Készítette

Baranyai Dominik

E-mail: bdominik0914@gmail.com

Csoportszám: 17

Feladat

3.beadandó / 3.feladat

Elszabadult robot

Készítsünk programot, amellyel a következő játékot játszhatjuk. Adott egy $n \times n$ mezőből álló játékpálya, amelyben egy elszabadult robot bolyong, és a feladatunk az, hogy betereljük a pálya közepén található mágnes alá, és így elkapjuk.

A robot véletlenszerű pozícióban kezd, és adott időközönként lép egy mezőt (vízszintesen, vagy függőlegesen) úgy, hogy általában folyamatosan előre halad egészen addig, amíg falba nem ütközik. Ekkor véletlenszerűen választ egy új irányt, és arra halad tovább. Időnként még jobban megkergül, és akkor is irányt vált, amikor nem ütközik falba.

A játékos a robot terelését úgy hajthatja végre, hogy egy mezőt kiválasztva falat emelhet rá. A felhúzott falak azonban nem túl strapabíróak. Ha a robot ütközik a fallal, akkor az utána eldől. A ledőlt falakat már nem lehet újra felhúzni, ott a robot később akadály nélkül áthaladhat.

A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (7 × 7, 11 × 11, 15 × 15), valamint játék szüneteltetésére (ekkor nem telik az idő, nem lép a robot, és nem lehet mezőt se kiválasztani). Ismerje fel, ha vége a játéknak, és jelenítse meg, hogy milyen idővel győzött a játékos. A program játék közben folyamatosan jelezze ki a játékidőt. Ezen felül szüneteltetés alatt legyen lehetőség a játék elmentésére, valamint betöltésére.

Elemzés:

• A játékot három nehézségi szinttel játszhatjuk: könnyű (1 ms/lépés, 7 széles pálya), közepes (0,8 ms/lépés, 11 széles pálya), nehéz (0,7 ms/lépés c, 15 széles pálya). A program indításkor könnyű nehézséget állít be, és automatikusan új játékot indít.

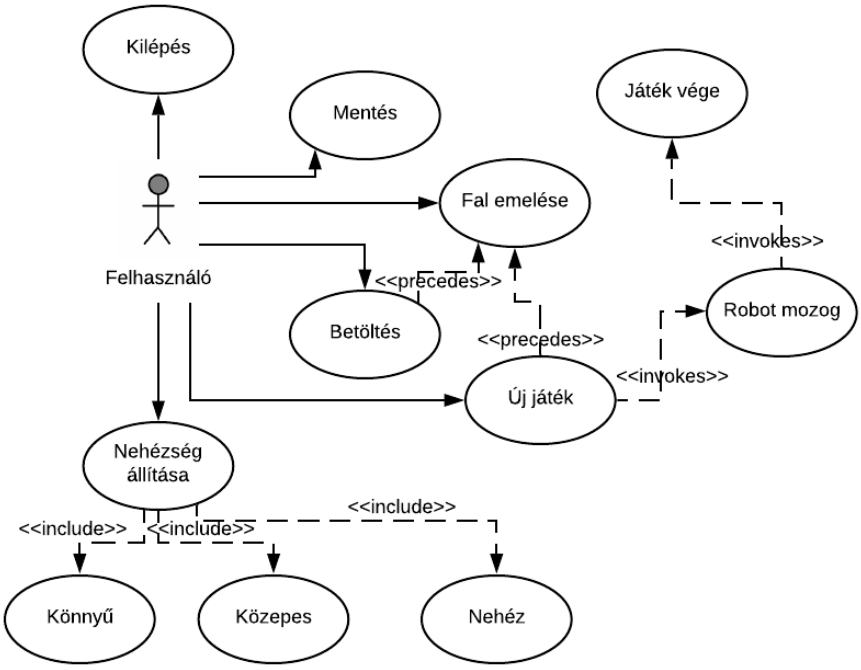
• A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.

• Az ablakban elhelyezünk egy menüt a következő menüpontokkal: File (Új játék, Játék megállítása/elindítása, Játék betöltése, Játék mentése, Kilépés), Beállítások (Könnyű játék, Közepes játék, Nehéz játék). Az ablak alján megjelenítünk egy státuszsort, amely a nehézség szintjét, illetve az időt jelzi.

