3. Beadandó feladat dokumentáció

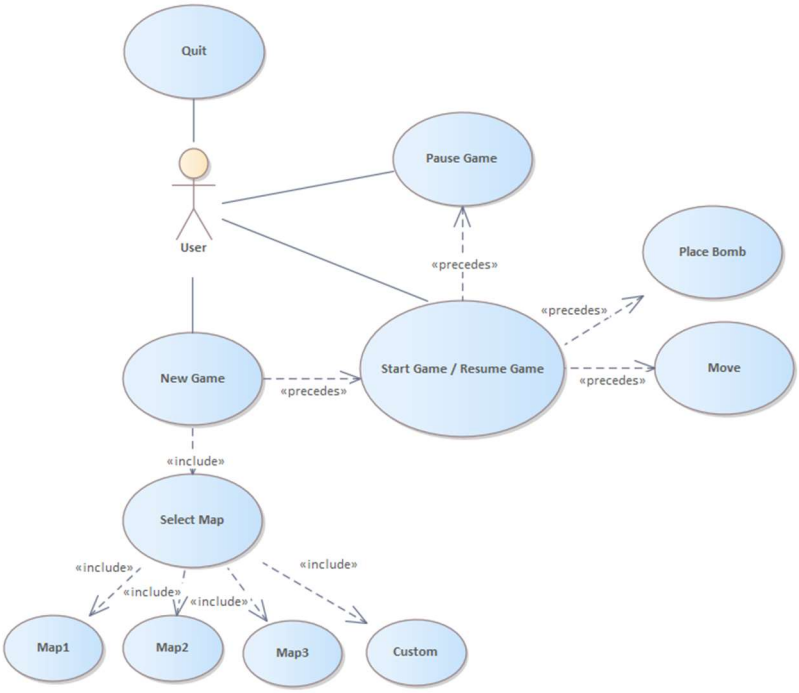
Készítette:
Raczkó Dávid Attila
Email: ycfciy@inf.elte.hu

Feladat:

Készítsünk programot, amellyel a következő játékot játszhatjuk. Adott egy $n \times n$ mezőből álló játékpálya, amelyeken falak, illetve járható mezők helyezkednek el, valamint ellenfelek járőröznek. A játékos célja, hogy ellenfeleit bombák segítségével minél gyorsabban legyőzze. Az ellenfelek adott időközönként lépnek egy mezőt (vízszintesen, vagy függőlegesen) úgy, hogy folyamatosan előre haladnak egészen addig, amíg falba nem ütköznek. Ekkor véletlenszerűen választanak egy új irányt, és arra haladnak tovább. A játékos figurája kezdetben a bal felső sarokban helyezkedik el, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán, de ha találkozik (egy pozíciót foglal el) valamely ellenféllel, akkor meghal. A játékos bombát rakhat le az aktuális pozíciójára, amely rövid időn belül robban megsemmisítve a 3 sugáron belül (azaz egy 7 × 7-es négyzetben) található ellenfeleket (falon át is), illetve magát a játékost is, ha nem menekül onnan. A pályák méretét, illetve felépítését (falak, ellenfelek kezdőpozíciója) tároljuk fájlban. A program legalább 3 különböző méretű pályát tartalmazzon. A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet a játékos). Ismerje fel, ha vége a játéknak, és jelezze, győzött, vagy veszített a játékos. A program játék közben folyamatosan jelezze ki a játékidőt, valamint a felrobbantott ellenfelek számát.

Elemzés:

