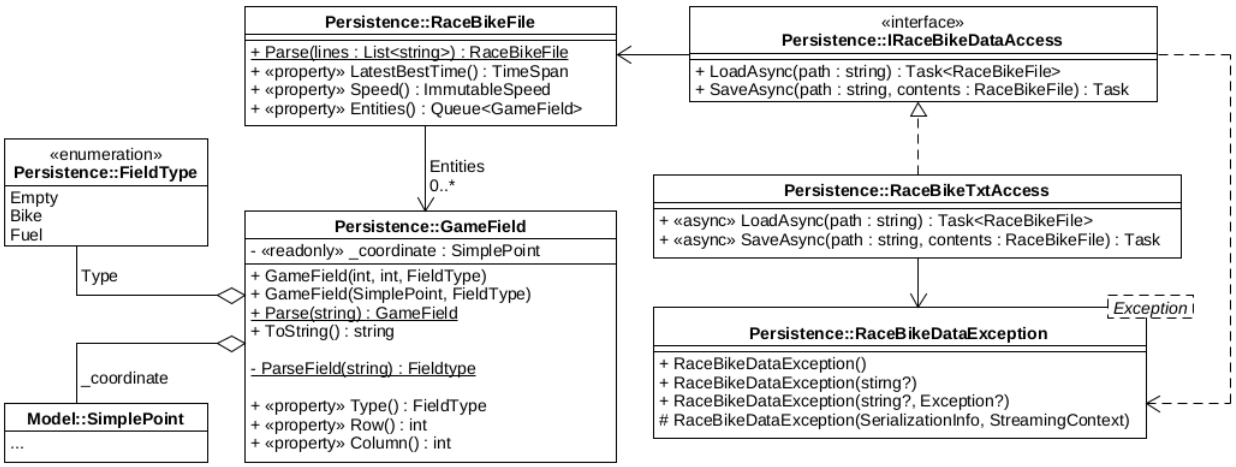
2.1.1. Perzisztencia

- A játék `RaceBikeFile` struktúrákból olvassa be a korábbi mentéseinket, valamint ugyanilyen formátumban menti el nekünk. Ezek rögzítik a legjobb időtartamot, az aktuális sebességet, valamint a pályán elhelyezkedő entitások koordinátáit. Mindkét esetben `*.txt` fájlformátumú a szóban forgó fájl.

```
00:01:20.0209835
Medium
Bike (1,20)
Fuel (30,4)
Fuel (50,60)
```

