Beadandó dokumentáció

4. feladatsor

Készítette:
Szabó Krisztián
E-mail: cdx5eo@inf.elte.hu

Tartalom

1 Feladat

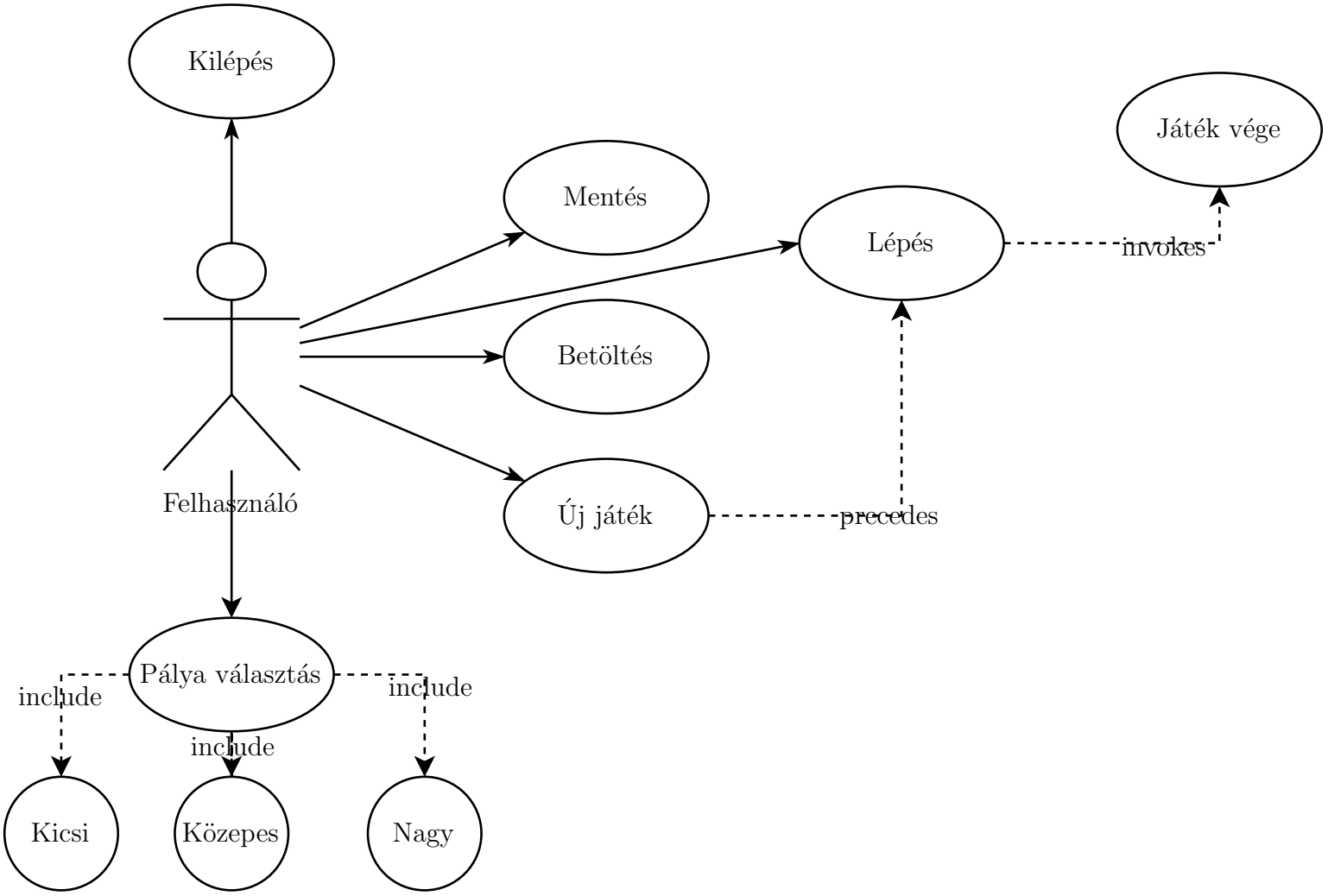
Labirintus

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.

