„Eseményvezérelt Alkalmazások”

1.Beadandó dokumentáció

4.feladat

Készítette: Székely Regő

Neptun-azonosító: H5LJFS

E-mail: h5ljfs@inf.elte.hu

2023.október 21.

Feladat

4. Labirintus

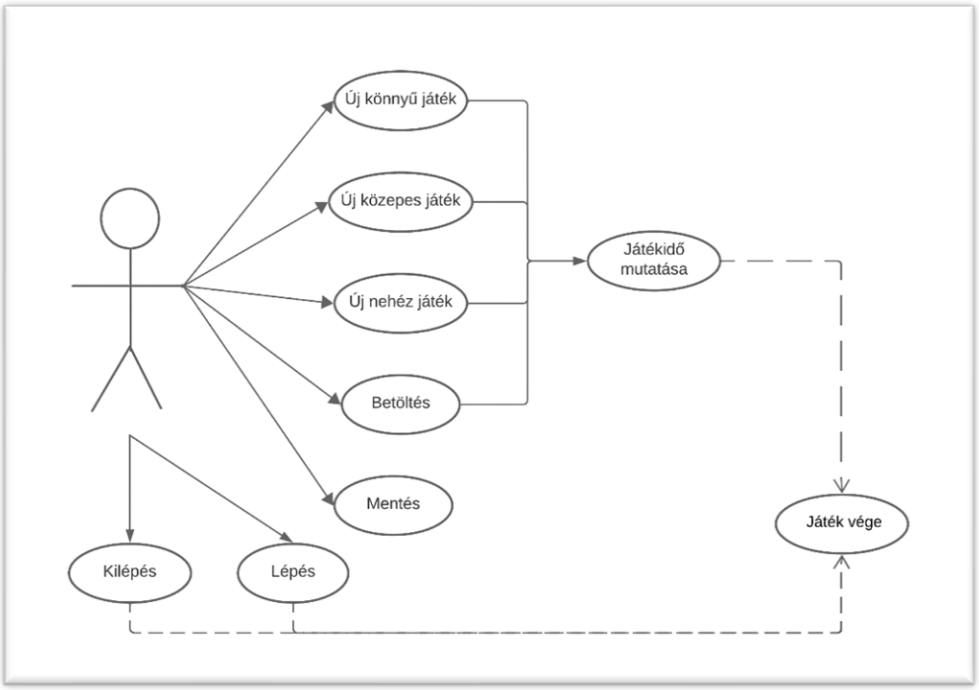
Készítsünk programot, amellyel a következő játékot játszhatjuk.
Adott egy $n \times n$ elemből álló játékpálya, amely labirintusként épül fel, azaz fal, illetve padló mezők találhatóak benne, illetve egy kijárat a jobb felső sarokban. A játékos célja, hogy a bal alsó sarokból indulva minél előbb kijusson 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 kiválasztott pályamérettel használhatjuk: 10 X 10, 15 X 15, 20 X 20. Ezeket az „Easy”, „Medium” és „Hard” jelzőkkel választhatjuk ki. Érdemes megjegyezni azt, hogy ebben az esetben minél nagyobb a pálya annál nehezebb a játék hiszen a több fallal rendelkező pályák bonyolultabb utakat eredményeznek. A program indításkor New Game gombbal indítható a játék.
- A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.
- Az ablak aljában elhelyezem státuszsort, ami tartalmazza a szüneteltetés (Pause) gombot és az eltel idő jelzésére szolgáló számlálót. A felső menüsávban található a lenyíló menüpont, ahol a következők találhatóak: új játék (New Game), betöltés (Load Game), mentés (Save Game) és kilépés (Exit Game).
- A játéktábla a kiválasztott méret alapján generálódik le.
- A játékos mozgása a W, A, S, D gombokkal történik (Fel-Bal-Le-Jobb).
- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak rajta megfelelő üzenettel.
- A mentés és betöltés helyét előre meg kell adni a programba változóként. Betöltés után rögtön elindul a játék.