• A játéktáblát egy n × n nyomógombokból álló rács reprezentálja. A nyomógomb egérkattintás hatására falat emel az adott területre, amire ezután nem tudunk kattintani.

• A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (a robot elérte a pálya közepét). Szintén dialógusablakokkal végezzük el a mentést, illetve betöltést, a fájlneveket a felhasználó adja meg.

• A felhasználói esetek az alábbi ábrán láthatóak:



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:

  • A betöltés/mentés biztosítása.

  • A hosszú távú adattárolás lehetőségeit az IRobotDataAccess 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 RobotFileDataAccess osztály valósítja meg. A fájlkezelés során fellépő hibákat a RobotDataException kivétel jelzi.

  • A program az adatokat szöveges fájlként tudja eltárolni. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.

  • A fájl nxn sora és oszlopa adja meg a tábla adatait, majd az utolsó sorban található meg az eltelt idő.

• Modell:

• A modell lényegi részét a RobotGameModel osztály valósítja meg, amely szabályozza a tábla és a robot tevékenységeit. A típus lehetőséget ad új játék kezdésére (NewGame), valamint fal emelésre (RaiseWall). Új játéknál megadható a kiinduló játéktábla is, különben automatikusan generálódnak kezdő mezők. Az idő előre léptetését időbeli lépések végzésével halad a robot (Step).

• A játékállapot változásáról és a játék végéről UpdateGame esemény, míg az idő teléséről TimerEvent esemény tájékoztat. Az események argumentuma (bool) tárolja a játék végének állapátotát, és (string) ami a játék idejét tárolja.

• A modell példányosításkor létrehoz egy adatkezeléses felületet, amelynek segítségével lehetőséget ad betöltésre (LoadGameAsync) és mentésre (SaveGameAsync)

• A játék nehézségét a GameDifficulty felsorolási típuson át kezeljük, és a RobotGame osztályban konstansok segítségével tároljuk az egyes nehézségek paramétereit.

• A robot mozgásának irányát a RobotDirection felsorolási típuson át kezeljük, és a RobotGame osztályban konstansok segítségével tároljuk az egyes irányok paramétereit.

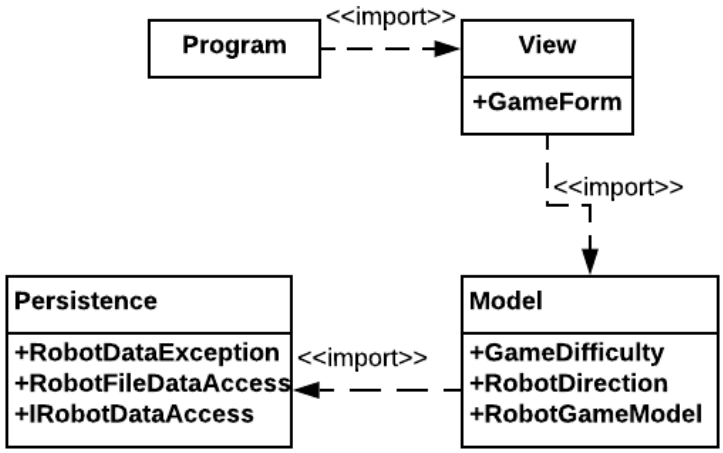
• A játék időbeli kezelését két időzítő végzi (gameTimer, robotTimer), amelyet mindig aktiválunk játék során, illetve inaktíválunk, amennyiben bizonyos menüfunkciók futnak.

• A gameTimer maga az eltelt idő mutatására szolgál, míg a robotTimer a robot lépésére szolgál aminek gyorsasága változik nehézség alapján.

