Eseményvezérelt programozás
3. beadandó dokumentációja

Kiss-Bartha Nimród

2024. január 7.

Tartalomjegyzék

Előszó

„*Ha megírtad jól a WPF-et, nem kell sokat átírnod*” – mondták. „*Csak pár apróságra kell odafigyelni*” – mondták... Hát, ehelyett elölről kellett kezdenem a modell réteget, ugyanis nem lehet normálisan megjeleníteni négyzeteket adott koordinátában úgy, ahogyan azt *Windows Forms*ban lehetett – hatékonyan.

Utálom a MAUI-t.

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.

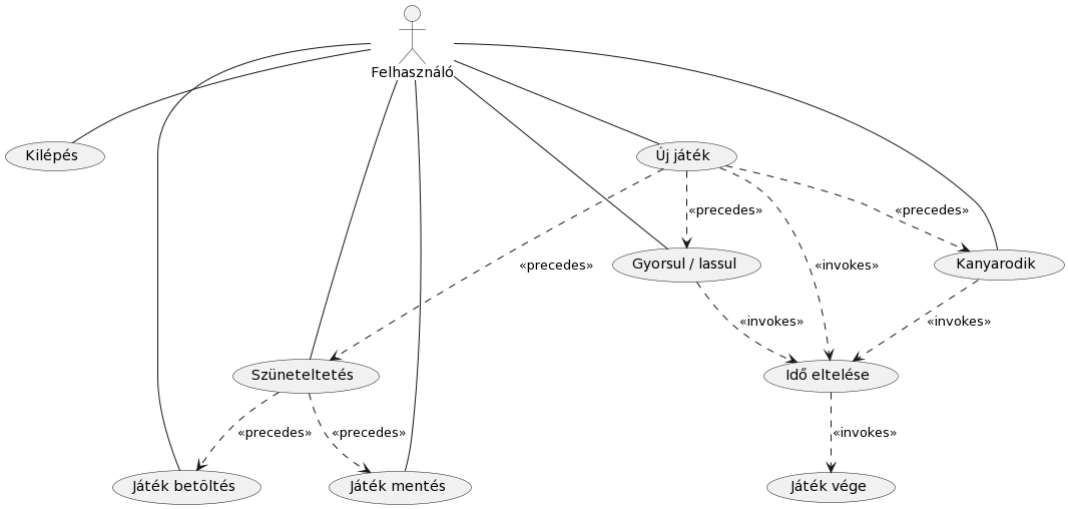
A motorral a képernyő alsó sorában tudunk balra, illetve jobbra navigálni. A képernyő felső sorában meghatározott időközönként véletlenszerű pozícióban jelennek meg üzemanyagcellák, amelyek folyamatosan közelednek a képernyő alja felé. Mivel a motor gyorsul, ezért a cellák egyre gyorsabban fognak közeledni, 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 kék **motorkerékpár** (vagy motoros). Rendelkezik **gyorsaság**gal és **üzemanyagtartály**al. A motoros csak jobbra vagy balra tud mozdulni a képernyőn a megfelelő gombok lenyomásával, valamint felgyorsulni és lelassulni.
- A pályát egy 9×16 méretű mátrix reprezentálja. Ennek az első sorában szabályos időközönként (önkényesen legyen 2 mp-ként) megjelennek ún. piros **üzemanyagcellák**, véletlenszerű x koordinátában. Ezek a motor gyorsaságával egyenes arányosságban haladnak lefelé. Ha a motor pontosan a cella alatt van, 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 *.NET Multi-Platform UI* (MAUI) grafikus felülettel készítjük el.
- Mivel a mobilos eszközök más megjelenítési eszközöket alkalmaznak, mint számítógépen, így négy különböző **oldal**on különítjük el az alkalmazás eseményeit.
  1. Főmenü: innen indítjatunk új játékot (vagy folytathatjuk a már megkezdett vagy betöltött menetet), tölthetünk be korábbi mentést, készíthetünk új mentést. A súgóval egy rövid bemutatkozó dialógusablakot is megnyithatunk
  2. Játék oldala: Felül négy címkel jelöli a rekordidőt, az aktuális időt, a tartáléy szintjét és a sebességet. alul a négy gombbal tudunk navigálni a játákban. A maradék területet meg a 9×16-os mátrix teszi ki.
  3. Betöltés: korábban mentett fájlokat listázza ki.
  4. Mentés: megadjuk a fájlnevet, amivel elmentjük a menetünket.