2 Elemzés

- A játék három pályát (kicsi, közepes, nagy) tartalmaz. A program indításakor a felhasználó látni fog egy menüt, ahol ki tudja választani a pálya típusát.
- A feladatot egyablakos asztali alkalmazásként WPF grafikus felülettel valósítjuk meg.
- A menüben az aláppi opciók vannak:
  1. Start
  2. Játék betöltése
  3. Kilépés
  4. Pályaméret választás
  5. Játék megállítása
- A labirintust az adott pályaméretnek (11×11, 21×21, 35×35) megfelelő méretű *UniformGrid* reprezentálja, szintúgy a játékost. A játék kijelzi a kezdés óta eltelt időt, ami folyamatosan változik (kivéve ha a játék meg lett állítva).
- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (a játékos kijutott a labirintusból). Szintén dialógusablakokkal végezzük el a mentést, illetve betöltést, a fájlneveket a felhasználó adja meg.
- A felhasználói esetek ábrán láthatóak.

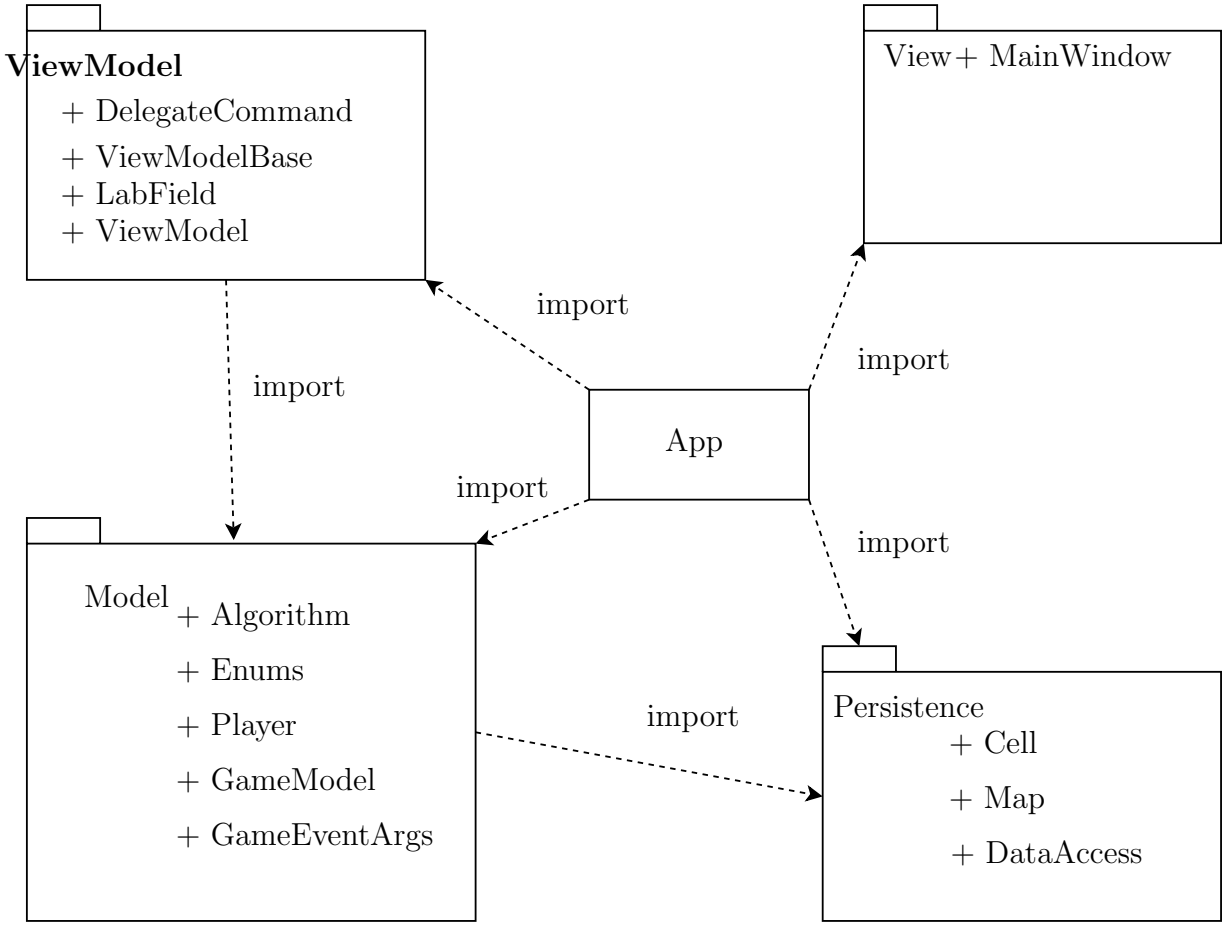
2.1 Felhasználói esetek diagramja



3 Tervezés

- Programszerkezet:
  - A programot MVVM architektúrában valósítjuk meg, ennek megfelelően View, Model, ViewModel és Persistence névtereket valósítunk meg az alkalmazáson belül. A program környezetét az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést. 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 ViewModel és View csomagok a WPF függő projektjében kapnak helyet.
- Perzisztencia:
  - Az adatkezelés feladata a Labirintus pályával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
  - A labirintust egy fájlból olvassa be a *DataAcess* osztály egy osztályszintű metódusa. A Labirintust egy tömbként tároljuk, amit a *Map* osztály reprezentál. A *Cell* osztály reprezentálja a labirintus *celláit*, amiről le lehet kérdezni, hogy az fal-e. Lehetőségünk van egy fájlból beolvasni egy labirintust / fájlba kiírni egy labirintust (elmenteni a játékot). A *DataAccess* osztály kommunikál a *GameModel* osztállyal (játék betöltése, mentése).
  - A *GameModel* észleli, ha a játékos fájlból akar játékot betölteni vagy új játékot akar kezdeni. Ha a játékos új játékot kezd, a kiválasztott pálya beolvasásra kerül és a bal alsó sarok lesz a játékos kezdőpozíciója. A játék betöltésekor megtörténik ugyanez, viszont a mentés előtti pozícióra helyezi a játékost a program.