2.2. ábra. Példa bemeneti / kimeneti fájlra

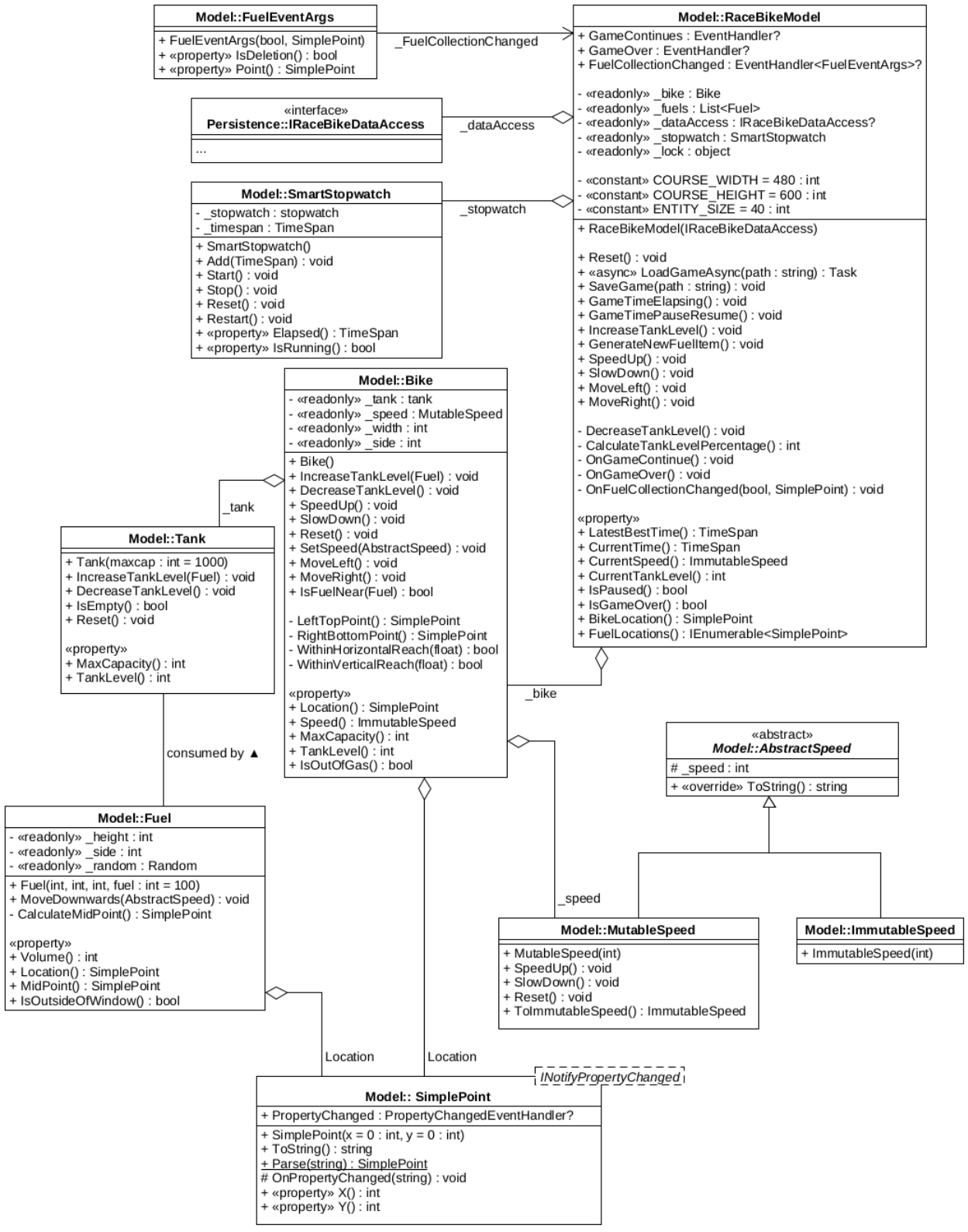
- Az `IRaceBikeDataAccess` interfész tartalmazza az aszinkron betöltést és a szinkron mentést végző metódusokat, amit a `RaceBikeTxtAccess` implementál `*.txt` fájlokra.
- Abban az esetben, ha a fájl hibás formátumú, fájlkiterjesztésű vagy nem létező helyre mutat az elérése, akkor a program egy `RaceBikeDataException` formájában hibaüzenetet dob a felhasználónak.
- A fájlnak legalább az első két soránbak meg kell lennie formázási hibák nélkül. Nem gond, ha hiányoznak a koordináták, ugyanis ebben az esetben minden marad a játék kurrens állapotában. Gondoskodjunk arról, hogy az entitások típusnevei és a koordináták között szerepeljen szóköz, valamint arról, hogy a koordinátákat elválasztó vessző előtt és után ne szerepeljen szóköz.



2.3. ábra. A *Persistence* csomag osztálydiagramja

2.1.2. Modell

- A modell réteget a `RaceBikeModel` osztály valósítja meg.
  - Privát adattagok: `_bike`, `_fuels`, `_dataAccess`, `_stopwatch`, `_lock` (a `_fuels` zárolására kell, hogy egyszerre több metódus ne tudjon hozzáférni)
  - Tulajdonságok: `LatestBestTime`, `CurrentTime`, `CurrentSpeed`, `CurrentTankLevel`, `IsPaused`, `IsGameOver`, `BikeLocation`. `FuelLocations`
  - Események: `GameContinues`, `GameOver`, `FuelCollectionChanged`. Az utolsót leszámítva (`FuelEventArgs`) nem igényelnek egyedi eseményargumentumtípust.
  - Konstansok: `COURSE_WIDTH` (pálya szélessége), `COURSE_HEIGHT` (pálya magassága), `ENTITY_SIDE` (játékbeli entitások dimenziói)
  - 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, pozícióját 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.

