1. Beadandó feladat dokumentáció

Készítette:
Szabó Adorján Ottó

E-mail:

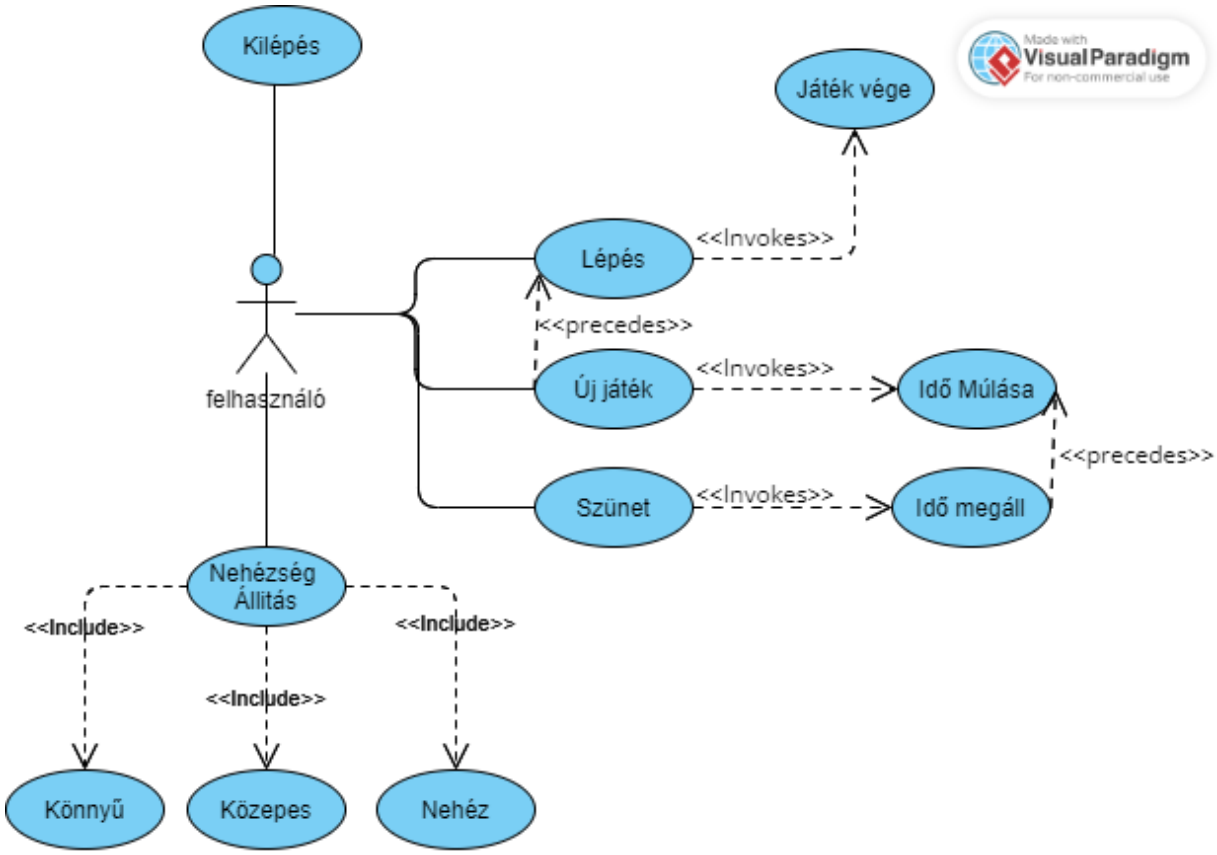
s4i92x@inf.elte.hu

Feladat:

Készítsünk programot, amellyel a következő játékot játszhatjuk. Adott egy $n \times n$ elemből álló játékpálya, amely labirintusként épül fel, azaz fal, illetve padló mezők találhatóak benne, illetve egy kijárat a jobb felső sarokban. A játékos célja, hogy a bal alsó sarokból indulva minél előbb kijusson a labirintusból. A labirintusban nincs világítás, csak egy fáklyát visz a játékos, amely a 2 szomszédos mezőt világítja meg (azaz egy 5 × 5-ös négyzetet), de a falakon nem tud átvilágítani. A játékos figurája kezdetben a bal alsó sarokban helyezkedik el, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán. A pályák méretét, illetve felépítését (falak, padlók) 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), továbbá ismerje fel, ha vége a játéknak. A program játék közben folyamatosan jelezze ki a játékidőt.