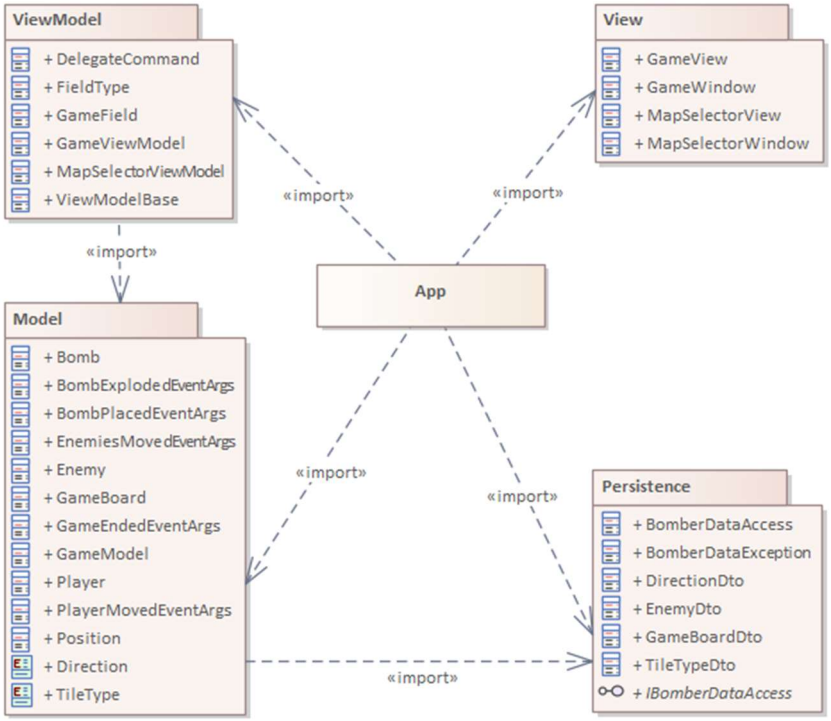
- A feladatot .NET Avalonia alkalmazásként valósítjuk meg, amely két fő nézetből áll: pályaválasztó és játék. Az alkalmazás elsődlegesen asztali platformon és Androidon érhető el, és portré tájolást támogat.
- A pályaválasztó ablakon lehetőség van 3 előre definiált pálya közül választani, vagy egyedi pályát betölteni fileból.
- A Játék ablakon megjelenítjük az eltelt időt, a játékos pontszámát, valamit 3 gombot, amelyek lehetőséget adnak a játékosnak
    - új játék kezdésére: Ilyenkor megjelenik a pályaválasztó és eltűnik a játékablak
    - játék elindítására/újraindítására: Elkezd telni az idő, elkezdenek mozogni az ellenségek és a felhasználónak is lehetősége van a játékost mozgatni.
    - játék megállítására
- A játéktáblát egy nxn-es (pályamérettől függő) képekből álló rács reprezentálja. Minden mező háttérképe a játék állapotától függően változik. Egy bomba felrobbanásakor a robbanás közepétől mért 3 sugarú négyzetben a mezők háttérképe a robbanás képére változik, jelezve a bomba robbanását.
- Amikor egy ellenség meghal, a játékos pontot szerez, ami meg is jelenik a képernyőn. Amikor a játéknak vége (meghal az összes ellenség / az ellenségek elkapják a játékost), egy dialógusablak jelenik meg, ami tájékoztat a játék kimeneteléről.
- A felhasználói esetek az 1. ábrán láthatóak.