  - Mivel egy játék nem csak a pálya méretétől és a játékos pozíciójától függ, a játékmódot is el kell tárolni mentéskor.

3.1 Az alkalmazás csomagdiagramja



- Modell:
  - A modell lényegi részét a *GameModel* osztály valósítja meg. Itt történik a játék logikájának lebonyolítása. Ebbe például beletartoznak az alábbiak:
    - Játékos lépéseinek feldolgozása
    - Pálya megvilágítása a játékos pozíciójának és környezetének megfelelően
    - Játékos célbeérésének észlelése

    Ez a három fő része a modellnek.
  - A modell létrehozásakor a *DataAccess* osztály segítségével felpopuláljuk a labirintus *celláit*, majd létrehozzuk a *Player* objektumot, ami valójában csak arra fog szolgálni, hogy eltárolja a jelenlegi pozicíót, amin a játékos áll. A *Map* valósítja meg a labirintus-t, ami egy egyszerű *Cell*[,] tömb. Annak érdekében, hogy a View, majd könnyebben tudja kezelni az új *Cellát*, ahova lép a játékos (vagy amit meg kell világítani miután lépett a játékos) érdemes elvonatkoztatni a tömb indexelésétől és a valós x, y koordinátákat átadni.

  - A jobb olvashatóság szempontjából az alábbi *enum* osztályokat hoztam létre, amiket a nevükből könnyedén rájöhetünk milyen célt fognak szolgálni:
    * *MapSize* (eltárolja a pálya méretét, pl. kicsi)
    * *Arrow* (eltárolja az irályt amerre a játékos lépni szeretne, pl. bal)

- Nézetmodell:
  - A nézetmodell megvalósításához felhasználunk egy általános utasítás (DelegateCommand), valamint egy ős változásjelző (ViewModelBase) osztályt.
  - A nézetmodell feladatait a ViewModel osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, mentéséhez, valamint a kilépéshez. A parancsokhoz eseményeket kötünk, amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja a modell egy hivatkozását, de csupán információkat kér le tőle, illetve a játéknehézséget szabályozza. Direkt nem avatkozik a játék futtatásába.
  - A játékmező számára egy külön mezőt biztosítunk (LabField), amely eltárolja, hogy az adott cellán a játékos, egy cella vagy fal szerepel.