1.1. ábra. Felhasználói esetek diagramja

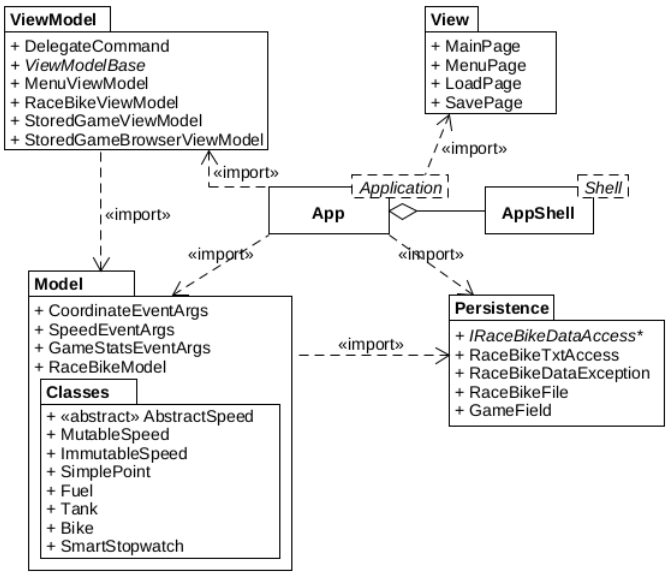
2. fejezet

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.

Az alkalmazás két projektből áll: az `RaceBike.ClassLib` a `Model` és a `Persistence` rétegeket, névtereket tartalmazza, mappákba elkülönítve[1], a második (ez csak simán `RaceBike`) meg a felhasználói felület implementációját foglalja magába, benne a `View` és a `ViewModel` névtérrel. Bevezettünk még a `Store` csomagot is, ami a fájlok betöltésének és mentésének megjelenítéséért felel.