1. ábra: Felhasználói esetek diagramja

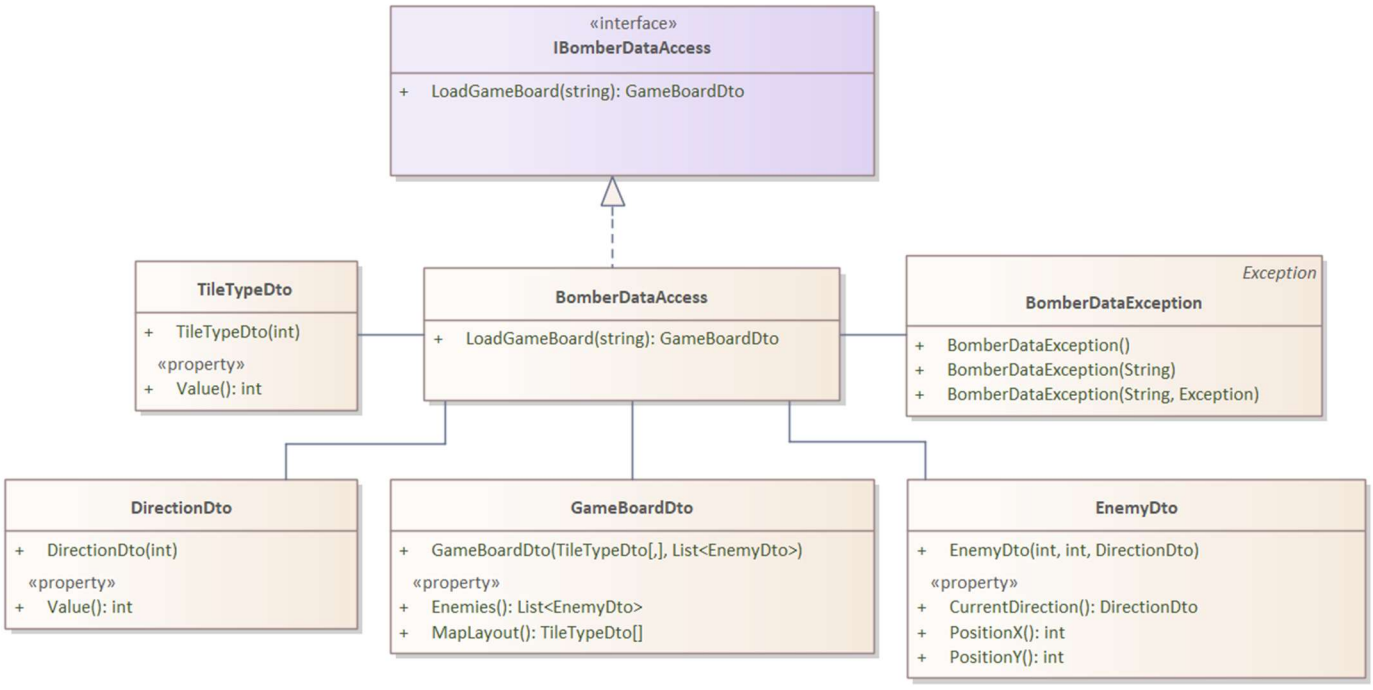
Tervezés:

- **Programszerkezet:**
  - A szoftvert négy projektből építjük fel: a modellt és a perzisztenciát tartalmazó osztálykönyvtárból (.NET Standard Class Library), valamint a .NET Avalonia projektjeiből (platformfüggetlen osztálykönyvtár és platformfüggő végrehajtható projektek), amelyet így Windows és Android operációs rendszerekre is le tudunk fordítani.A programot MVVM architektúrában valósítjuk meg, ennek megfelelően View, Model, ViewModel és Persistence névtereket valósítunk meg az alkalmazáson belül.