Elemzés:

• A játékot három nehézségi szinttel játszhatjuk: könnyű (10x10es pálya), közepes (11x11es pálya), nehéz (12x12es pálya). A program indításkor könnyű nehézséget állít be.
• A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.
• Az ablakban elhelyezünk egy menüt a következő menüpontokkal: File (Új játék, Szünet, Kilépés), Beállítások (Könnyű, Közepes,
Nehéz). Az ablak alján megjelenítünk egy státuszsort, amely a hátralévő időt jelzi.
priossal a fal,zöldel az út és kékkel a játékos van jelölve.
• A játéktáblát egy n × n Label-ekből álló rács reprezentálja. A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak
(a jobb felső sarokba értünk). Szintén dialógusablakokkal történik a szünet is.
• A felhasználói esetek az 1. ábrán láthatóak.



1. ábra: Felhasználói esetek diagramja

Tervezés:

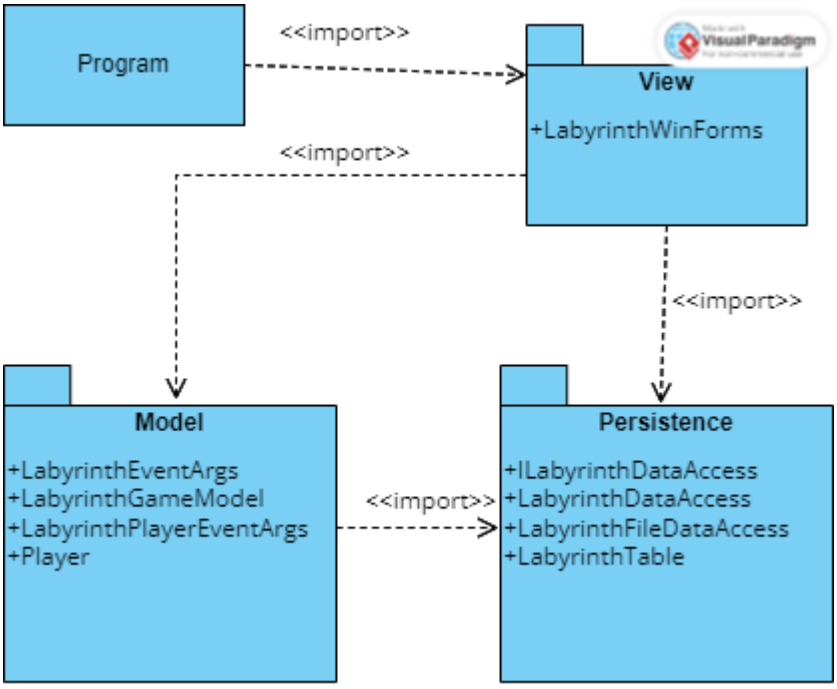
- Programszerkezet:
- A programot háromrétegű architektúrában valósíjuk meg. A megjelenítés a **`View`**, a modell a **`Model`**, míg a perzisztencia a **`Persistence`** névtérben helyezkedik el. A program csomagszerkezete a 2. ábrán látható.

- A program szerkezetét két projektre osztjuk implementációs megfontolásból: a **`Persistence`** és **`Model`** csomagok a program felületfüggetlen projektjében, míg a **`View`** csomag a Windows Formstól függő projektjében kap helyet.

