

BEADANDÓ DOKUMENTÁCIÓ

4. feladatsor

Készítette:

Szabó Krisztián

E-mail: cdx5eo@inf.elte.hu

Tartalom

1 Feladat	2
2 Elemzés	2
2.1 Felhasználói esetek diagramja	3
3 Tervezés	4
3.1 Az alkalmazás csomagdiagramja	5
3.2 Statikus szerkezet	7
3.2.1 Adatelérés osztály	7
3.2.2 Cella osztály	7
3.2.3 Pálya osztály	8
3.2.4 Játékos osztály	8
3.2.5 Új mentés osztályok	9
3.2.6 Algoritmus osztály	10
3.2.7 Modell osztály	11
3.2.8 DelegateCommand osztály	12
3.2.9 LabField osztály	12
3.2.10 ViewModelBase osztály	13
3.2.11 App osztály	13
3.2.12 ViewModel osztály	14
3.2.13 ViewModel osztály	15
3.2.14 StoredGame osztályok	16
3.3 Egész osztálydiagram	17
4 Tesztelés	18

1 Feladat

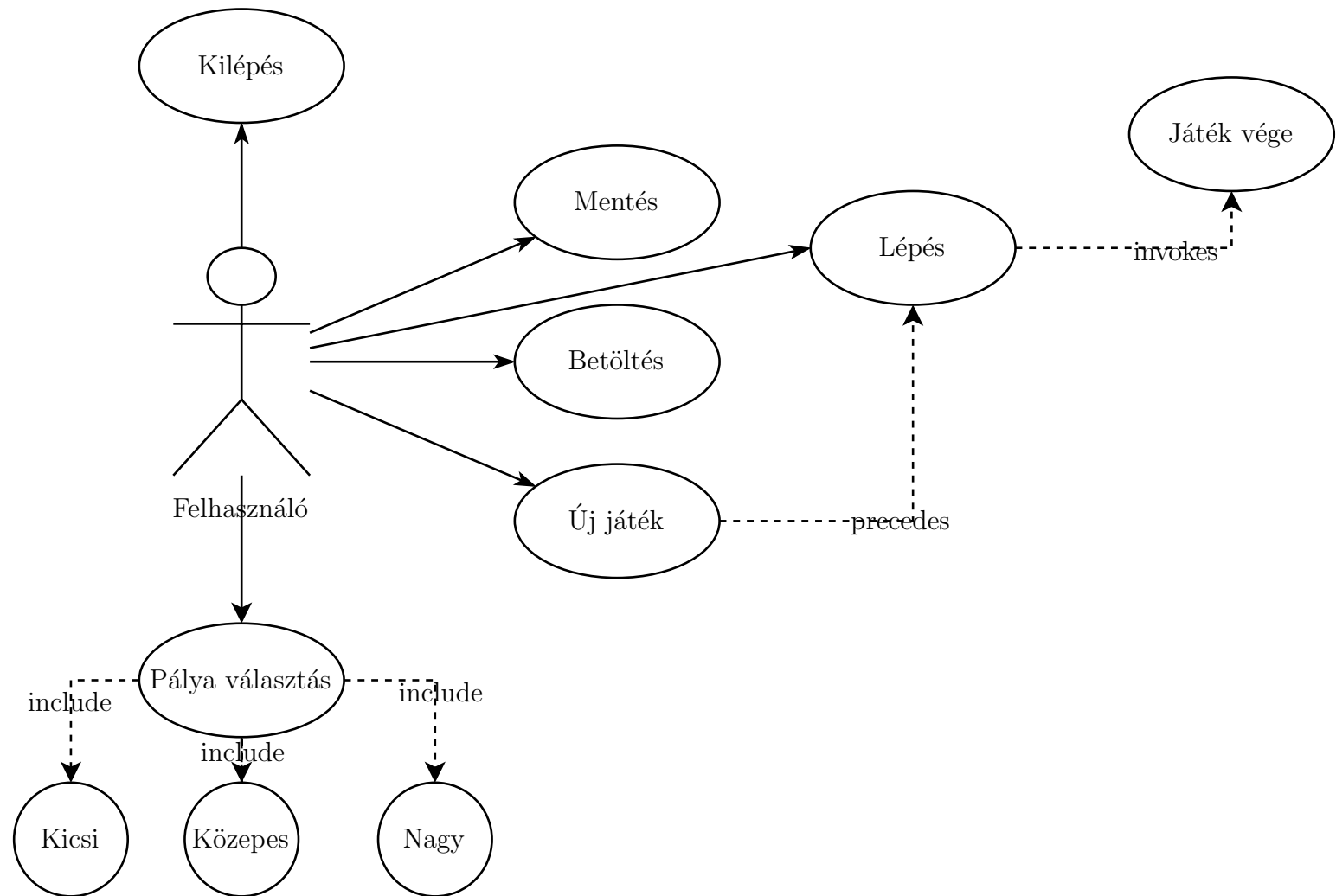
Labirintus

Készítsünk programot, amellyel a következő játékot játszhatjuk. Adott egy $n \times n$ elemből álló játékpálya, amely labirintusként épül fel, azaz fal, illetve padló mezők találhatók 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 alábbi 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 dialógusablakkal végezzük el a mentést, illetve betöltést, a fájlneveket a felhasználó adja meg.
- A felhasználói esetek ábrán láthatóak.