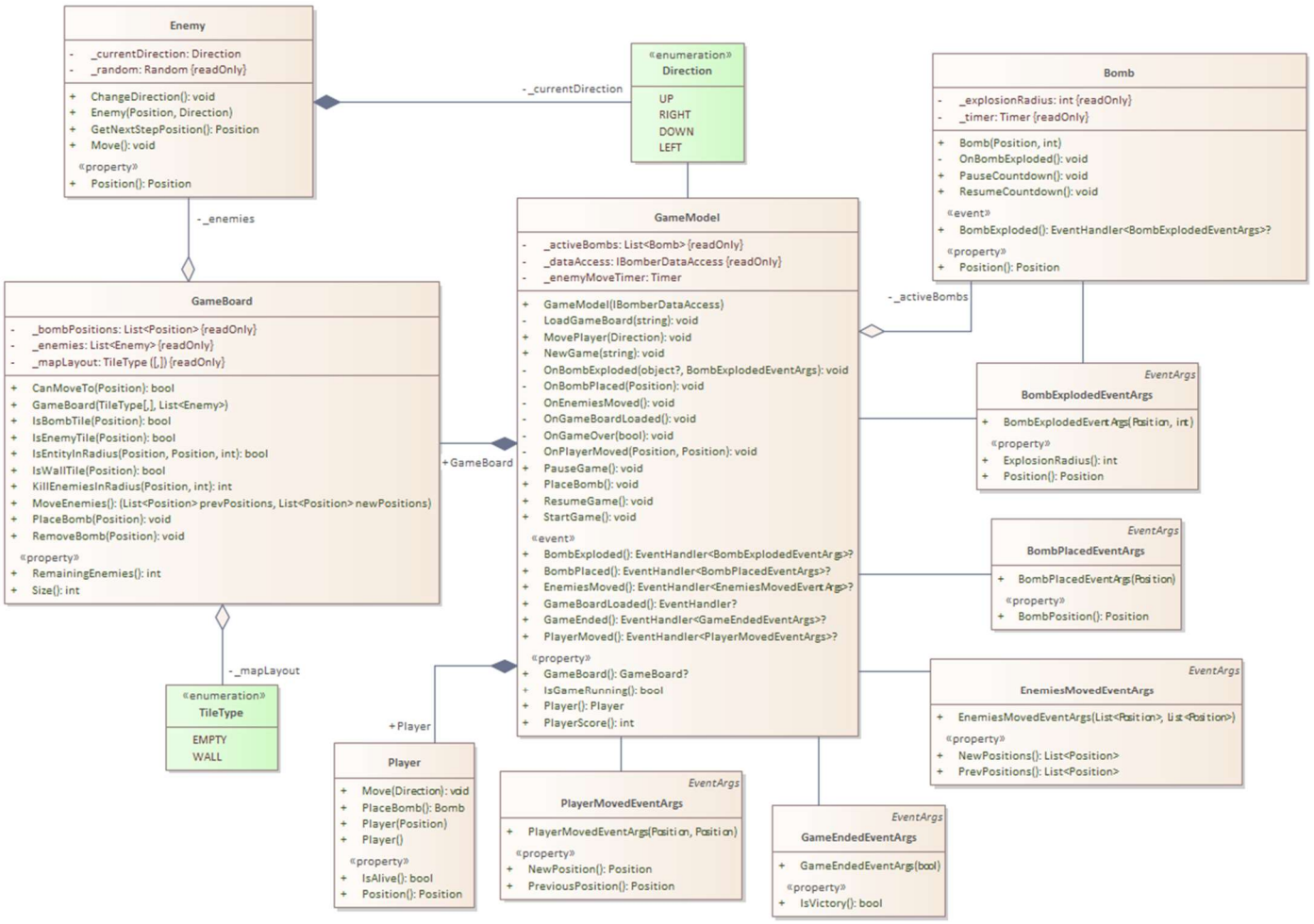
2. ábra: Az alkalmazás csomagdiagrammja

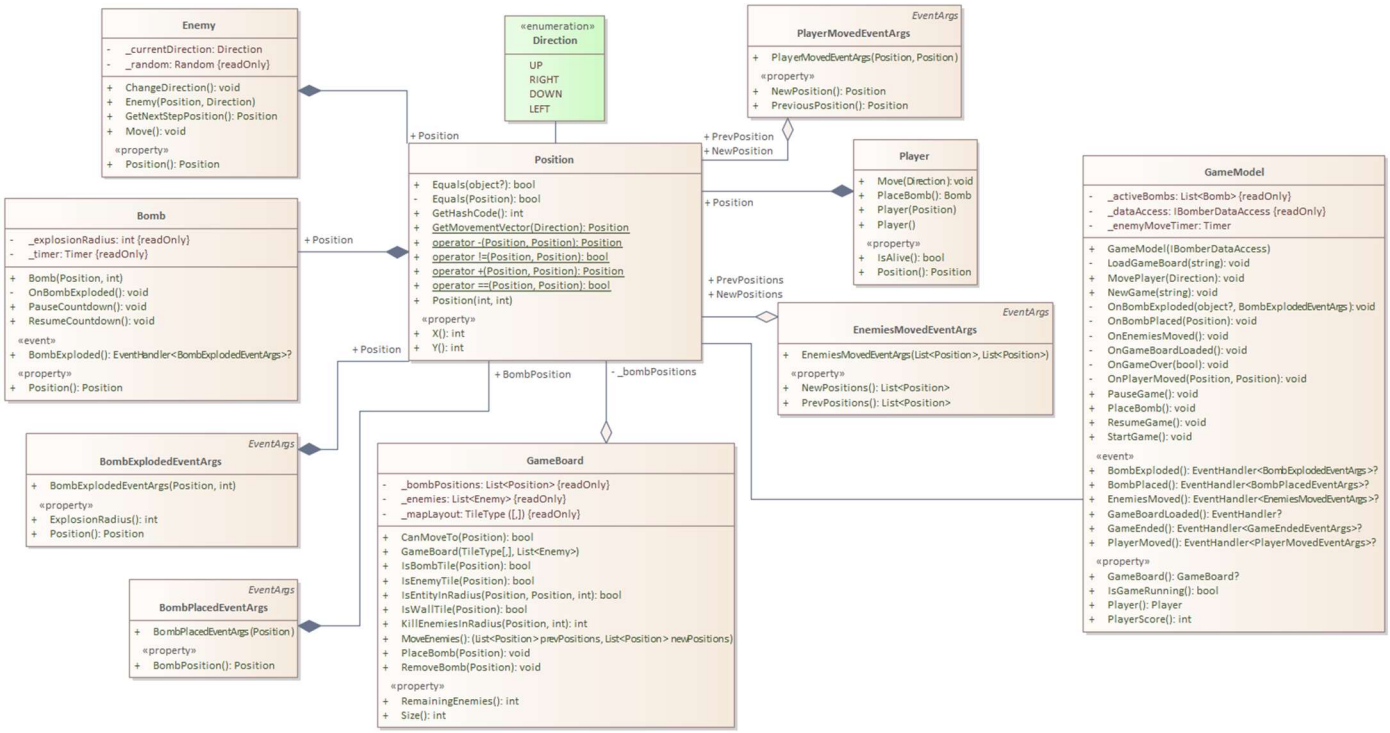
- **Perzisztencia**
  - Az adatkezelés feladata a Játéktáblával kapcsolatos információk betöltése fileból (előre definiált / egyedi pályák).
  - A perzisztencia rétegben úgynevezett data transfer objectek (DTO-k) találhatók, amik biztosítják az adatáramlást a két réteg között anélkül, hogy a perzisztencia réteg hozzáférne a modellben található implementációs részletekhez.
  - A hosszú távú adattárolás lehetőségeit az IBomberDataAccess interfész adja meg, amely lehetőséget ad a tábla betöltésére (LoadGameBoard).
  - Az interfészt szöveges fájl alapú adatkezelésre a BomberDataAccess osztály valósítja meg. A fájlkezelés során fellépő hibákat a BomberDataException kivétel jelzi.
  - A fájl első sora megadja a tábla méretét (n) és az ellenségek számát (e). Az ezek után következő n sorban a pálya elrendezése látható, ahol „.” -tal vannak jelölve az üres mezők, a falak pedig „#” jellel. Az ezt követő e-sorban az ellenfelek pozícióinak és kezdő irányainak megadása valósul meg.



3. ábra: Perzisztencia csomag osztálydiagrammja

- **Modell**
