

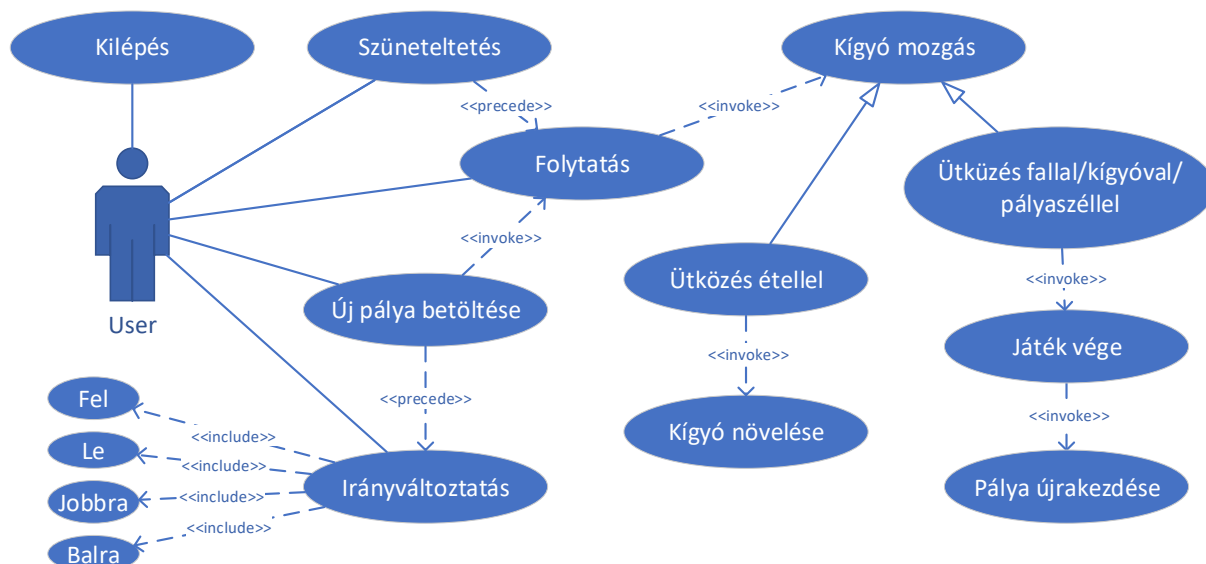
## Feladatleírás

### Snake

Készítsük programot, amellyel a klasszikus kígyó játékot játszhatjuk. Adott egy  $n \times n$  elemből álló játékpálya, amelyben akadályok (falak) találhatóak. A játékos egy kezdetben 5 hosszú kígyóval indul a képernyő közepén, amely vízszintesen, illetve függőlegesen halad rögzített időközönként a legutoljára beállított irányba. A kígyóval elfordulhatunk balra, illetve jobbra. A pályán véletlenszerű pozícióban mindig megjelenik egy tojás, amelyet a kígyóval meg kell etetni. Minden etetéssel eggyel nagyobb lesz a kígyó. A játék célja, hogy a kígyó minél tovább elkerülje az ütközést az akadályokkal, a pálya szélével, illetve saját magával. A pályák méretét, illetve felépítését (falak helyzete) 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 mozog a kígyó). Továbbá ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hány tojást sikerült elfogyasztania a játékosnak.

### Elemzés:

- A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.
- A játék alapja egy rácselrendezés, melyen  $n \times n$  db panel helyezkedik el. A panelek színük változtatásával jelzik a játék aktuális állását.
- Az ablak tetején elhelyezett menüpontokkal betölthető új pálya, szüneteltethető a játék, valamint kilépésre is van mód.
- Az ablak mérete automatikusan alkalmazkodik a pálya méretéhez.
- A kígyó egyenletes időközönként a felhasználó által beállított irányba mozog. Étellel való találkozás esetén mérete eggyel nő, azonban, ha ütközik a pálya szélével, önmagával, vagy egy fallal, a játék véget ér.
- Amennyiben a játék véget ér, előugró ablakkal jelezzük az elért pontszámot. Egy új pálya betöltésére szintén felugró dialógusablakkal van mód.
- A felhasználói esetek az 1. ábrán találhatóak.