- Nézet:
  - A nézet csak egy képernyőt tartalmaz, a MainWindow osztályt. A nézet egy rácsban tárolja a játékmezőt, a menüt és a státuszsort. A játékmező egy ItemsControl vezérlő, ahol dinamikusan felépítünk egy rácsot (UniformGrid), amely négyzetekből áll. Minden adatot adatkötéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a négyzetek színét is.
  - A fájlnév bekérését betöltéskor és mentéskor, valamint a figyelmeztető üzenetek megjelenését beépített dialógusablakok segítségével végezzük.

- Környezet:
  - Az App osztály feladata az egyes rétegek példányosítása (App_Startup), összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.

3.2 Statikus szerkezet

Az osztályok az alábbi módon épülnek fel:

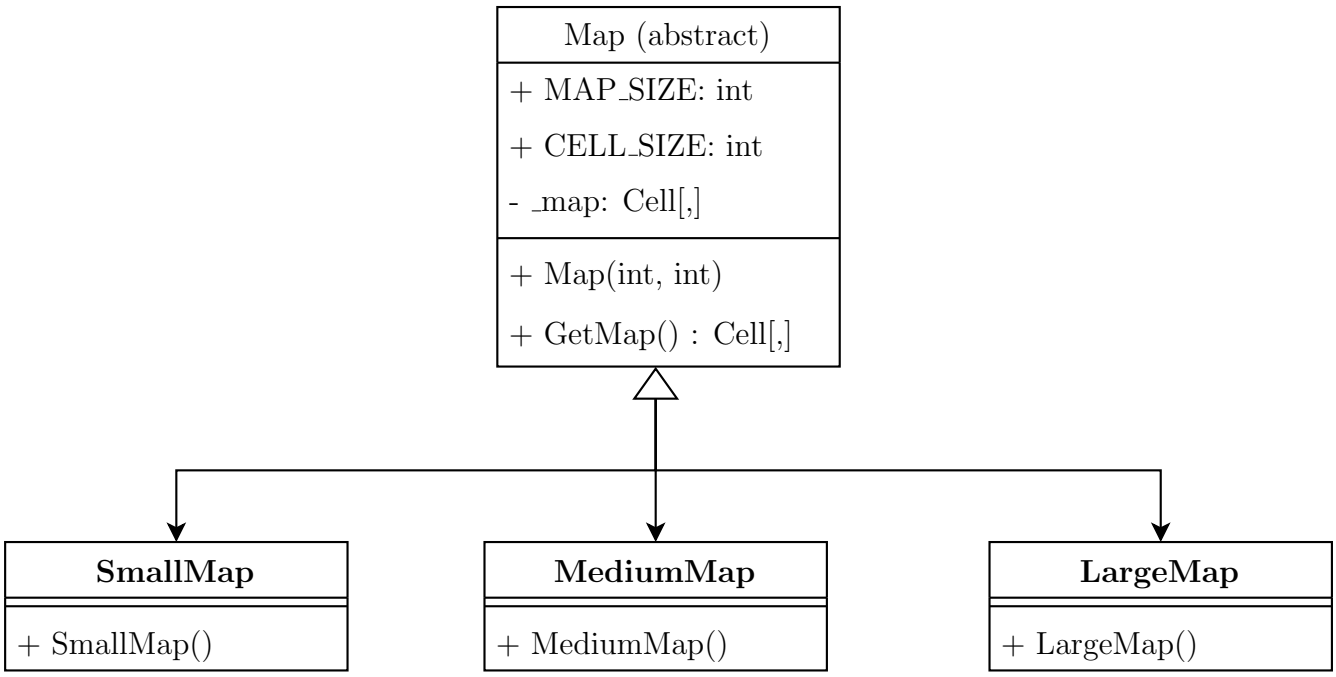
3.2.1 Adatelérés osztály

DataAccess
+ SaveFile(string, Map, enum Gamemode, Point) : static void
+ ReadFromFile(string, Map) : static void
+ LoadFromFile(string, Map, out Point, out enum MapSize) static enum Gamemode
- GetGameMode(string) : static enum Gamemode
- GetMapSize(string) : static enum MapSize
- GetPlayerPosition(string) : static enum Point