  - A játékmodell lényegi részét BomberModel és a GameBoard osztályok alkotják. A GameBoard osztály tárolja a pálya adatait (elrendezés, bombák pozíciói, ellenségek) és lekérdező műveleteket biztosít. A GameModel osztály valósítja meg a játék logikáját (játékos mozgatása, bomba lerakása) és felelős az események kiváltásáért, kezeléséért.
  - Események:
    - PlayerMoved: a játékos lépéséről értesíti a nézetet, átadva a játékos előző és aktuális pozícióját.
    - EnemiesMoved: az ellenségek mozgásának hatására váltódik ki, az ellenségek korábbi és jelenlegi pozícióját adja át.
    - A BombPlaced: Kiváltódik, ha a játékos lerak egy bombát, a lerakott bomba pozíciója továbbítódik a nézet felé.
    - BombExploded: Kiváltódik, amikor egy bomba felrobban, és továbbítja a bomba helyzetét, valamint a robbanás sugarát.
    - GameEnded: Jelzi a játék végét, és átadja a kimenetelt (győzelem vagy vereség).
    - GameBoardLoaded: Akkor váltódik ki, amikor a játéktábla sikeresen betöltődött, jelezve, hogy megjeleníthető.
    - Az események száma hatékonysági okokból ilyen sok, hiszen lehetne egyesíteni őket egy UpdateUI eseménnyel is, aminek hatására a nézet újrarajzolná az egész játéktáblát. De ez helyett inkább csak azok a mezők kerülnek újrarajzolásra, amik megváltoztak, ezeket pedig az események továbbítják a nézet felé.