2.1. ábra. Az alkalmazás csomagdiagramja

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

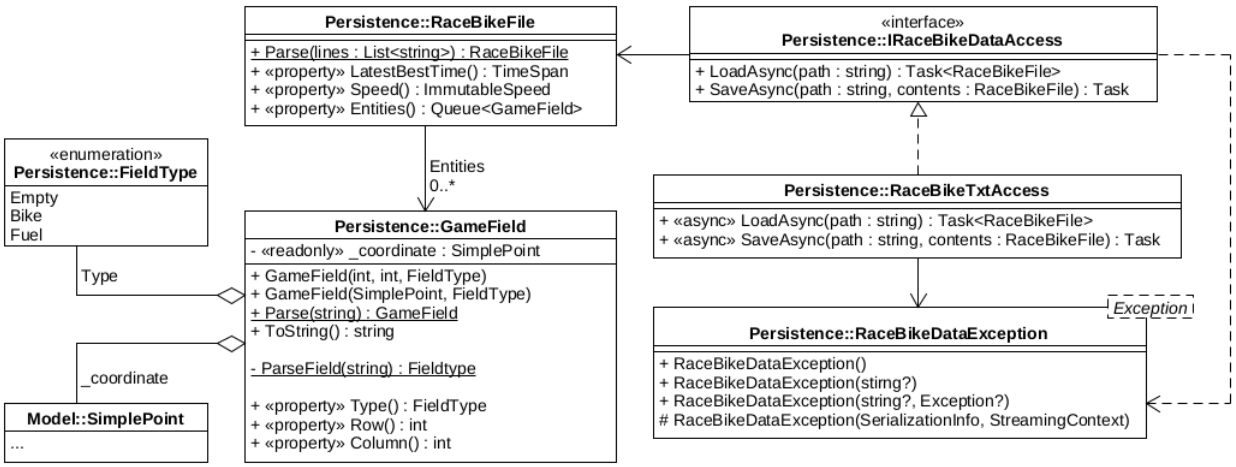
2.1.1. Perzisztencia

- 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 (5,13)
Fuel (3,4)
Fuel (5,6)
```

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.
  - A perzisztenciaréteghez **függőségi befecskendezés**sel jut hozzá a konstruktoron keresztül.
  - A réteg eltárolja egy mátrixban, hogy az adott koordinátákban milyen entitás található. A hatékonyabb keresés érdekében külön eltároljuk, hogy a motoros és az üzemanyagok hol helyezkednek el.
  - A pálya mérete be van égetve konstansokkal, amik lekérdezhetők tulajdonságokon keresztül.
  - Az alábbi események érhetők el: `GameContinues`, `GameOver`, `GameOnPause`, `CoordinateChanged`, `SpeedChanged`. Az első argumentum nélküli, a 2-3.-nak `GameStatsEventArgs` a típusa (ezzel frissítjük a menüben megjelenő szövegeket), a maradék kettőnek meg rendre `CoordinateEventArgs`, `SpeedEventArgs`.