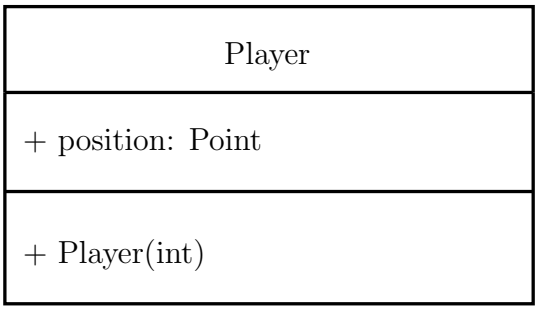
3.2.2 Cella osztály

Cell
- _isWall: bool - _position: Point
+ Cell(char, int, int) + GetPosition() : Point + IsWall() : Bool

3.2.3 Pálya osztály

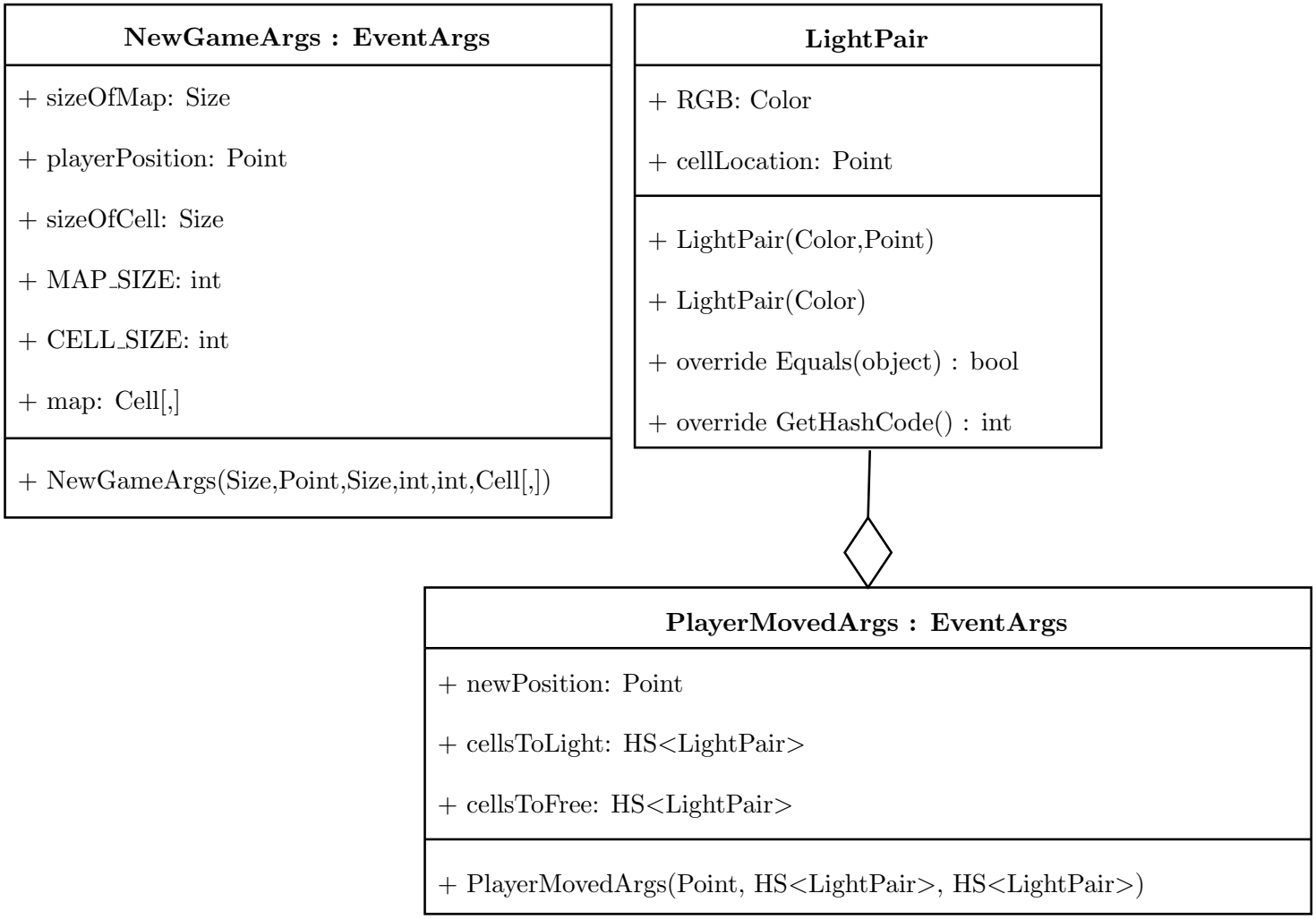


3.2.4 Játékos osztály

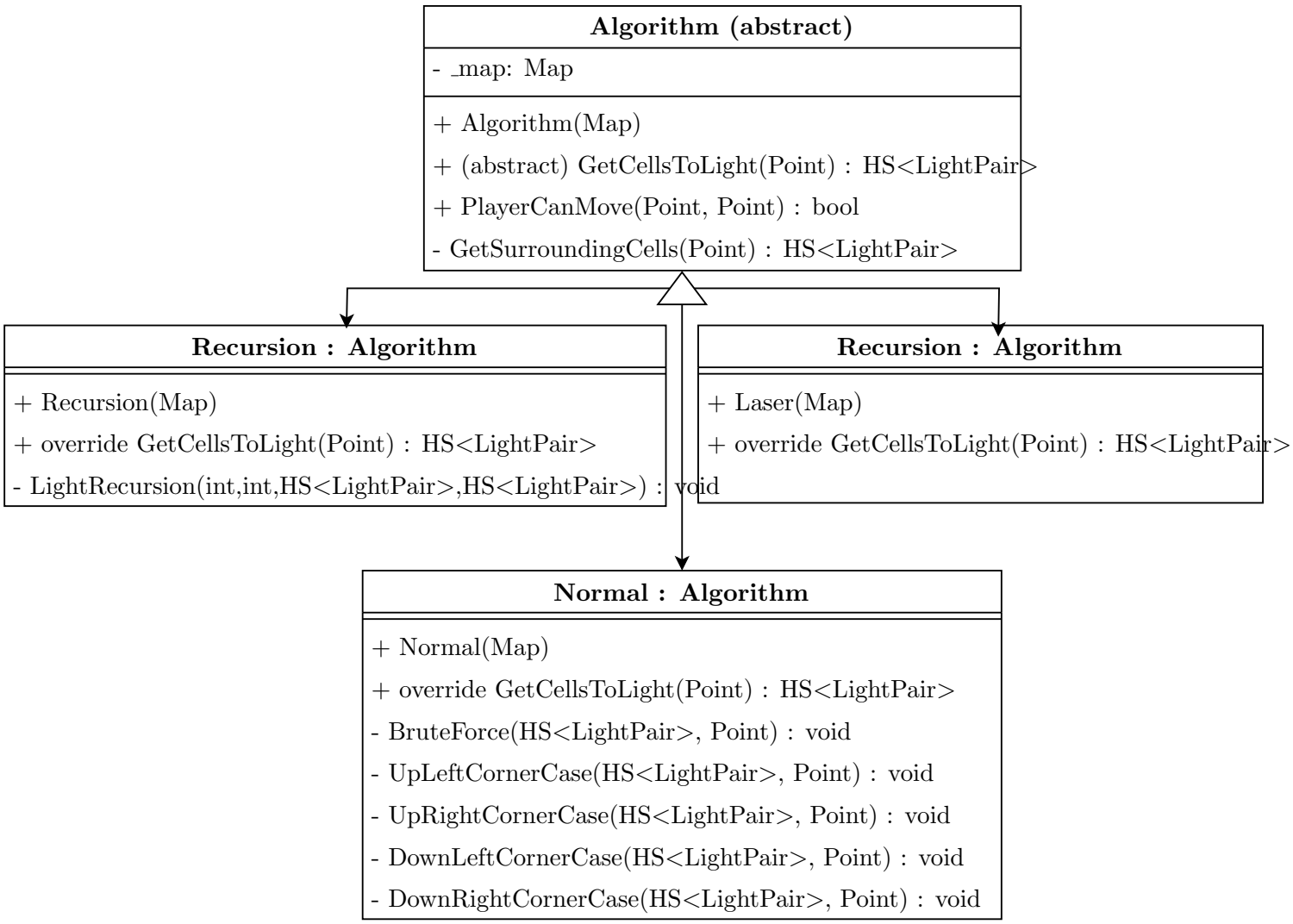


3.2.5 Esemény argumentum osztály

A "HS" rövidítése a HashSet típusnak.