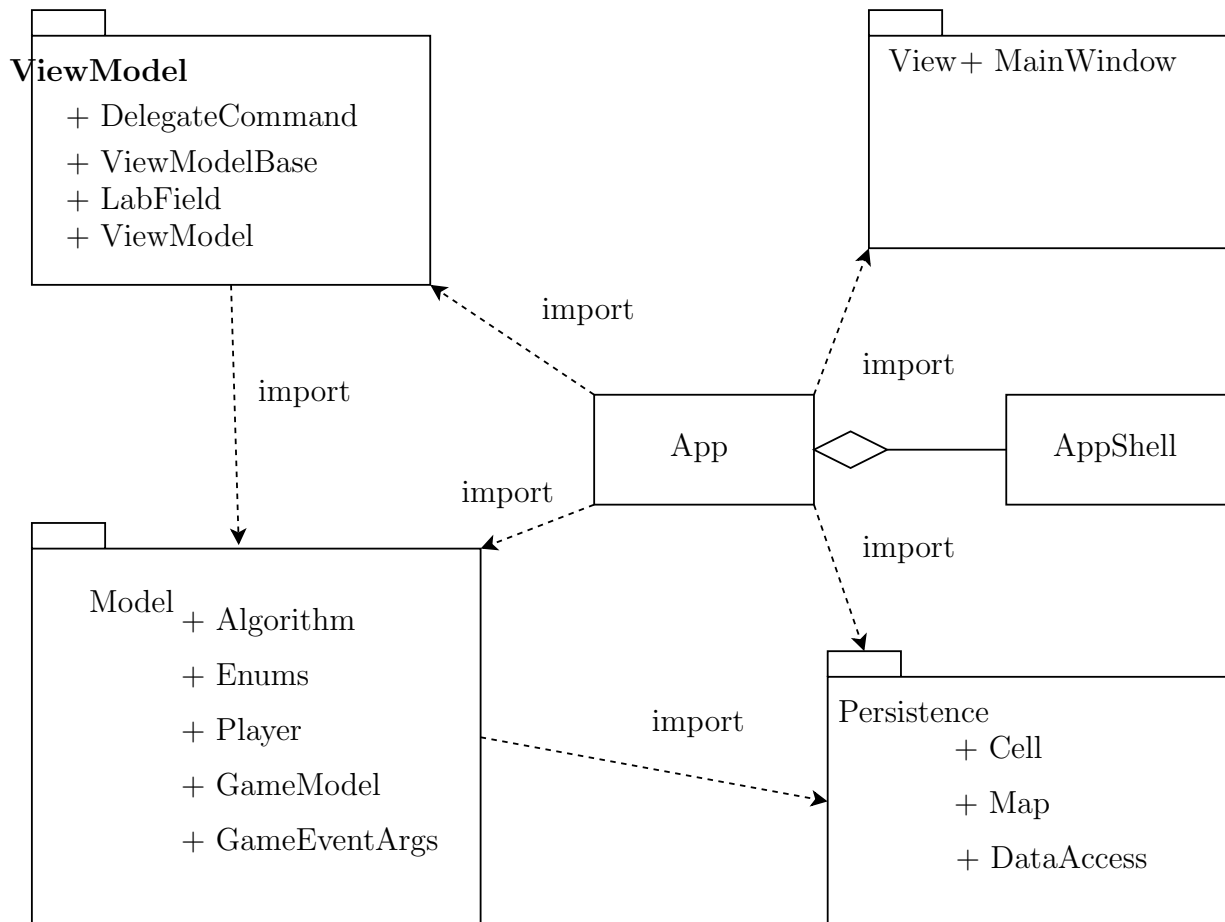
2.1 Felhasználói esetek diagramja



3 Tervezés

- Programszerkezet:
 - A programot MVVM architektúrában valósítjuk meg, ennek megfelelően View, Model, ViewModel és Persistence névttereket valósítunk meg az alkalmazáson belül. A program környezetét az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodellt 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 *DataAccess* 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.

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.

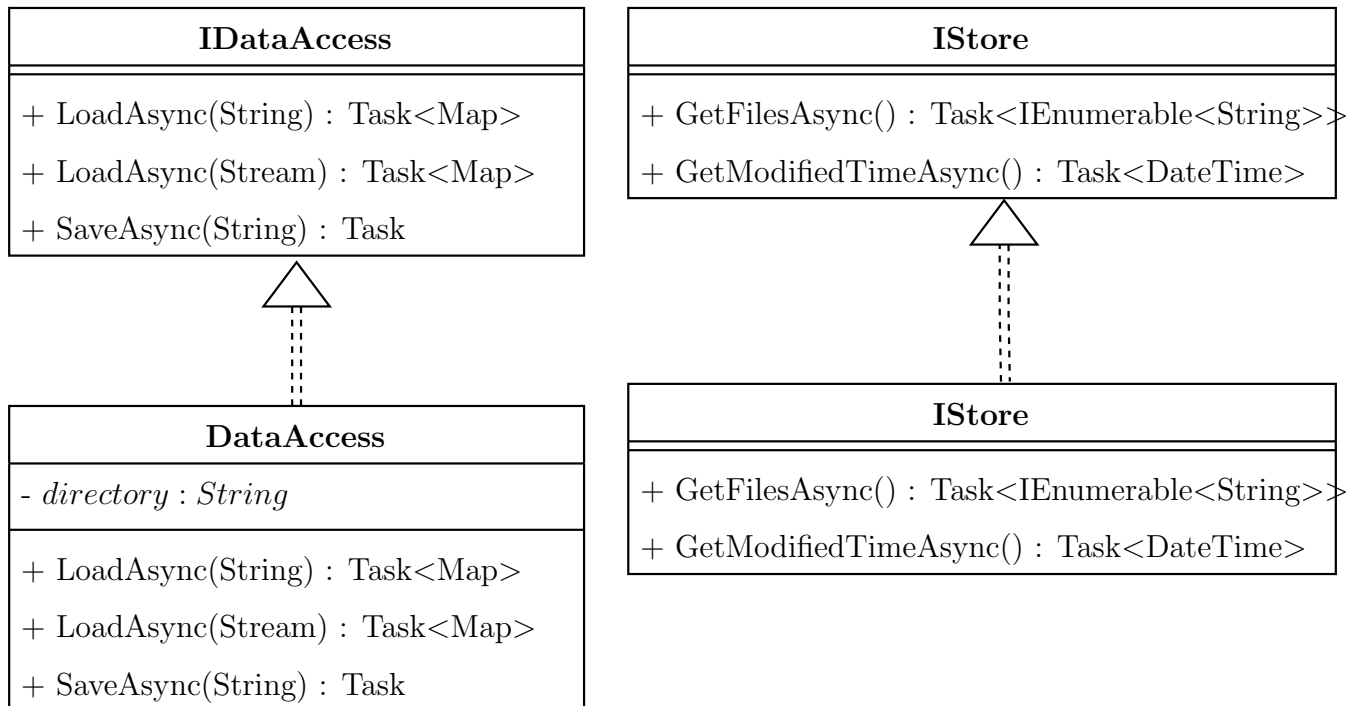
- A modell létrehozásakor a *DataAccess* osztály segítségével felpopuláljuk a labirintus *celláit*, majd létrehozuk a *Player* objektumot, ami valójában csak arra fog szolgálni, hogy eltárolja a jelenlegi pozíciót, amin a játékos áll. A *Map* valósítja meg a labirintus-t, ami egy egyszerű *Cell[,]* tömb. Annak érdekében, hogy a View, majd könnyebben tudja kezelni az új *Cell*-t, 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 rájöhethetünk milyen célt fognak szolgálni:
 - * *MapSize* (eltárolja a pálya méretét, pl. kicsi)
 - * *Arrow* (eltárolja az irányt 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 vezérőnek. A nézetmodell tárolja a modell egy hivatkozását, de csupán információkat kér le tőle, illetve a játéknehézséget szabályozza. Direkt nem avatkozik a játék futtatásába.
 - A játéklemező 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 játéklemezőt, a menüt és a státuszsort. A játéklemező egy ItemsControl vezérő, ahol dinamikusan felépítünk egy rácsot (UniformGrid), amely négyzetekből áll. Minden adatot adatkötéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a négyzetek színét is.
 - A fájlnev 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.