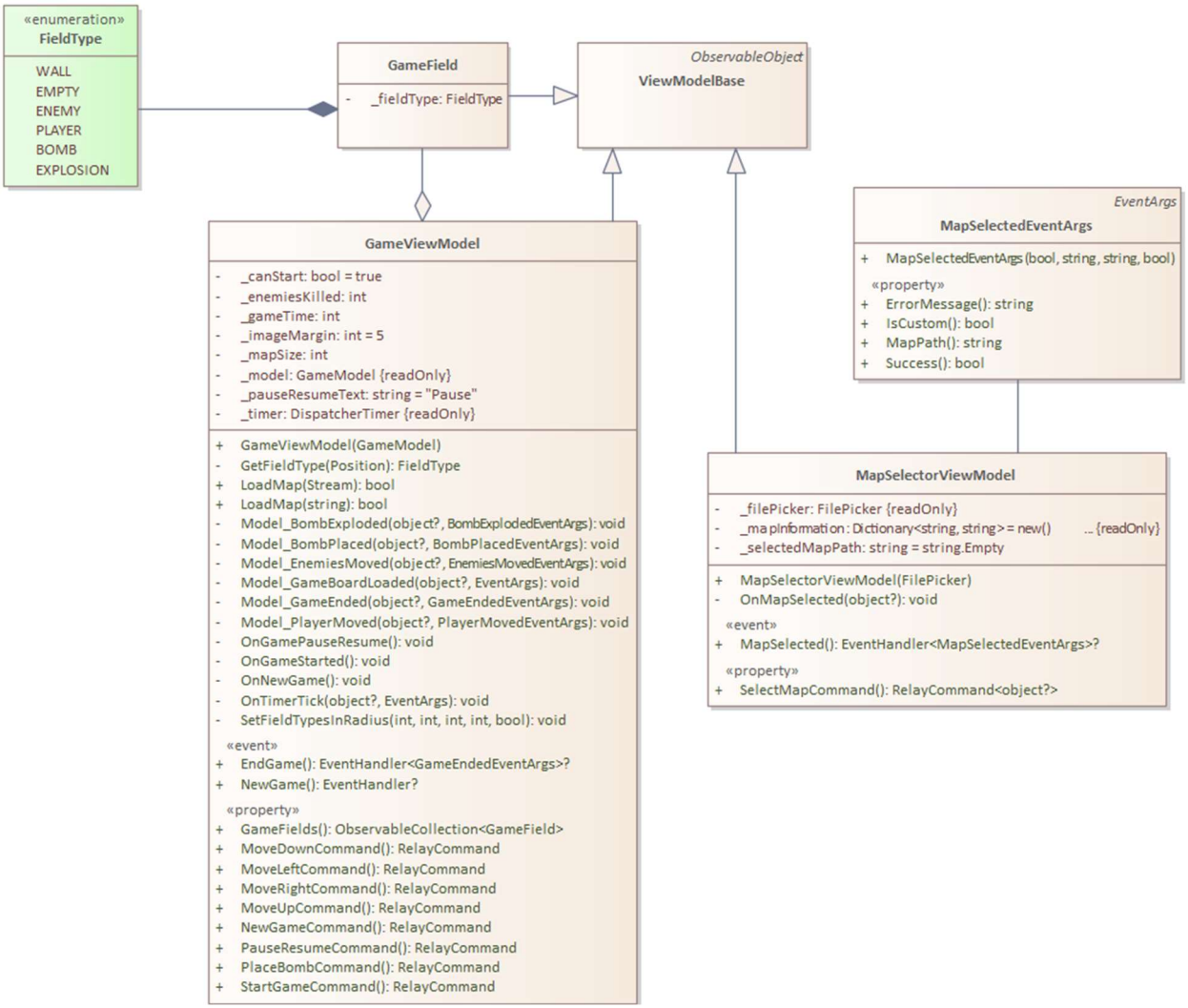




4-5. ábra: Model csomag osztálydiagrammja

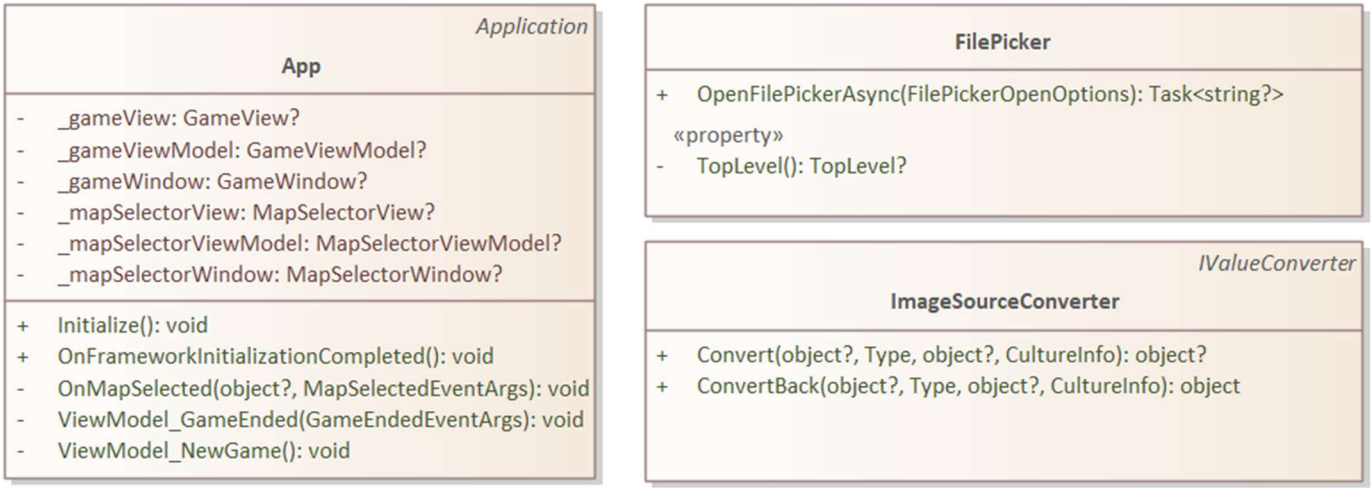
- **NézetModell - MapSelectorViewModel**
  - A nézetmodell megvalósításához felhasználjuk az MVVM Toolkit csomagból elérhető általános utasítás (RelayCommand), valamint egy ős változásjelző (ObservableObject) osztályt.
  - A nézetmodell feladatait a MapSelectorViewModel látja el, amely lehetőséget ad a pálya kiválasztására (könnyű, közepes, nehéz, egyedi). A parancshoz (MapSelectCommand) eseményt kötünk, így biztosítjuk, hogy a megfelelően kiválasztott pálya után megnyíljon a játékablak, benne a megfelelően példányosított GameModel osztállyal.

- **NézetModell – GameViewModel**
  - A nézetmodell megvalósításához felhasználjuk az MVVM Toolkit csomagból elérhető általános utasítás (RelayCommand), valamint egy ős változásjelző (ObservableObject) osztályt.
  - A nézetmodell feladatait a GameViewModel látja el, amely parancsokat biztosít új játék kezdéséhez, a játék elindításához, szüneteltetéséhez és újraindításához. A parancsokhoz eseményeket kötünk, amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja a modell egy hivatkozását, de csupán információkat kér tőle, a játék futásába nem avatkozik bele. A játékmező számára egy külön mezőt biztosítunk (GameField), amely eltárolja azt, hogy az adott helyen milyen képnek kell szerepelnie (FieldType enumként). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellben (GameFields).