3.2.6 Algoritmus osztály



3.2.7 Modell osztály

GameModel
+ newGame: event<NewGameArgs>
+ playerMoved: event<PlayerMovedArgs>
+ playerWon: event<EventArgs>
- _map: Map
- _player: Player
- _algo: Algorithm
- _cellsToFree: HS<Point>
- _gamemode: enum Gamemode
+ GameModel()
+ StartNewGame(enum MapSize, enum Gamemode) : void
+ LoadNewGame(string) : void
- CreateMap(enum MapSize) : void
- CreateGameMode(enum Gamemode) : void
- OnNewGame() : void
+ PlayerWantsToMove(enum Arrow) : void
- MovePlayer(Point) : void
- OnPlayerMoved() : void
- ModifyCellsToFree(HS<LightPair>) : void
- OnPlayerMoved() : void
+ SaveGame(string) : void

3.2.8 DelegateCommand osztály

DelegateCommand
- canExecute : Func<Object, Boolean> : {*readonly*}
- execute : Action<Object> : {*readonly*}
+ CanExecute(Object) : Boolean
+ DelegateCommand(Action<Object>)
+ DelegateCommand(Func<Object, Boolean>, Action<Object>)
+ Execute(Object) : void
+ RaiseCanExecuteChanged() : void
+ CanExecuteChanged() : EventHandler

3.3 LabField osztály

LabField
- isLit : Boolean
- isPlayer : Boolean
+ IsLit() : Boolean
+ IsPlayer() : Boolean

3.4 ViewModelBase osztály

ViewModelBase
OnPropertyChanged(String) : void
ViewModelBase()
+ PropertyChanged() : PropertyChangedEventHander