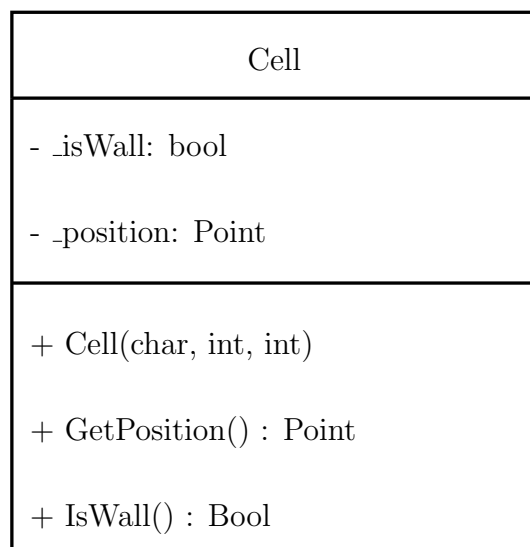
3.2 Statikus szerkezet

Az osztályok az alábbi módon épülnek fel:

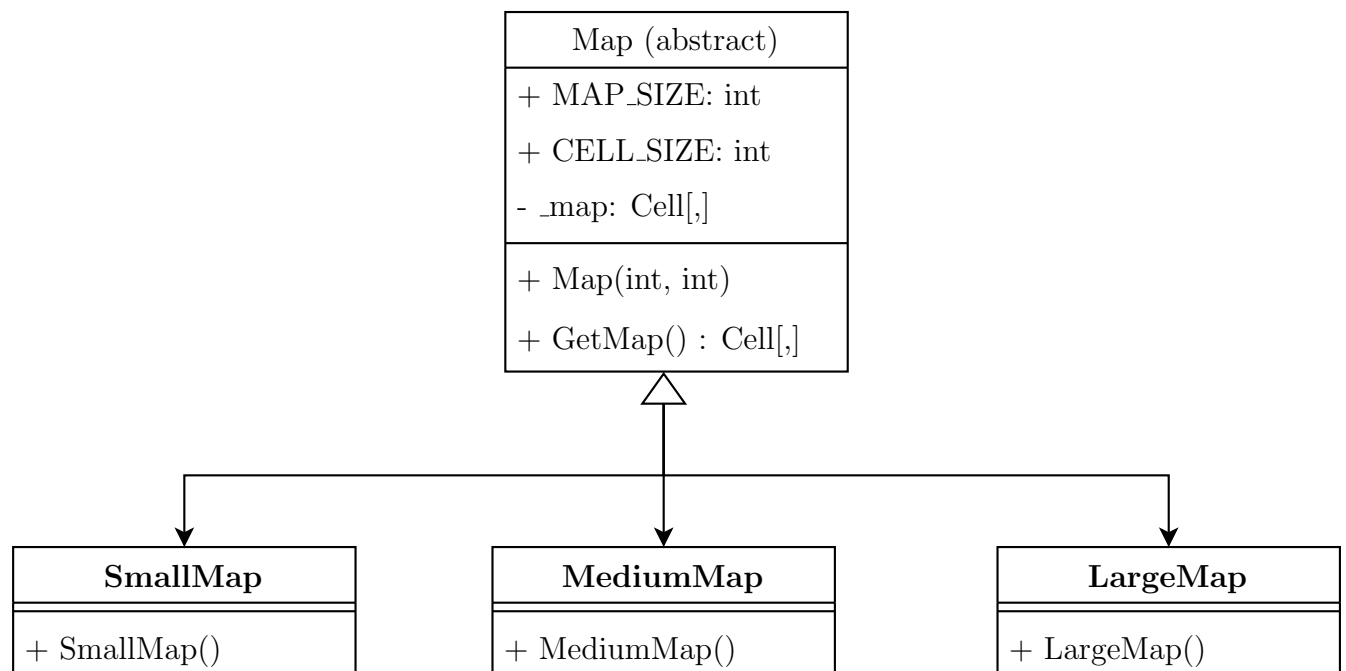
3.2.1 Adatelérés osztály



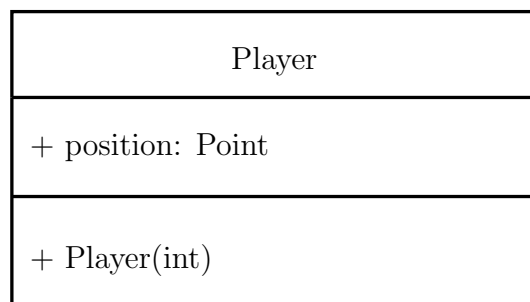
3.2.2 Cella osztály



3.2.3 Pálya osztály



3.2.4 Játékos osztály

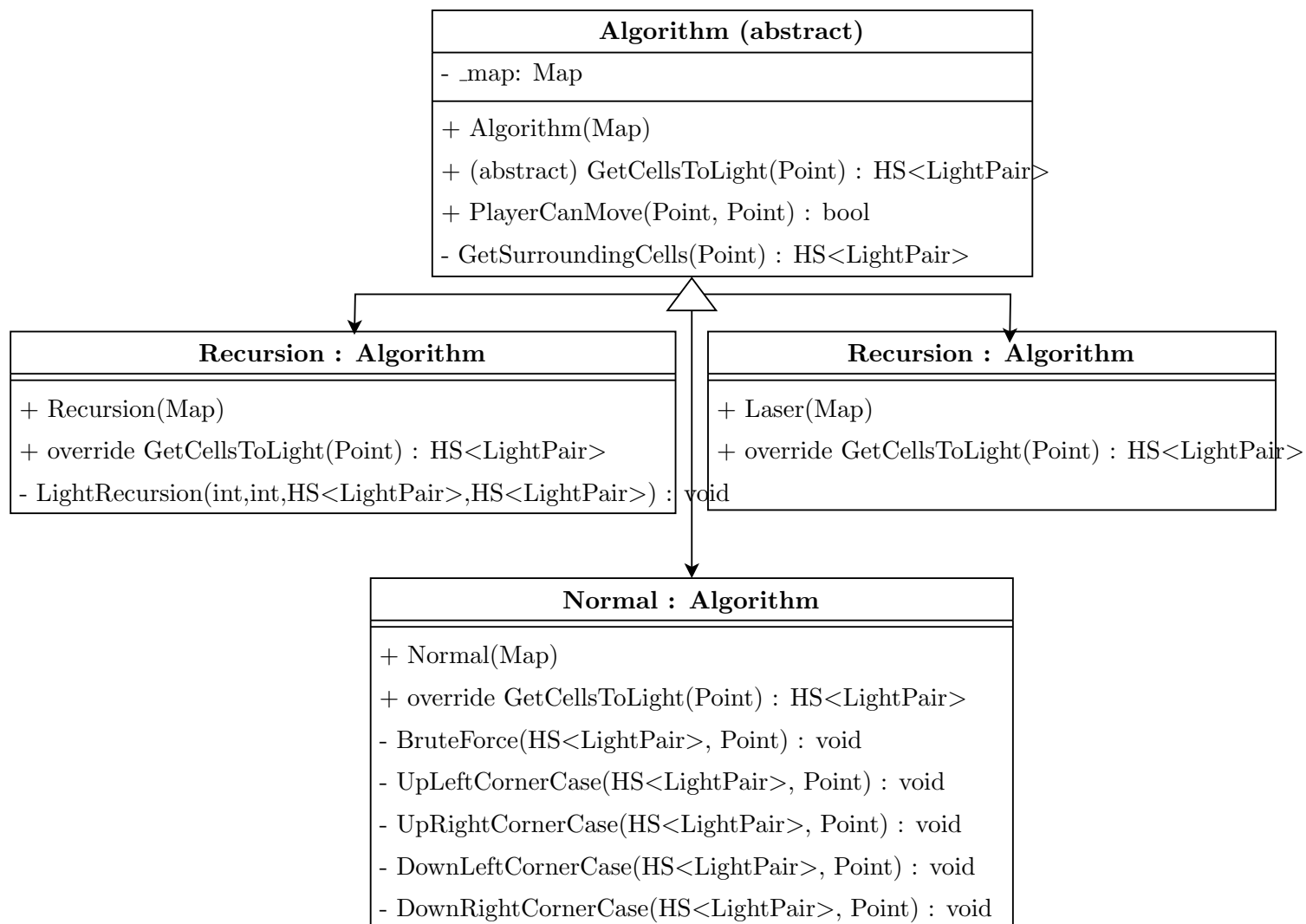


3.2.5 Új mentés osztályok

StoredGameViewModel
<ul style="list-style-type: none">+ Modified() : DateTime+ Name() : int+ LoadGameCommand() : DelegateCommand+ SaveGameCommand() : DelegateCommand

StoredGameBrowserViewModel
<ul style="list-style-type: none">- model : StoredGameBrowserModel