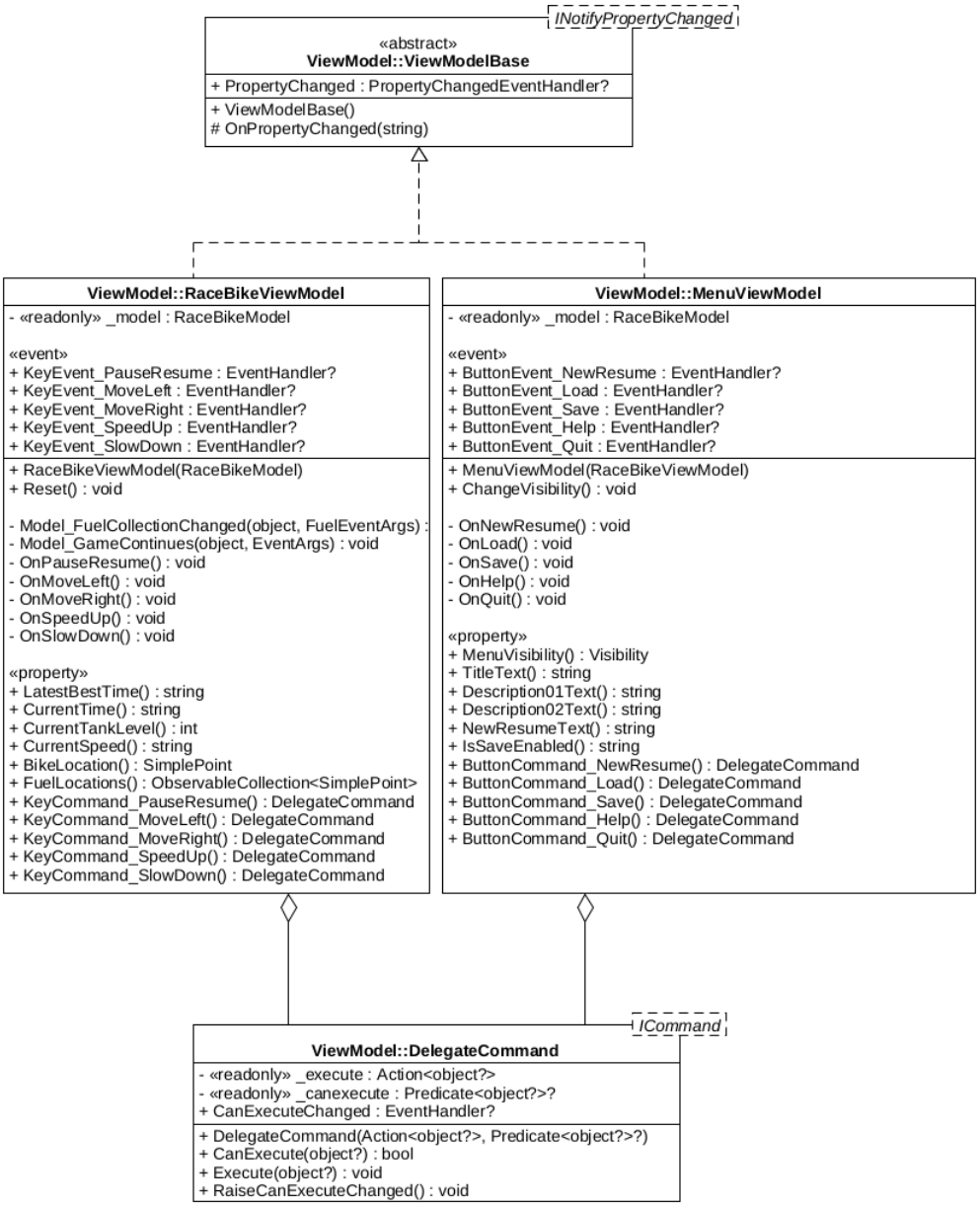


2.4. ábra. A *Model* csomag osztálydiagramja

2.1.3. Nézetmodell

- A nézetmodell megvalósításához felhasználunk egy általános utasítás (`DelegateCommand`), valamint egy ős változásjelző (`ViewModelBase`) osztályt.

- A nézetmodell feladatait a `RaceBikeViewModel` és a `MenuViewModel` osztály látja el. Az első osztály felelős a felületen megjelenítendő információk lekérdezéséért (motoros pozíciója, aktuális sebesség, stb.), míg a második a főmenü gombjainak funkcionalitásait fedi le. A parancsokhoz eseményeket kötünk, amelyek a parancs lefutását jelzik a vezérlőnek. Mindkét nézetmodell tárolja a modell egy hivatkozását (`_model`), de csupán információkat kér le tőle, direkt nem avatkozik a játék futtatásába.