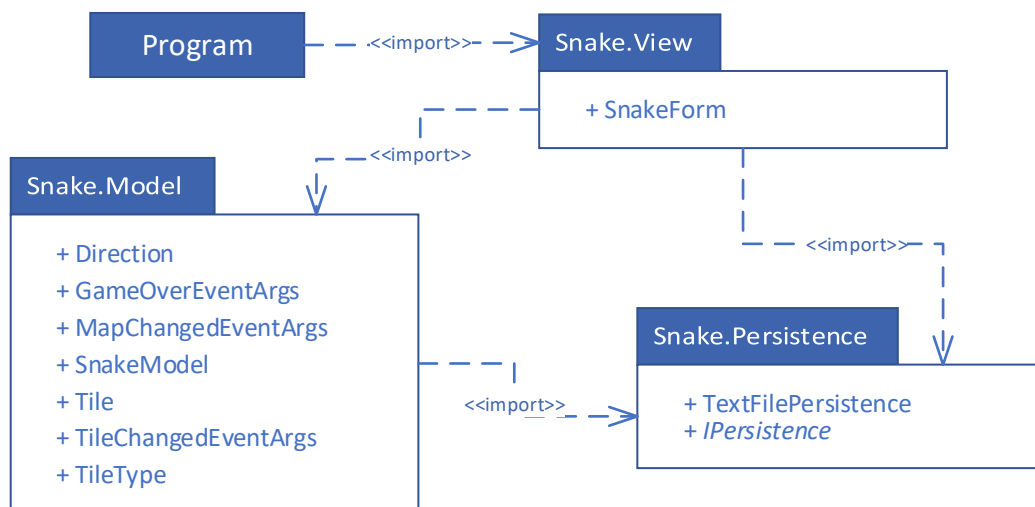
1. ábra: A felhasználói esetek diagramja

## Tervezés:

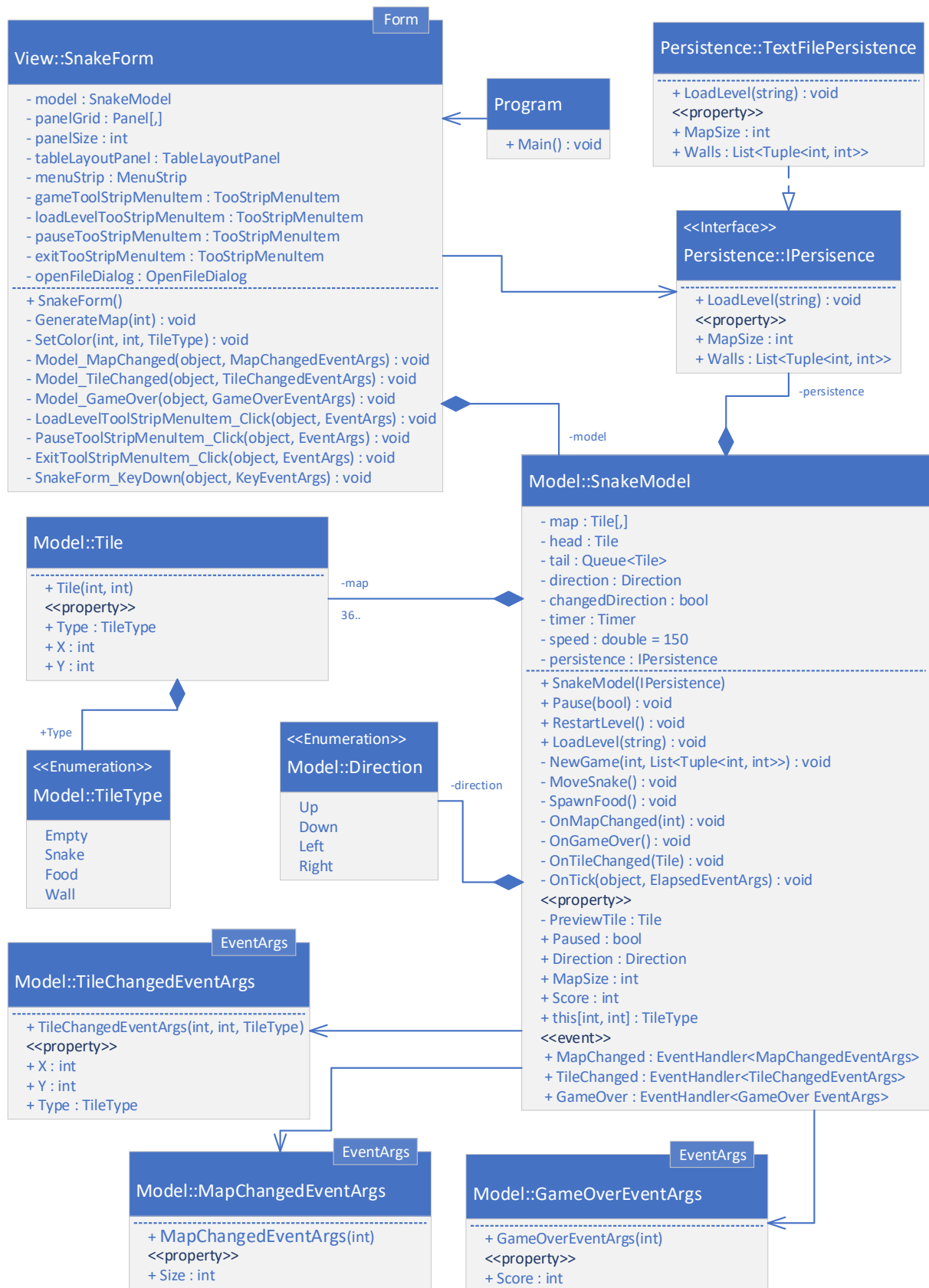
- **Programszerkezet:**
  - A programot háromrétegű architektúrában valósítjuk meg. A megjelenítés a **Snake.View**, a modell a **Snake.Model**, míg a perzisztencia a **Snake.Persistence** névtérben helyezkedik el. A program csomagszerkezete a 2. ábrán látható.
- **Perzisztencia**
  - Az adatelérés feladata az egyes pályákkal kapcsolatos információk tárolása, valamint azok betöltése.
  - Ezen feladatok végrehajtásának lehetőségét az **IPersistence** interfész adja meg: lekérdezhető a pálya mérete (**MapSize**), a falak pozíciója (**walls**), valamint új pálya betöltésére is van mód (**LoadLevel**)
  - Az előző pontban említett interfészt a **TextFilePersistence** osztály szöveges alapú adatkezelésre valósítja meg. Ez a fájlok olvasása közbeni előforduló hibákat **DataException** kivétellel jelzi.
  - Az egyes pályák szöveges fájlokban vannak tárolva, a játék futása során bármikor betölthetők.
  - A fájlokban az első szám adja meg a pálya méretét, majd ezután párosával jönnek a falak koordinátái.
- **Modell**
  - A modell lényegi részét a **SnakeModel** osztály valósítja meg, mely tárolja a pályát alkotó mezőket (**map**), a kígyót (**head**, **tail**), annak irányát (**direction**), és egy időzítőt (**timer**). Számon tartja, hogy szünetel-e a játék (**Paused**), lekérdezhető a pálya mérete (**MapSize**), a játékos eddig elért pontszáma (**Score**), valamint az egyes mezők típusa. Képes új játék indítására (**NewGame**), egy meglévő újakezdésére (**RestartLevel**), a kígyó mozgatására (**MoveSnake**), valamint egy új étel letételére (**SpawnFood**).
  - A nézettel eseményeken keresztül kommunikál. A pálya változásáról a **MapChanged** (**MapChangedEventArgs** argumentummal), egy mező változásáról a **TileChanged** (**TileChangedEventArgs** argumentummal), a játék végéről pedig a **GameOver**