<ul style="list-style-type: none">+ StoredGameBrowserViewModel(StoredGameBrowserModel)- UpdateStoredGames() : void- ModelStoreChanged(object,EventArgs) : void+ GameLoading() : EventHandler<StoredGameEventArgs>+ GameSaving() : EventHandler<StoredGameEventArgs>+ StoredGames() : ObservableCollection<StoredGameViewModel>

3.2.6 Algoritmus osztály



3.2.7 Modell osztály

GameModel
<ul style="list-style-type: none">+ newGame: event<NewGameArgs>+ playerMoved: event<PlayerMovedArgs>+ playerWon: event<EventArgs>- _map: Map- _player: Player- _algo: Algorithm- _cellsToFree: HS<Point>- _gamemode: enum Gamemode
<ul style="list-style-type: none">+ 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
<ul style="list-style-type: none">- canExecute : Func<Object, Boolean> : {<i>readonly</i>}- execute : Action<Object> : {<i>readonly</i>}+ CanExecute(Object) : Boolean+ DelegateCommand(Action<Object>)+ DelegateCommand(Func<Object, Boolean>, Action<Object>)+ Execute(Object) : void+ RaiseCanExecuteChanged() : void+ CanExecuteChanged() : EventHandler

3.2.9 LabField osztály

LabField
<ul style="list-style-type: none">- isLit : Boolean- isPlayer : Boolean+ IsLit() : Boolean+ IsPlayer() : Boolean

3.2.10 ViewModelBase osztály

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

3.2.11 App osztály

App
- model : GameModel - dataAccess : IDataAccess - appShell : AppShell - store : IStore - viewModel : GameViewModel
+ App()