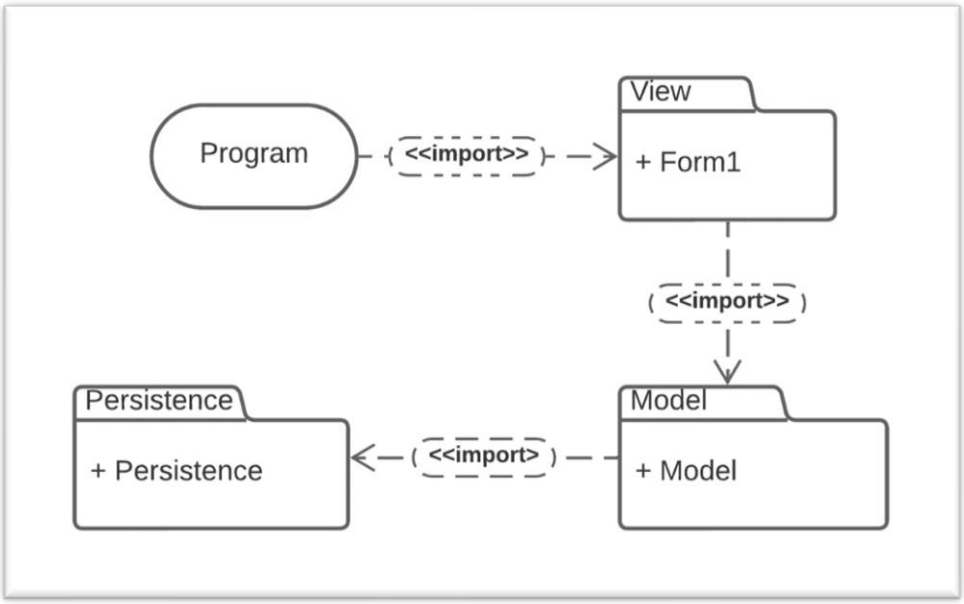
- Az ábrán láthatóak a felhasználói esetek:



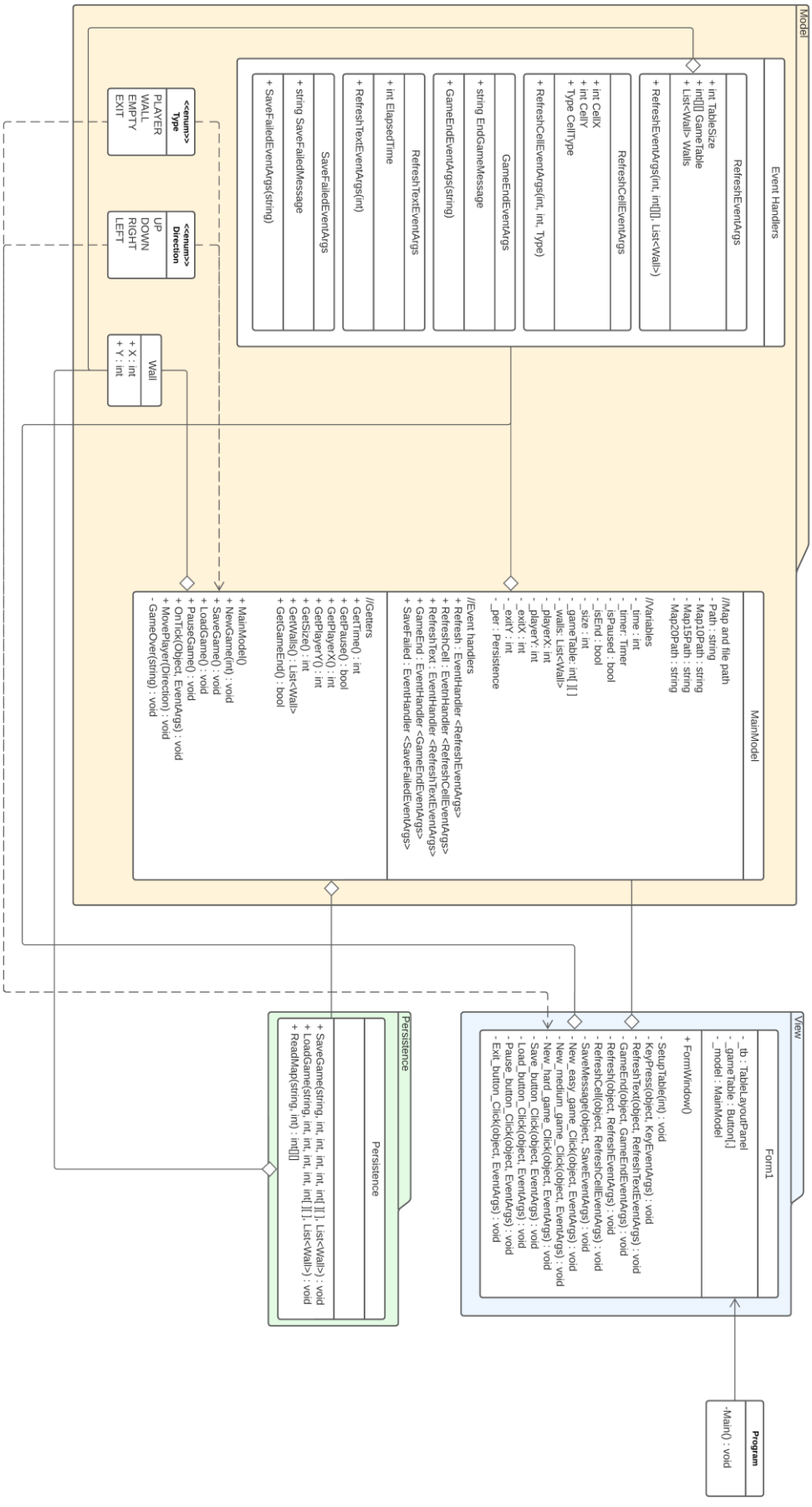
Tervezés:

- *Programszerkezet:*
  - A programot háromrétegű architektúrában van megvalósítva. A megjelenés a Form1, a modell a Model, míg a perzisztencia a Persistence fájlokban helyezkedik el. A program csomagszerkezete a 2. ábrán látható.
- *Perzisztencia:*
  - A betöltés és a mentés feladatát látja el. Illetve saját hibát is dob, ha sikertelen a mentés vagy a betöltés.
  - A program az adatokat egy szövegfájlban tárolja, amit a megadott mentési útvonalon bármikor el lehet menteni. Hiba lép fel, ha nem létezik a megadott útvonal.
  - A mentési fájl felépítése:
    - Eltelt idő
    - Játékos X pozíciója
    - Játékos Y pozíciója
    - Tábla mérete
    - A tábla az utak és falak eloszlásával
    - Jelen lévő falak pozíciója (X, Y) egymás alatt szóközzel elválasztva.
- *Modell:*
  - A modell lényegi részét a MainModel osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék paramétereit úgy, mint az idő(_time), játékos pozíciója (_playerX és _playerY) stb. A típus lehetőséget ad új játék kezdésére (NewGame), valamint lépésre (MovePlayer). Új játéknál

a kiválasztott méret alapján legenerálódnak a falak, a játékos és a kijárat. Az idő haladását időbeli lépések végzése (OnTick) teszi lehetővé.
  - A játékállapot változásáról az OnTick és a MovePlayer esemény gondoskodik, hiszen a kettő egymástól függetlenül működik. A játék végét a GameOver esemény kezeli.
  - A modell példányosítása során példányosítva lesz a timer is, aminek Tick adattagjához hozzá lesz rendelve az OnTick esemény, ami kiváltja az idő előrehaladását.
  - A játéktér előre el van tárolva.
  - A játék időbeli kezelését egy időzítő végzi (_timer), amelyet mindig aktiválunk játék során, illetve inaktiváljuk, amennyiben a játék megállításra kerül.



- *Nézet:*
  - A nézetet a Form osztály biztosítja, amely tárolja a modell egy példányát (_model), a gombokat (_gameTable), amik a mezőkként szerepelnek, a perzisztenciát (_per) és az elérési útvonalat (Path).
  - A játéktábla egy dinamikusan létrehozott TableLayoutPanel (_tb) reprezentálja, amiben gombokat helyezkednek el. A felületen létrehozott menüpont és a gombok egy státuszsorban helyezkednek el. Az új játékterület felállítását a SetupTable valósítja meg, ami a New_game_Click eseménykezelőtől megkapta a gomb lenyomásakor a méretet.
- A program teljes statikus szerkezete az alábbi képen látható:


Model
Event Handlers
RefreshEventArgs
+ int TableSize
+ int[][] GameTable
+ List<Vall> Walls
+ RefreshEventArgs(int, int[][], List<Wall>)
RefreshCellEventArgs
+ int CellX
+ int CellY
+ Type CellType
+ RefreshCellEventArgs(int, int, Type)
GameEndEventArgs
+ string EndGameMessage
+ GameEndEventArgs(string)
RefreshTextEventArgs
+ int ElapsedTime
+ RefreshTextEventArgs(int)
SaveFailedEventArgs
+ string SaveFailedMessage
+ SaveFailedEventArgs(string)
<<enum>>
Type
PLAYER
WALL
EMPTY
EXIT
<<enum>>
Direction
UP
DOWN
RIGHT
LEFT
Wall
+ X : int
+ Y : int
MainModel
//Map and file path
- Path : string
- Map10Path : string
- Map15Path : string
- Map20Path : string
//Variables
- _time : int
- _timer: Timer
- _isPaused : bool
- _isEnd : bool
- _size : int
- _gameTable: int[,][]
- _walls: List<Wall>
- _playerX: int
- _playerY: int
- _exitX : int
- _exitY : int
- _per : Persistence
//Event handlers
+ Refresh : EventHandler <RefreshEventArgs>
+ RefreshCell : EventHandler <RefreshCellEventArgs>
+ RefreshText : EventHandler <RefreshTextEventArgs>
+ GameEnd : EventHandler <GameEndEventArgs>
+ SaveFailed : EventHandler <SaveFailedEventArgs>
//Getters
+ GetTime() : int
+ GetPause() : bool
+ GetPlayerX() : int
+ GetPlayerY() : int
+ GetSize() : int
+ GetWalls() : List<Wall>
+ GetGameEnd() : bool
+ MainModel()
+ NewGame(int) : void
+ SaveGame() : void
+ LoadGame() : void
+ PauseGame() : void
+ OnTick(Object, EventArgs) : void
+ MovePlayer(Direction) : void
- GameOver(string) : void
View
Form1
- _tb : TableLayoutPanel
- _gameTable : Button[,]
- _model : MainModel
+ FormWindow()
- SetupTable(int) : void
- KeyPress(object, KeyEventArgs) : void
- RefreshText(object, RefreshTextEventArgs) : void
- GameEnd(object, GameEndEventArgs) : void
- Refresh(object, RefreshEventArgs) : void
- RefreshCell(object, RefreshCellEventArgs) : void
- SaveMessage(object, SaveEventArgs) : void
- New_easy_game_Click(object, EventArgs) : void
- New_medium_game_Click(object, EventArgs) : void
- New_hard_game_Click(object, EventArgs) : void
- Save_button_Click(object, EventArgs) : void
- Load_button_Click(object, EventArgs) : void
- Pause_button_Click(object, EventArgs) : void
- Exit_button_Click(object, EventArgs) : void
Persistence
Persistence
+ SaveGame(string, int, int, int, int, int[][], List<Wall>) : void
+ LoadGame(string, int, int, int, int, int[][], List<Wall>) : void
+ ReadMap(string, int) : int[][]
Program
-Main() : void

Tesztelési terv:

- A modell működését egységtesztek segítségével lett vizsgálva (GameTest osztály).
- Megvalósított tesztesetek:
    - NewGameTest1: Új játék létrehozásának elemzése és annak változói ellenőrzése 10 x 10-es tábla esetén.
    - NewGameTest2: Új játék létrehozásának elemzése és annak változói ellenőrzése 15 x 15-es tábla esetén.
    - NewGameTest3: Új játék létrehozásának elemzése és annak változói ellenőrzése 20 x 20-es tábla esetén.
    - PlayerMovementTest: Játékos mozgásának ellenőrzése minden irányba, illetve, hogy nem mozdul pályán kívülre.
    - GameEndTest: Játék végállapotának vizsgálata indítás előtt.
    - TimeTest: Állapotvizsgálatok, hogy az időzítő tényleges rendben működik.
    - EndGameTest: A játék végállapotának vizsgálata
    - HitWallTest: A játékos pozíciójának vizsgálata falnak ütközés esetén.