- 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.
- A koordinátákat a `SimplePoint` osztály reprezentálja, amely implementálja az `INotifyPropertyChanged` interfészt – ez elősegíti, hogy adatkötésnél megfigyelhető legyen. Eltárolja az `X` és `Y` koordinátáit, valamint azt, hogy milyen entitás helyeztedik az adott pozícióban.

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.

2.1.4. Nézet

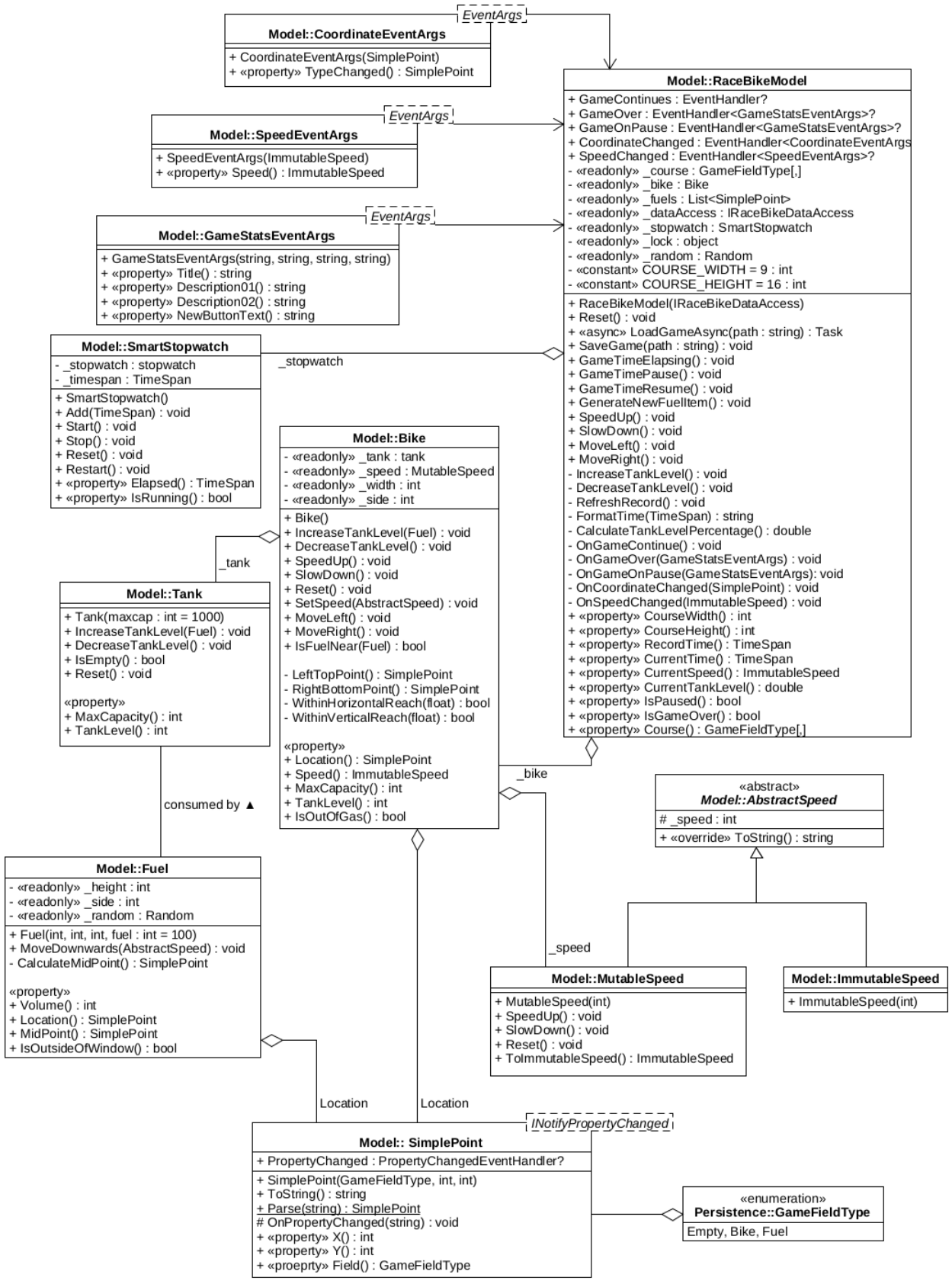
- A nézetet a `MainPage`, a `GamePage`, a `LoadPage` és a `SavePage` lapok alakítják ki.
- A `GamePage` jeleníti meg a játéktáblát egy `Grid` segítségével.
- A `LoadPage` és a `SavePage` szolgál egy létező játékállapot betöltésére, illetve egy új mentésére.

2.1.5. Vezérlés

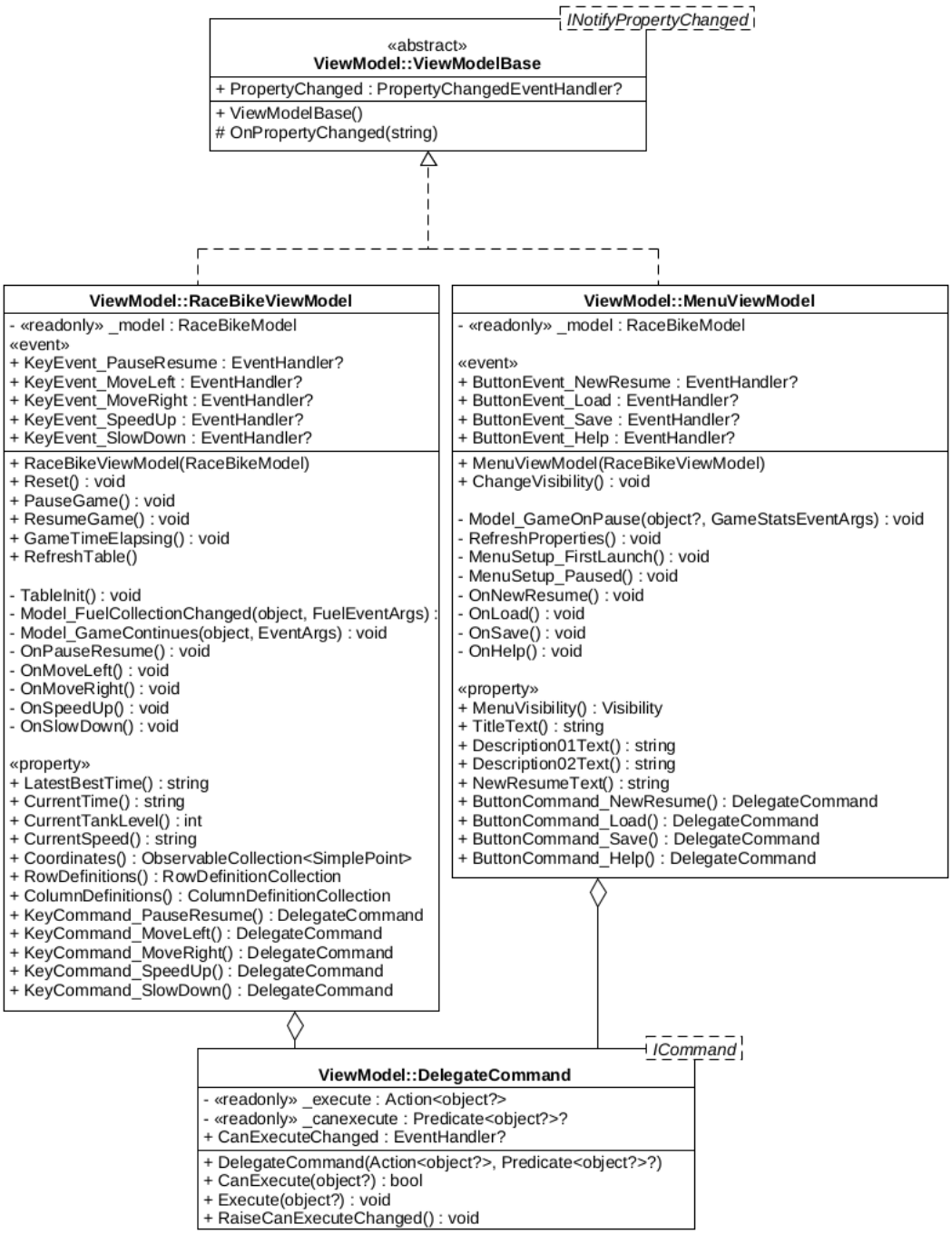
- Az `App` osztály feladata az alkalmazás vezérlése, a rétegek példányosítása és az események feldolgozása.