Perzisztencia:

- Az adatkezelés feladata a Labyrinth táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
- A **`LabyrinthTable`** osztály egy Boolean-okal kitöltött labirintust biztosit. A pálya majd egy txt file-ból lesz betöltve. A konstruktorba megkell adni a pálya méretét amit automatikusan feltölt *false*-al.A tábla lehetőséget az értékek változtatására/lekérdezésére.

-
  - A hosszú távú adattárolás lehetőségeit az **`IlabyrinthDataAccess`** interfész adja meg, amely lehetőséget ad a tábla betöltésére LoadEasy,LoadMedium,LoadHard ,Load.
  - Az interfészt szöveges fájl alapú adatkezelésre a **`LabyrinthFileDataAccess`** osztály valósítja meg. A fájlkezelés során fellépő hibákat a **`LabyrinthDataException`** kivétel jelzi.
  - A program az adatokat szöveges fájlból nyeri ki.
    - A fájl első sora megadja a tábla méretét, majd nxn méretben Boolean értékek vannak.



2. ábra: Az alkalmazás csomagdiagramja

Modell:

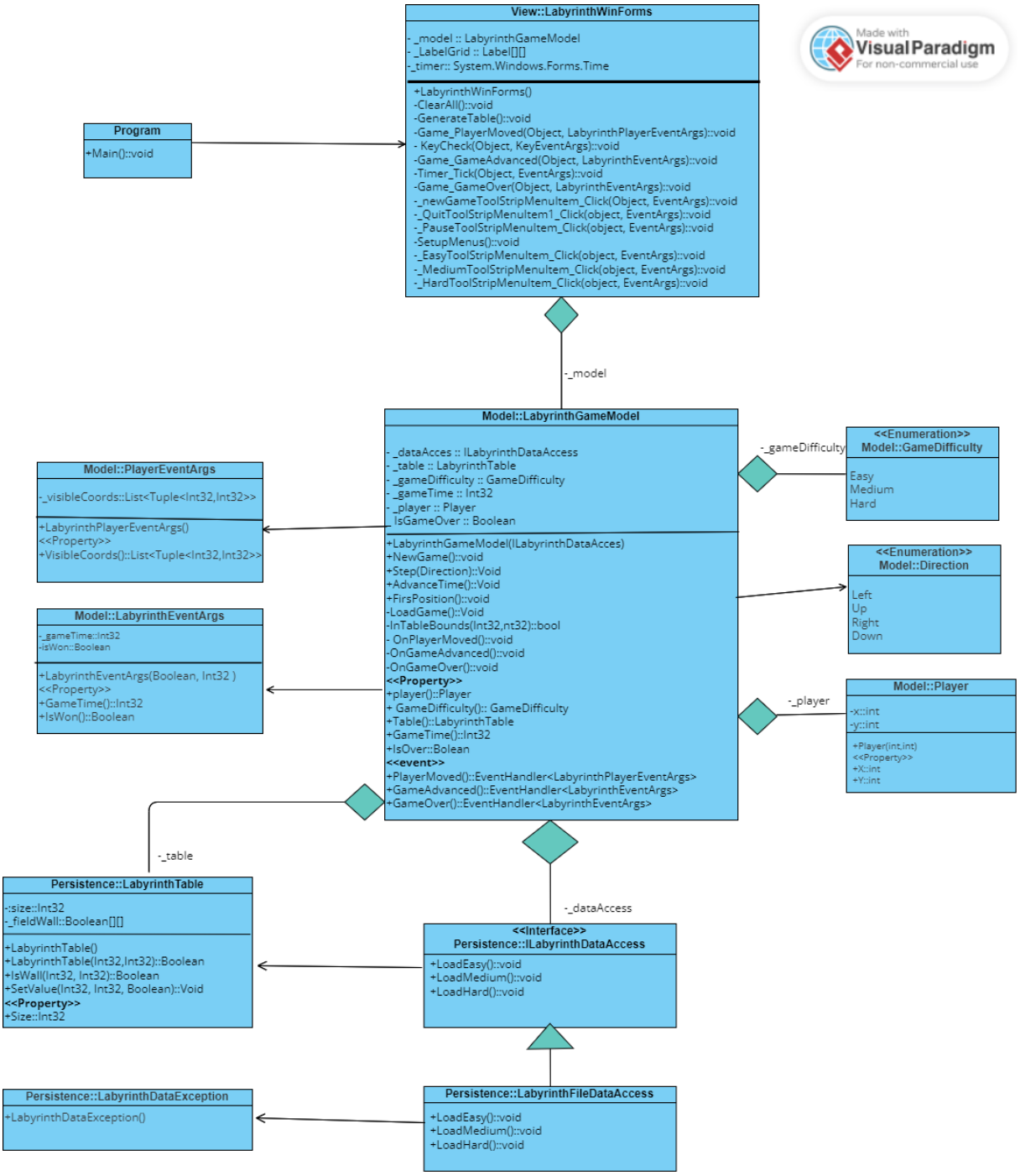
- A modell lényegi részét a **`LabyrinthGameModel`** osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék egyéb paramétereit, úgymint az idő (**`_gameTime`**) és az hogy a játékos célba ért-e **(_isGameOver)**. A típus lehetőséget ad új játék kezdésére (NewGame), valamint lépésre (**`Step`**). Új játéknál megadható,hogy melyik pálya legyen.
- A játékállapot megváltozásáról (vége van-e, hátra lévő idő) a **`GameAdvanced`** esemény, míg a játék végéről a **`GameOver`** esemény tájékoztat. Az események argumentuma (**`LabyrinthEventArgs`**) tárolja a győzelem állapotát, valamint a játékidőt.
- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**`LoadGame`**).