esemény (`GameOverEventArgs` argumentummal) tájékoztat. Az argumentumok mindig az eseményekhez kapcsolódó lényegi információkat tartalmazzák.

- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad pályák betöltésére (`LoadLevel`).
  - A pálya egyes mezőit egy `Tile` típusú objektum reprezentálja, melyek tárolják a mező koordinátáit és típusát.
  - A kígyó irányát a `Direction`, a mezők típusait pedig a `TileType` felsorolási típusokkal kezeljük.
- **Nézet**
    - A nézetet a `SnakeForm` osztály biztosítja, mely tárolja a modell aktuális példányát (`model`).
    - A pályát egy dinamikusan létrehozott (`GenerateMap`), panelekből álló tábla-elrendezés reprezentálja, melyek színe egyenként változtatható (`SetColor`).
    - A példányosításkor létrehozunk a `model` egy új példányát, és feliratkozunk annak eseményeire.
    - A felületen elhelyeztünk egy menüsort a megfelelő elemekkel és a hozzájuk tartozó eseménykezelőkkel.
    - Egy pálya betöltéséhez egy felugró dialógusablakot használunk.
    - A billentyű-leütések eseményeit a `SnakeForm_KeyDown` kezeli. A kígyó mozgatására a kurzormozgató nyilakkal, valamint a WASD billentyűkkel van lehetőség.



2. ábra: A program csomagszerkezete



3. ábra: A program osztálydiagramja

## Tesztelés:

- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a **SnakeModelTest** osztályban.
- A perzisztenciát használó funkciók a **Mock**, a privát metódusok pedig a **PrivateObject** osztály felhasználásával lettek tesztelve.
- Az alábbi tesztesetek kerültek megvalósításra:
  - **SnakeLoadLevelTest**: Egy pálya betöltése után a pálya mérete megfelelő, a falak és a kígyó a helyes pozíciókban és irányban jelennek meg, az időzítő fut, a pontszám pedig 0.
  - **SnakeRestartLevelTest**: Az előző tesztesettel megegyező feltételeket vizsgál.
  - **SnakeMoveTest**: A kígyó a megfelelő irányba, teljes testével mozog.
  - **SnakeSpawnFoodTest**: Helyesen megjelenik egy ételt tartalmazó mező.
  - **SnakeDirectionTest**: A kígyó önmagába fordulni nem tud.
  - **SnakeGameOverTest**: Ütközés esetén a játék helyesen véget ér, kiváltódik a megfelelő esemény.