6. ábra: NézetModellek osztálydiagrammjai

- **Nézet**
  - A nézet különböző platformokon (telefonos és asztali) történő megjelenítéshez készült, és két fő elemet tartalmaz: a *MapSelectorView*-t és a *GameView*-t. Ezeket a *MapSelectorWindow* és a *GameWindow* ablakok jelenítik meg.
  - MapSelectorView
    - A megfelelő megjelenést Grid elrendezéssel valósítjuk meg. A pályaválasztási lehetőségek képként jelennek meg, amelyekhez adatkötéseken keresztül parancsokat társítunk. Ezek biztosítják a pályaválasztási folyamat megfelelő működését. Ha a kiválasztott pálya nem megfelelő, figyelmeztető üzenetet jelenítünk meg. Sikeres választás esetén a nézet automatikusan a játéknézetre (*GameView*) vált.
  - GameView
    - A játékmező és az információs panelek egy rácsban helyezkednek el. A játékmezőt egy *ItemsControl* vezérlő valósítja meg, amelyben dinamikusan építünk fel egy képekből álló *UniformGrid* rácsot. Az összes adatot adatkötésekkel kapcsoljuk a felülethez, és ezen keresztül szabályozzuk a képek hátterét is. A játék befejezésekor egy felugró ablak tájékoztat a játék kimeneteléről.
  - MapSelectorWindow
    - MapSelectorView asztali megjelenítéséért felel.
  - GameWindow
    - GameView asztali megjelenítéséért felel.
- **Vezérlés**
  - Az App osztály az alkalmazás vezérléséért, a rétegek példányosításáért és az események feldolgozásáért felel.
  - **OnFrameworkInitializationCompleted**: A nézetek platformspecifikus megjelenítését kezeli:
    - Asztali platformokon a MapSelectorWindow és GameWindow ablakokat használja.
    - Mobil platformokon a MainView tulajdonságon keresztül jeleníti meg a MapSelectorView és GameView nézeteket.
  - **Nézetváltások**:
    - A OnMapSelected kezeli a pályaválasztást, és betölti a megfelelő nézetet.
    - A játék végeztével a ViewModel_GameEnded értesít, és visszavált a pályaválasztóra.
  - Új játék indítását a ViewModel_NewGame végzi.



7. ábra: A vezérlés osztálydiagrammja

Tesztelés:

- A teszteket a GameModelTest osztályban hajtjuk végre, mockolt adatkezelővel (Moq), hogy egy előre beállított játéktáblán futtathassuk a teszteket. Az alábbiakban részletezzük a tesztelési folyamatokat:
- TestNewGame:
  - Ebben a tesztben ellenőrizzük, hogy egy új játék indításakor helyesen inicializálódik-e a játéktábla.
    - Megfelelő a pálya mérete?
    - Az ellenségek helyes pozíción lettek inicializálva?
    - Megfelelő a pálya elrendezése?
    - Megfelelő számú ellenség került le a pályára?
  - Emellett ellenőrizzük a játékos kezdőpozícióját (0, 0), azt, hogy él-e, és a játék futási státuszát (a játék eleinte nem fut).
- TestPauseResumeStart:
  - Ebben a tesztben azt ellenőrizzük, hogyan viselkedik a játék szüneteltetése és újraindítása.
  - A játék elindítása előtt a szünet és folytatás funkcióknak nincs hatása, a játék futása nem indul meg.
  - Miután elindítjuk a játékot, a szüneteltetés leállítja azt, és a folytatás újraindítja.
- TestPlayerMovement:
  - A játékos mozgását teszteljük, ellenőrizve, hogy a játék indulása előtt a játékos nem tud mozogni.
  - Miután elindítjuk a játékot, megvizsgáljuk, hogy a játékos mozgásai helyesek-e: nem léphet ki a pálya szélén, és falakon sem tud áthaladni. Pl. a fal előtt a játékos megáll, és nem tud továbbmenni.
- TestBombPlacement:
  - Ebben a tesztben a bomba elhelyezését vizsgáljuk. Kezdetben, mivel a játék nem fut, nem tudunk bombát lerakni.
  - A játék elindítása után a bomba sikeresen lehelyezhető, és eseményt vált ki, amelyet a tesztben ellenőrzünk. A bombát a játéktábla megfelelő pozíciójára kerül lehelyezésre.
- TestBombExplosion:
  - A bomba robbanását teszteljük. Miután a játékos lerak egy bombát, a robbanás időzítése aktiválódik, és néhány másodpercen belül felrobban.
  - A robbanás eseményt vált ki, amelyet szintén ellenőrzünk. A robbanás után a bombának el kell tűnnie a játéktábláról, és a robbanásnak el kell pusztítania a közelben lévő ellenségeket.
- TestEnemyMovement:
  - Az ellenségek mozgását vizsgáljuk. Elindítjuk a játékot, és várunk, hogy az ellenségek mozgásba lendüljenek.
  - Ellenőrizzük, hogy az ellenségek valóban elmozdulnak-e az előre meghatározott irányukba, és eseményt váltanak-e ki, amely a mozgásukra vonatkozik.