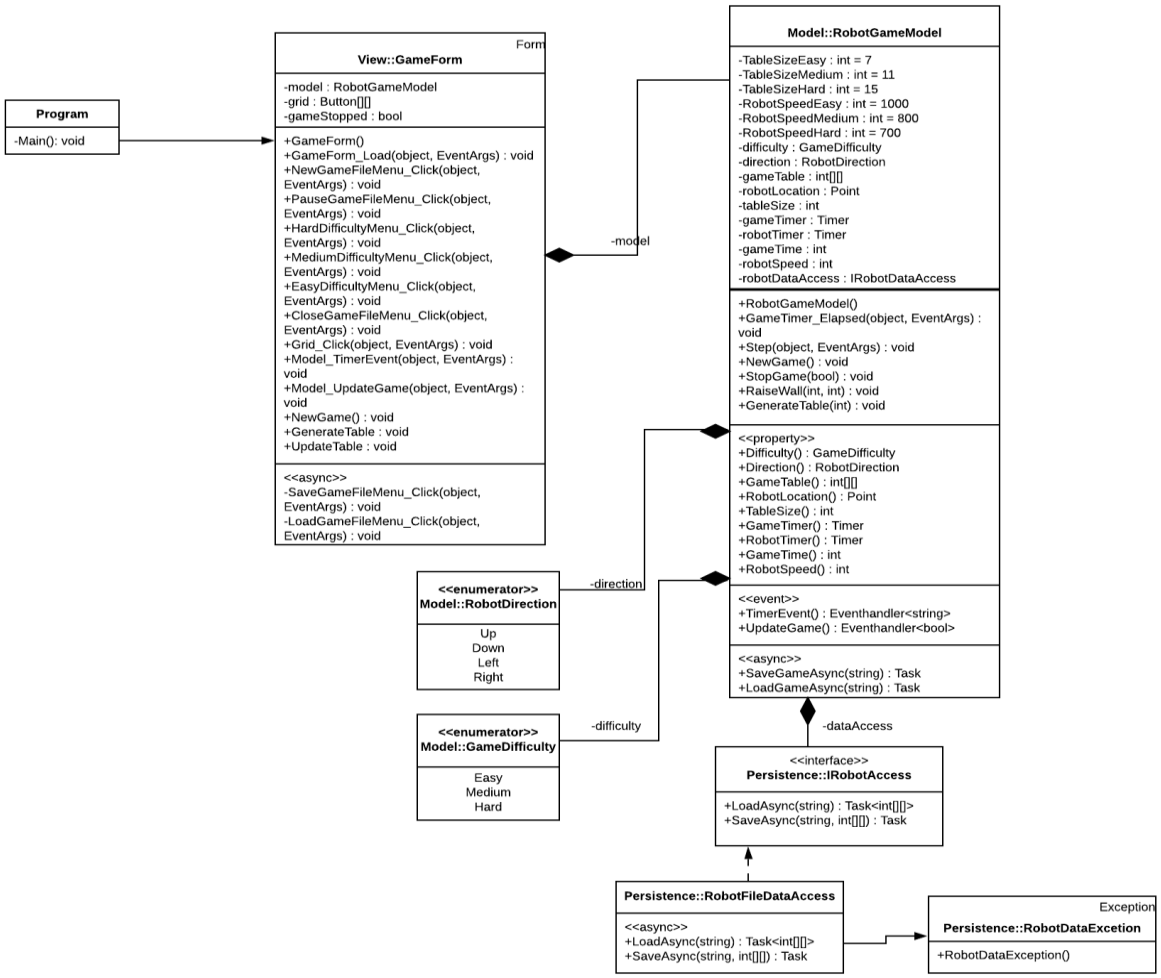
• Nézet:

• A nézetet a GameForm osztály biztosítja, amely tárolja a modell egy példányát (model).

• A játéktáblát egy dinamikusan létrehozott gombmező (Grid) reprezentálja. A felületen létrehozzuk a megfelelő menüpontokat, illetve státuszsort, valamint dialógusablakokat, és a hozzájuk tartozó eseménykezelőket. A játéktábla generálását (GenerateTable), illetve az értékek beállítását (UpdateTable) külön metódusok végzik.



• A program teljes statikus szerkezete az alábbi ábrán látható:



Tesztelés:

• A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a RobotGameModelTest osztályban.

• Az alábbi tesztesetek kerültek megvalósításra:

  • RobotGameModelNewGameEasyTest, RobotGameModelNewGameMediumTest, RobotGameModelNewGameHardTest: Új játék indítása, a mezők kitöltése, valamint a táblaméret, robot sebesség és nehézség ellenőrzése a nehézségi fokozat függvényében.

  • RobotGameModelStopGame: Játékbeli időzítők ellenőrzése.

  • RobotGameModelRaiseWall: A játékbeli a tábla módosítása.