-
    - A játék nehézségét a **`GameDifficulty`** felsorolási típuson át kezeljük.
    - Nézet:
    - A nézetet a Labyrinth**`WinForm`** osztály biztosítja, amely tárolja a modell egy példányát (**`_model`**), valamint az adatelérés konkrét példányát (**`_dataAccess`**).

    - A játéktáblát egy dinamikusan létrehozott (**`_LabelGrid`**) reprezentálja. A felületen létrehozzuk a megfelelő menüpontokat, illetve státuszsort, valamint dialógusablakokat, és a hozzájuk tartozó eseménykezelőket. A játéktábla generálását (**`GenerateTable)`** végzi.

    - A játék időbeli kezelését egy időzítő végzi (**`_timer`**), amelyet mindig aktiválunk játék során, illetve inaktiválunk, amennyiben bizonyos menüfunkciók futnak.

    - A program teljes statikus szerkezete a 3. ábrán látható.



3. ábra: Az alkalmazás osztálydiagramja (mivel nehezen olvasható ezért a kép a file-ok közt is van)

Tesztelés:

• A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a `LabyrinthGameModelTest` osztályban.

• Az alábbi tesztesetek kerültek megvalósításra:

• `LabyrinthGameModelNewGameMediumTest`,`LabyrinthGameModelNewGameEasyTest`, `LabyrinthGameModelNewGameHardTest`,`LabyrinthGameModelStepTest`, `LabyrinthGameModelAdvanceTimeTest`,

`LabyrinthGameModelNewGameMediumTest`: Új játék indítása, megfelelő tábla és nehézség ellenőrzése a nehézségi

fokozat
függvényében(`LabyrinthGameModelNewGameEasyTest`,`LabyrinthGameModelNewGameHardTest ugyan ez`).

- `LabyrinthGameModelStepTest`: Az üres pályán be megyünk a célba és leellenőrizzük, hogy a_model IsWon adattagja megfelelően változik-e
- `LabyrinthGameModelAdvanceTimeTest`: A játékbeli idő kezelésének

ellenőrzése, beleértve azt hogy nem telik az idő hogy ha már beértünk a célba.