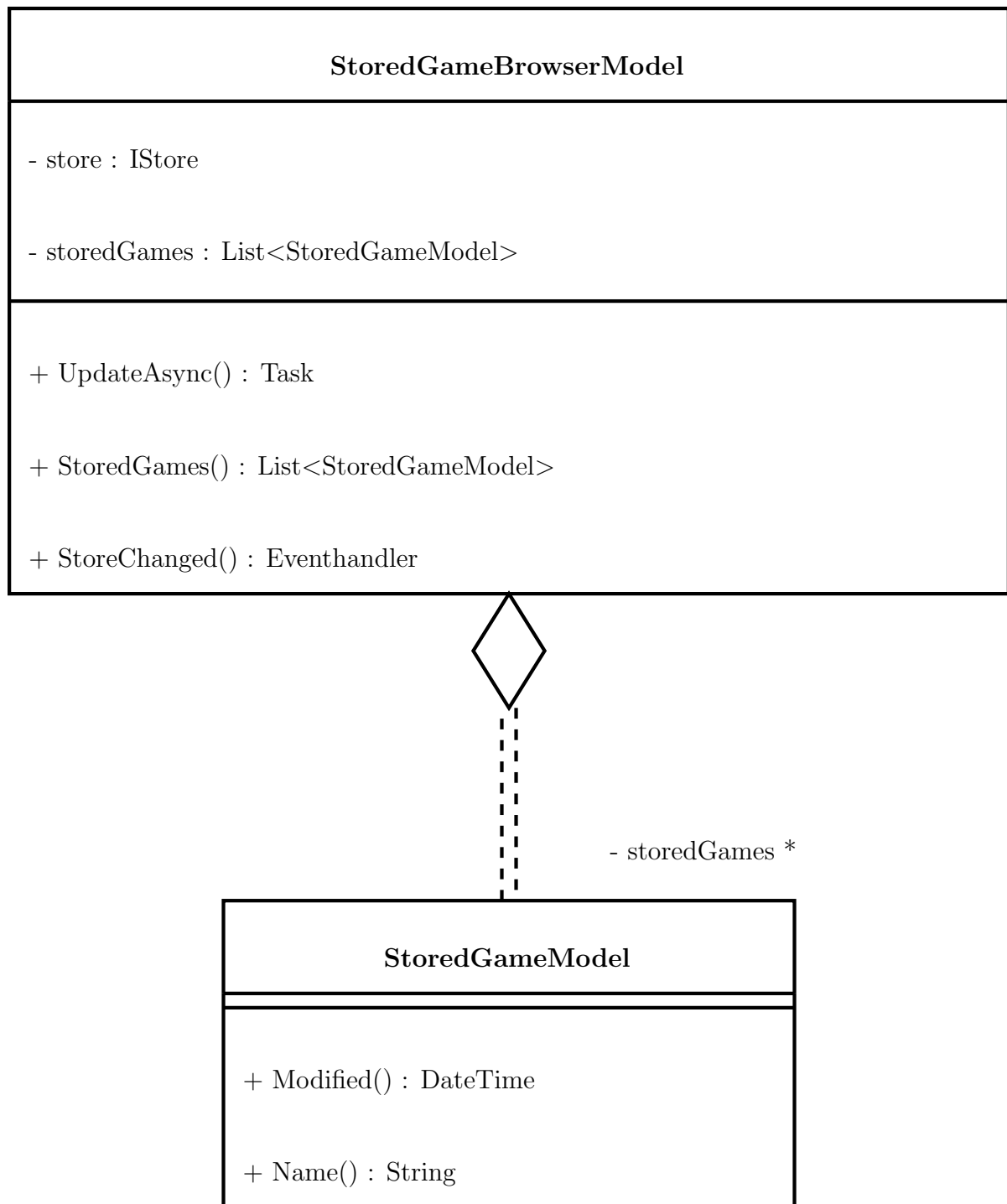
3.2.12 ViewModel osztály

ViewModel
<ul style="list-style-type: none">- 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

