1. Beadandó feladat dokumentáció

# Készítette:

Horváth Tamás

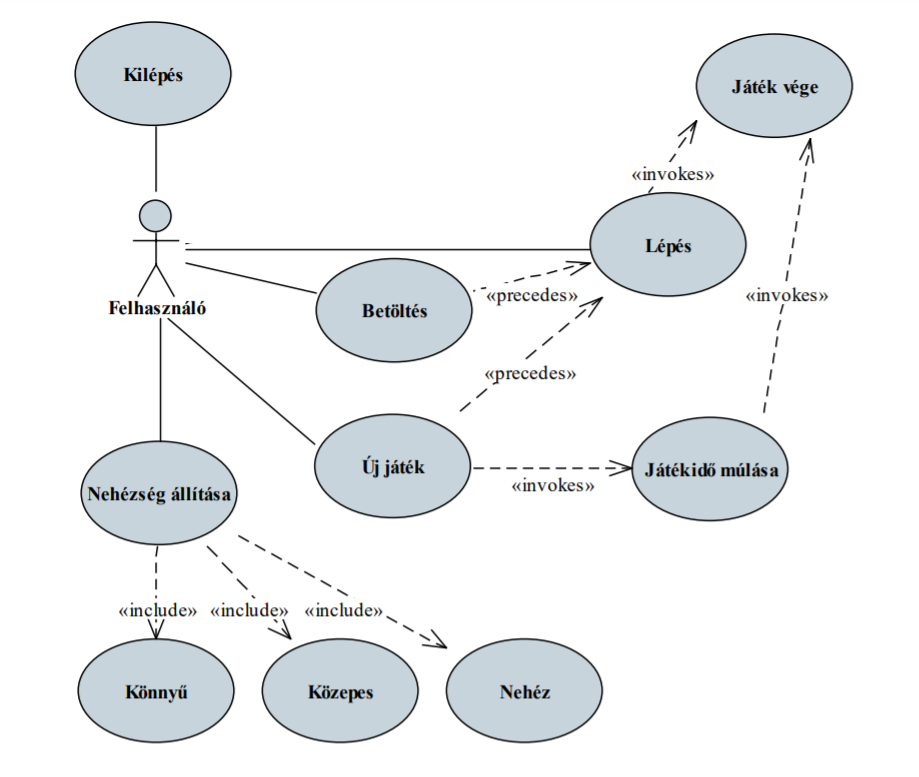
E-mail: [dly99j@inf.elte.hu](mailto:dly99j@inf.elte.hu)

# Feladat:

Készítsünk programot, amellyel a következő játékot játszhatjuk. Adott egy 𝑛 × 𝑛 elemből álló játékpálya, amelyben Maci Lacival kell piknikkosarakra vadásznunk. A játékpályán az egyszerű mezők mellett elhelyezkednek akadályok (pl. fa), valamint piknikkosarak. A játék célja, hogy a piknikkosarakat minél gyorsabban begyűjtsük. Az erdőben vadőrök is járőröznek, akik adott időközönként lépnek egy mezőt (vízszintesen, vagy függőlegesen). A járőrözés során egy megadott irányba haladnak egészen addig, amíg akadályba (vagy az erdő szélébe) nem ütköznek, ekkor megfordulnak, és visszafelé haladnak (tehát folyamatosan egy vonalban járőröznek). A vadőr járőrözés közben a vele szomszédos mezőket látja (átlósan is, azaz egy 3 × 3-as négyzetet). A játékos kezdetben a bal felső sarokban helyezkedik el, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán, a piknikkosárra való rálépéssel pedig felveheti azt. Ha Maci Lacit meglátja valamelyik vadőr, akkor a játékos veszít. A pályák méretét, illetve felépítését (piknikkosarak, akadályok, vadőrök 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 megszerzett piknikkosarak számát.

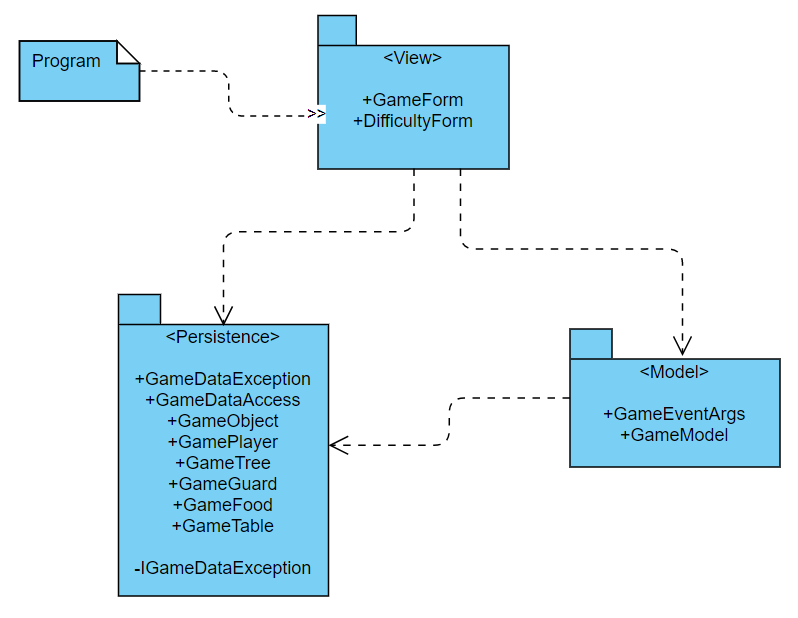
# Elemzés:

* A játékot három nehézségi szinttel játszhatjuk: könnyű (10 x 10-es pálya) közepes (15 x 15 pálya), nehéz (19 x 19 pálya). A program indításkor könnyű nehézséget állít be, és automatikusan új játékot indít.
* A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.
* Az ablakban elhelyezünk három gombot: egy a nehézség választásáért felelőset, egyet ami a játék szüneteltetéséért szolgál, és egyet ami a játék kezdésééer. Emellett két címke látható még, az egyik az eltelt időt, a másik a megszerzett piknikkosarak számát jelzi.
* A játéktáblát egy m x n-es nyomógombokból álló rács reprezentálja. A zöld mező a játékos /maci laci/, a szürke az üres mező, a kék az őr , a piros a piknikkosár és a barna a fa, azaz akadály
* A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (kiraktuk a táblát, vagy letelt az idő). Szintén dialógusablakokkal választuk ki a nehézséget.
* A felhasználói esetek az 1. ábrán láthatóak.



1. **ábra: Felhasználói esetek diagramja**

# Tervezés:

* Programszerkezet:
  + A programot háromrétegű architektúrában valósíjuk meg. A megjelenítés a **View**, a modell a **Model**, míg a perzisztencia a **Persistence** névtérben helyezkedik el. A program csomagszerkezete a 2. ábrán látható.
* Perzisztencia:
  + Az adatkezelés feladata a játék táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
  + A **GameTable** osztály egy érvényes GameTable táblát biztosít (azaz mindig ellenőrzi a beállított értékek), ahol minden mezőre ismert az értéke egy karaktermátrixban, emellett az őrök, ételek, fák külön objektumlistában, a játékos meg objektumban van tárolva. Emellett még el vannak tárolva a tábla dimenziói, az őrök és a piknikkosarak száma
  + A játékobjektumok egy **GameObject** Ősosztályból származnak, mindnek el van tárolva a tartózkodási helye.
  + A hosszú távú adattárolás lehetőségeit az **IGameDataAccess** interfész adja meg, amely lehetőséget ad a tábla betöltésére (LoadAsync), valamint mentésére (SaveAsync). A műveleteket hatékonysági okokból aszinkron módon valósítjuk meg.
  + Az interfészt szöveges fájl alapú adatkezelésre a **GameDataAccess** osztály valósítja meg. A fájlkezelés során fellépő hibákat a **GameDataException** kivétel jelzi.
  + A program az adatokat szöveges fájlként tudja eltárolni, melyek a txtkiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni.
  + A fájl első sora megadja a tábla méretét, A fájl többi része izomorf leképezése a játéktáblának, azaz összesen m sor következik, és minden sor n számot tartalma. A betűk ’E’ = üres, ’P’ = játékos, ’F’ = piknikkosár, ’T’ = fa és ’G’ = őr lehetnek.