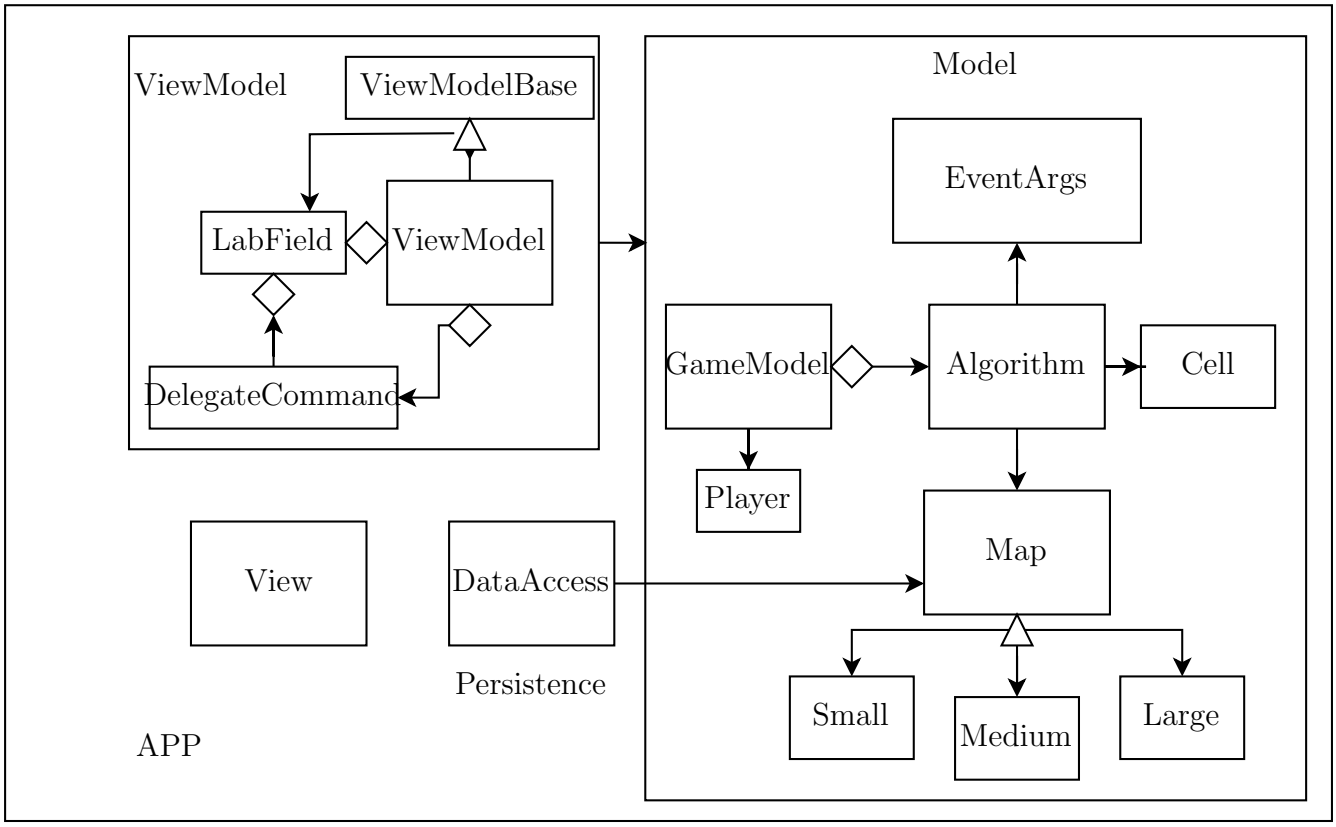
3.5 App osztály

App
- model : GameModel
- window : MainWindow
- viewModel : ViewModel
- timer : DispatcherTimer
+ App()
+ App_Startup(object, StartupEventArgs) : void
- Window_KeyDown(object, KeyEventArgs) : void
- GameModel_PlayerWon(object, EventArgs) : void
- ViewModel_NewGame(object, EventArgs) : void
- ViewModel_LoadGame(object, EventArgs) : void
- ViewModel_SaveGame(object, EventArgs) : void
- ViewModel_PauseGame(object, EventArgs) : void
- ViewModel_ExitGame(object, EventArgs) : void

3.6 ViewModel osztály

ViewModel
- model : GameModel
- previousPlayerPosition : Point
- gameIsPaused : Boolean
- gameTime : TimeSpan
- GameModel_NewGame(object, NewGameEventArgs) : void
- GameModel_PlayerMoved(object, PlayerMovedArgs) : void
+ UserKeyInput(Arrow) : void
- OnNewGame() : void
- OnLoadGame() : void
- OnSaveGame() : void
- OnPausGame() : void
- OnExitGame() : void
+ Fields : ObservableCollection<LabField>
+ GameTime : String
+ IsGameEasy : Boolean
+ IsGameMedium : Boolean
+ IsGameHard : Boolean
+ IsGamePaused : Boolean

3.7 Egész osztálydiagram



4 Tesztelés

Öt fő tesztmetódus szerepel a *Testing* projektben. Miután inicializáltuk a *GameModel* objektumot és beolvastunk hozzá egy pályát (adott esetben a legkisebbet), a *GameModel* osztály ezen funkcióit teszteljük:

- A pálya sikeresen lett létrehozva, nincsen *NullReferenceException*, megfelel a pályaméret
- A játék kezdésekor a *Player* kezdőpozíciója a pálya bal alsó sarka
- A játékos mikor a pályán kívülre akar lépni akkor nem változik meg a pozíciója
- A játékos mikor legális lépést szeretne, a program helyesen megváltoztatja pozícióját
- A játékos sikeresen ki tud jutni a labirintusból és mikor a jobb felső sarokba ér akkor a *GameModel* ezt jelzi
- A játékos nem tud átlépni falakat
- A pálya elmentése sikeresen megtörténik (elmentettük a pálya méretét, játékos pozícióját)
- A pálya betöltése sikeresen megtörténik