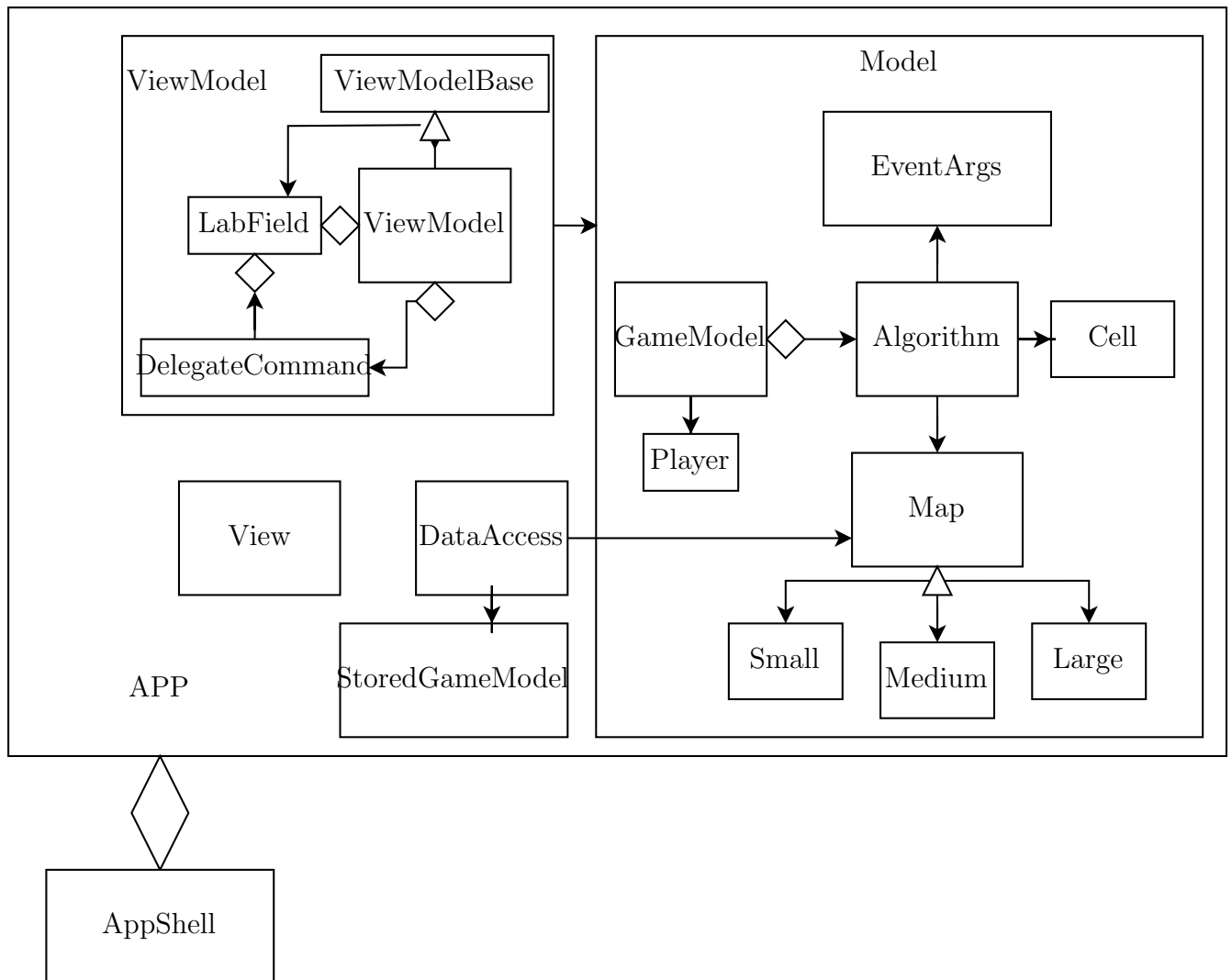
3.2.13 ViewModel osztály

AppShell
<ul style="list-style-type: none">- viewModel : GameViewModel- store : IStore- model : GameModel- storedGameBrowserM : StoredGameBrowserModel- storedGameBVM : StoredGameBrowserViewModel
<ul style="list-style-type: none">+ AppShell(IStore,GameModel,GameViewModel)+ ViewModelPlayerWon(object,EventArgs)+ ViewModelTimePaused(object,EventArgs)+ ViewModelNewGame(object,EventArgs)+ ViewModelExitGame(object,EventArgs)+ ViewModelLoadGame(object,EventArgs)+ ViewModelSaveGame(object,EventArgs)+ ViewModelGameLoading(object,EventArgs)+ ViewModelGameSaving(object,EventArgs)

3.2.14 StoredGame osztályok



3.3 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 *NullPointerException*, megfelel a pályaméret
- A játék kezdésekor a *Player* kezdőpozíciója a pálya bal alsó sarka
- A játékos mikor a pályán kívültre 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