- A `CreateWindow` metódus felüldefiniálásával kezeljük az alkalmazás életciklusát a megfelelő eseményekre történő feliratkozással. Így az alkalmazás felfüggesztéskor (`Stopped`) elmentjük az aktuális játékállást (`SuspendedGame`), míg folytatáskor vagy újraindításkor (`Activated`) pedig folytatjuk, amennyiben történt mentés.
- Az alkalmazás lapjait egy `AppShell` keretben helyezzük el. Ez az osztály felelős a lapok közötti navigációk megvalósításáért.

2.2. Tesztelés

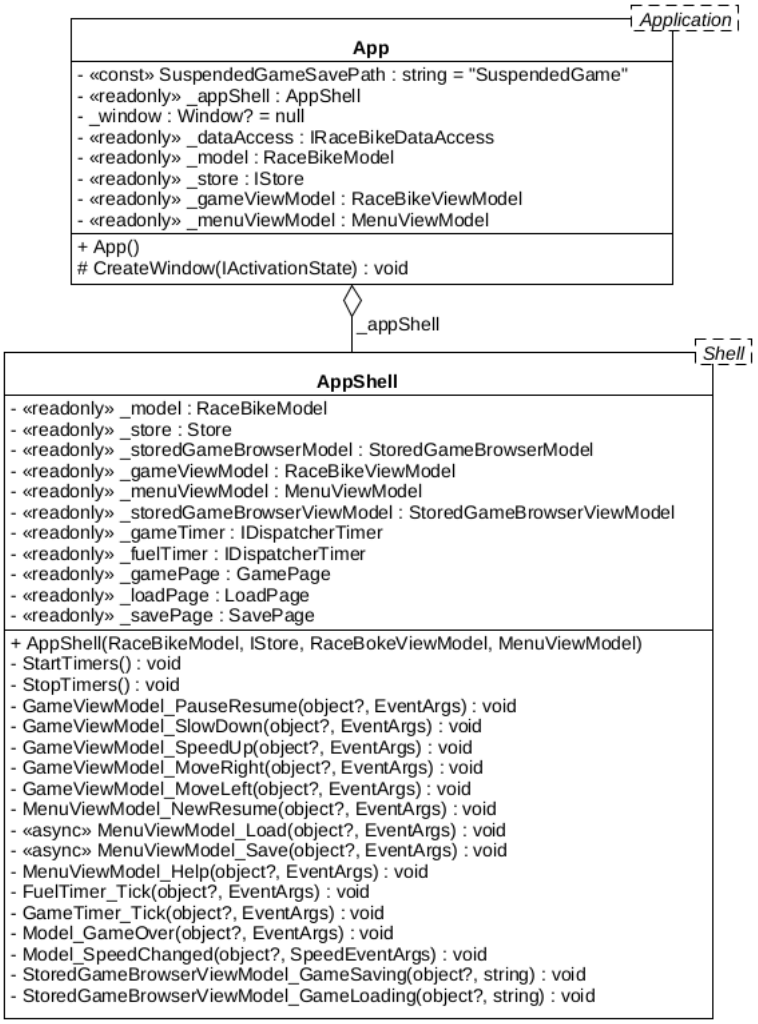
- Az alkalmazás működését egységteszteléssel ellenőriztem, melyeket a `RaceBikeUnitTest` osztályban helyeztem el. A *Moq* csomag felhasználásával oldottam meg a modell és a perzisztencia szimulációját.
  - `Initialize()` – inicializálja a tesztkörnyezetet.
  - `LoadFileContents()` – privát metódus, mely 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.



2.4. ábra. A *Model* csomag osztálydiagramja



2.5. ábra. A *ViewModel* csomag osztálydiagramja



2.6. ábra. A vezérlés osztálydiagramja