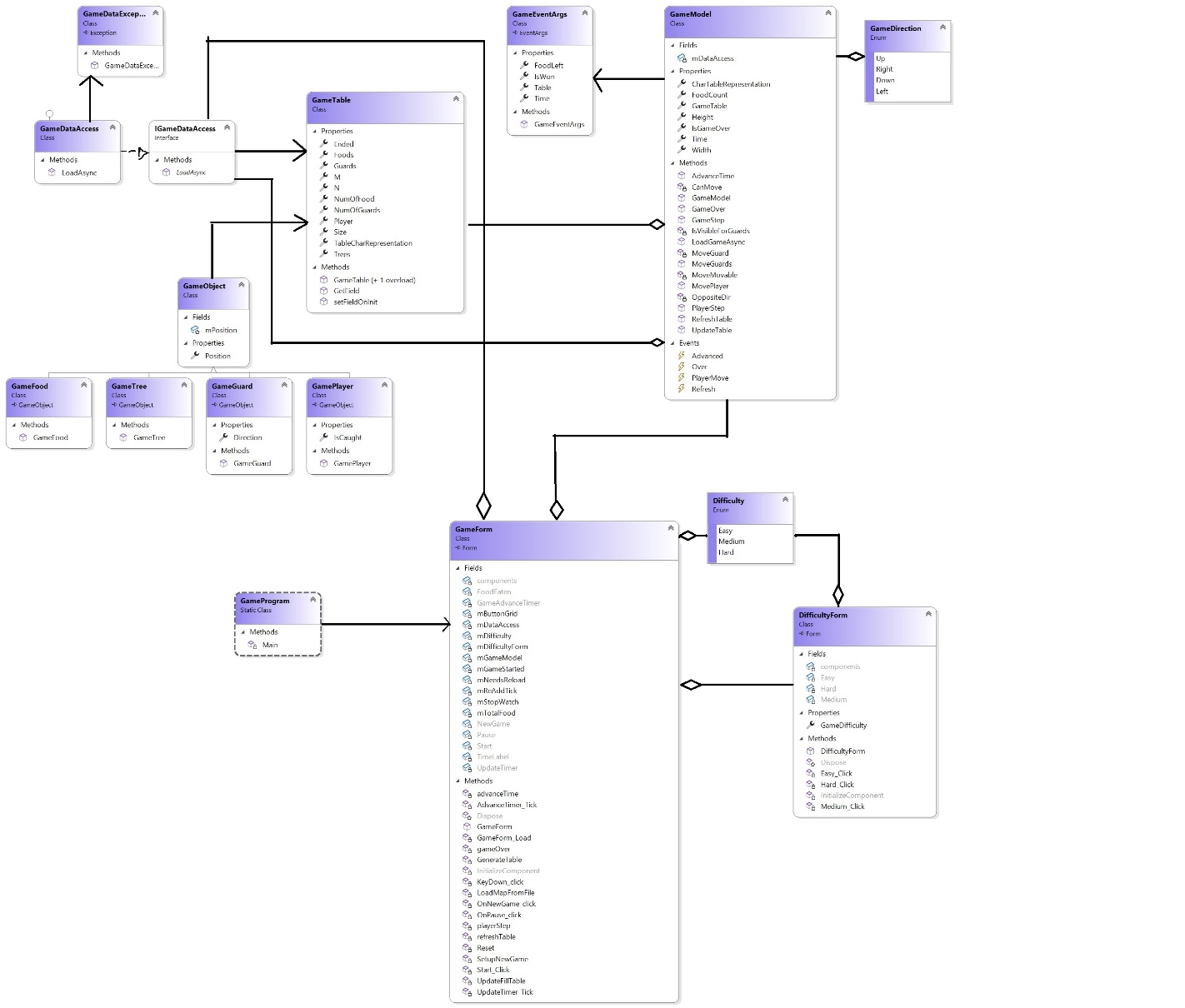
1. **ábra: Az alkalmazás csomagdiagramja**

Modell:

* + A modell lényegi részét a **GameModel** osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék egyéb paramétereit, úgymint az idő (**Time**) és a hátralevő piknikkosarak száma (**FoodCount**). A típus lehetőséget ad lépésre (**GameStep**), vagy a játékos lépésére (**PlayerStep**). Az idő előreléptetését időbeli lépések végzéséve (**AdvanceTime**) tehetjük meg.
  + A játékállapot változásáról a **Advanced** esemény, míg a játék végéről a **Over** esemény tájékoztat. A **Refresh** esemény a karaktertábla frissítéséért felelős, a **PlayerMove** pedig a játékos mozgásáért. Az események argumentuma (**GameEventArgs**) tárolja a győzelem állapotát, a lépések számát, a játékidőt és a játéktáblát.
  + A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**LoadGameAsync**).

Nézet:

* + A nézetet a **GameForm** osztály biztosítja, amely tárolja a modell egy példányát (**mgameModel**), valamint az adatelérés konkrét példányát (**mgameDataAccess**).
  + A játéktáblát egy dinamikusan létrehozott gombmező (**mButtonGrid**) reprezentálja. A felületen létrehozzuk a megfelelő menüpontokat, címkéket, valamint dialógusablakokat, és a hozzájuk tartozó eseménykezelőket. A játéktábla generálását (**GenerateTable**), illetve frissítését (**UpdateFillTable**) külön metódusok végzik.
  + A játék időbeli kezelését két időzítő végzi (**UpdateTimer, Game advanceTimer**), amelyeket mindig aktiválunk játék során, illetve inaktiválunk, amennyiben bizonyos menüfunkciók futnak. Az előbbi a játéktábla folyamatos frissítéséért, míg az utóbbi az őrök másodpercenkénti léptetéséért felelős-
  + A játék nehézségét a **Difficulty** felsorolási típuson át kezeljük, és a **DifficultyForm** osztályban tároljuk és kérdezzük le.
* A program teljes statikus szerkezete a 3. ábrán látható.



1. **ábra: Az alkalmazás osztálydiagramja**

# Tesztelés:

* + A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a

**GameTest** osztályban.

* + Az alábbi tesztesetek kerültek megvalósításra:
    - **Start**
    - **StepTest**
    - **AdvanceTimeTest**
    - **EatFood**