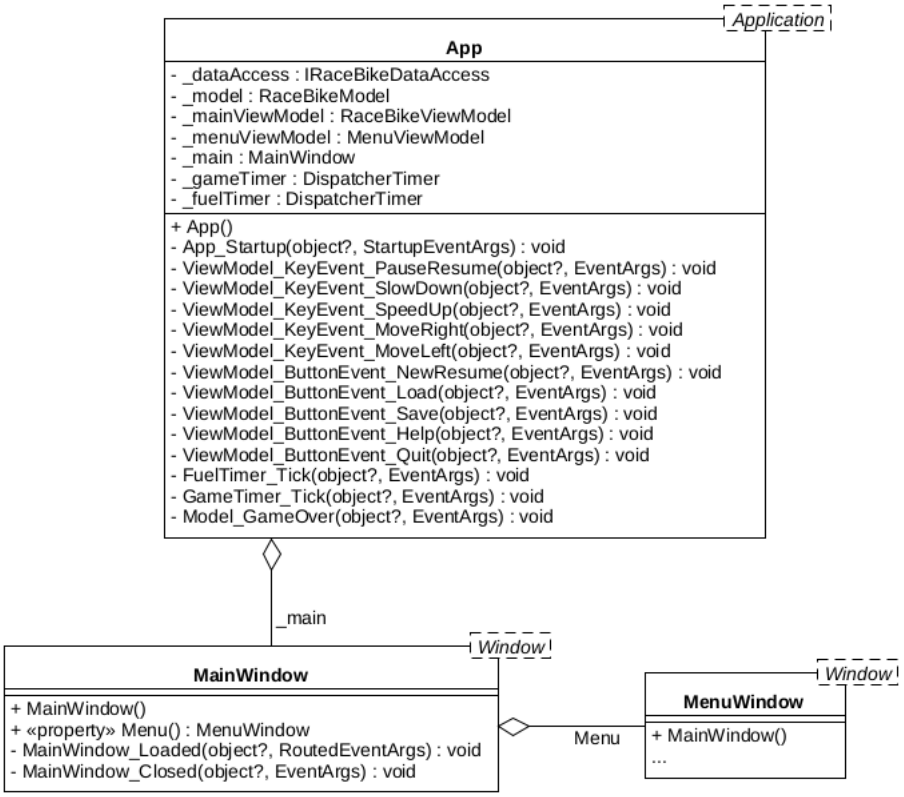
2.5. ábra. A *ViewModel* csomag osztálydiagramja

2.1.4. Nézet

- A nézetet a `MainWindow` és a `MenuWindow` osztályok biztosítják.
- A főablakon `InputBinding`-gal értük el a billentyűzeti bemenetre vonatkozó adatkötéseket. A felületi elemeket egy `Canvas`-ban helyezzük el, ami rugalmas pozicionálásra nyújt lehetőséget. Benne elhelyeztünk egy `ItemsControl`-t, ami a dinamikusan megjelenő üzemanyagcellákat kezeli.
- A menüablak megjelenik alapértelmezetten minden indulásnál. Amint új játékot indítunk vagy folytatjuk a már elkezdett menetünket, eltűnik az ablak. a szóközzel újból elő lehet hívni. Bármelyik ablakot zárjuk be, minden esetben leáll a program futása.

2.1.5. Környezet

- Az `App` osztály feladata az egyes rétegek példányosítása (`App_Startup`), összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.
- Tartalmaz két időzítőt (`DispatcherTimer`), melyek frissítik a játék aktuális állapotát (`_gameTimer`, `_fuelTimer`). Eltárolja azt a főablakot (`_main`), amely tartalmazza a közvetlenül lekérdezhető menüablak (pl. `_main.Menu.Show()`).



2.6. ábra. A vezérlés és a *View* csomag osztálydiagramja

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.
  - `ValidInputFileTest01()` – helyes formátumú fájl beolvasása (semmi nem hiányzik).
  - `ValidInputFileTest02()` – helyes formátumú fájl beolvasása (nincsenek üzemanyagok).
  - `ValidInputFileTest03()` – helyes formátumú fájl beolvasása (csak idő és sebesség).
  - `InvalidInputFileTest01()` – helytelen forműtumú fájl beolvasása.
  - `InvalidInputFileTest02()` – helytelen forműtumú fájl beolvasása.
  - `InvalidInputFileTest03()` – helytelen forműtumú fájl beolvasása.
  - `InvalidInputFileTest04()` – helytelen forműtumú fájl beolvasása.
  - `InvalidInputFileTest05()` – helytelen forműtumú fájl beolvasása.
  - `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.
  - `TestReset()` – az alapállapotra való visszaállítást teszteli.
  - `Model_GameOver()` – ellenőrzi, hogy game overnél leáll-e a játék.
  - `Model_GameContinues()` – ellenőrzi, hogy az eltelt idő nemnegatív-e.