Beadandó dokumentáció

4. feladatsor

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

Tartalom

1 Feladat

Labirintus

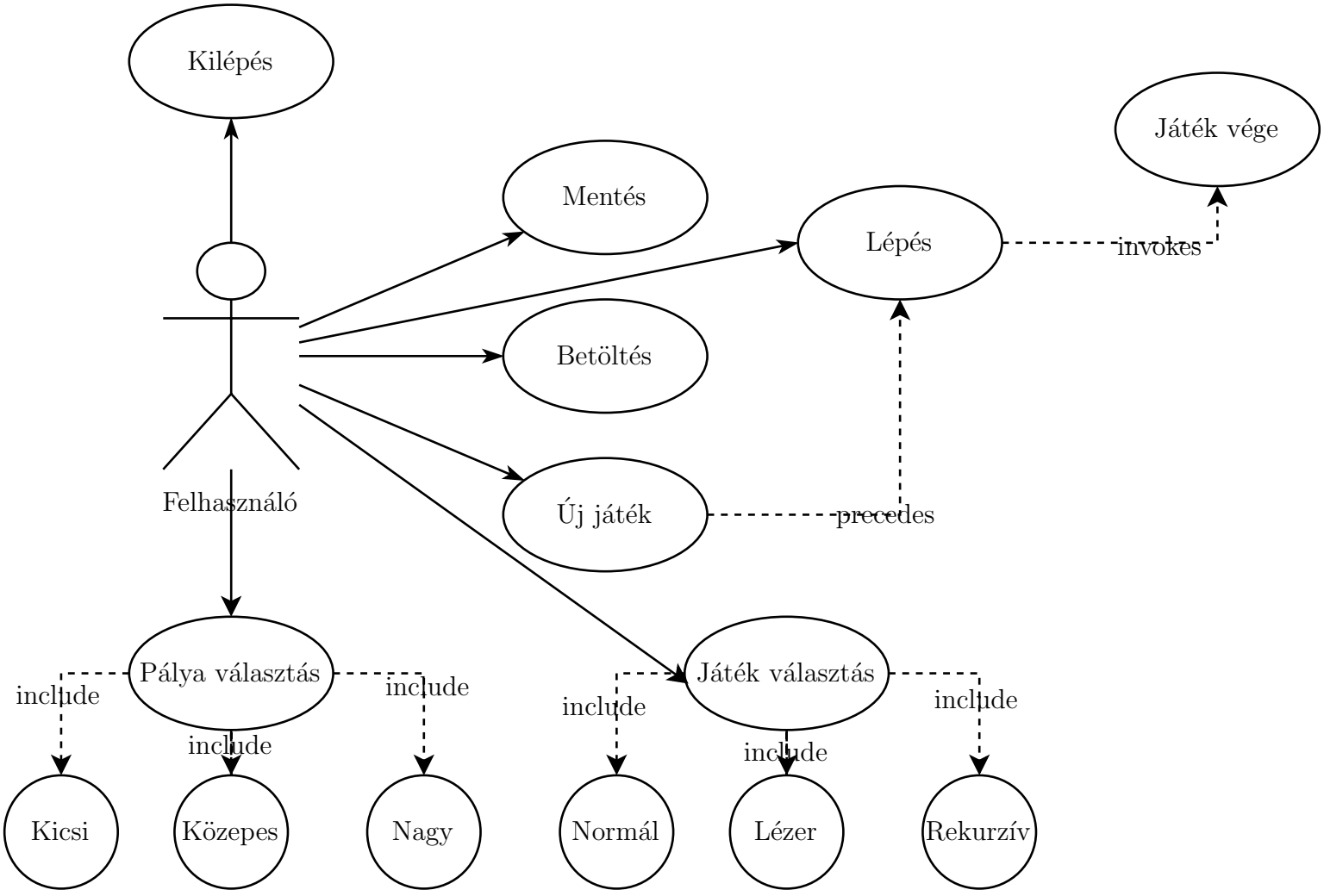
Készítsünk programot, amellyel a következő játékot játszhatjuk. Adott egy $n \times n$ elemből álló játékpálya, amely labirintusként épül fel, azaz fal, illetve padló mezők találhatóak benne, illetve egy kijárat a jobb felső sarokban. A játékos célja, hogy a bal alsó sarokból indulva minél előbb kijusson 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) és játékmódot tartalmaz (normál, lézer, rekurzív), ahol a normál játékmód felel meg a követelménynek, a maradék kettőt csak érdekességképpen raktam bele. A program indításakor a felhasználó látni fog egy menüt, ahol ki tudja választani a pálya és játékmód típusát.
- A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.
- A menüben az aláppi opciók vannak:
  1. Start
  2. Játék betöltése
  3. Kilépés
  4. Pályaméret választás
  5. Játékmód választás
- A labirintust az adott pályaméretnek (11×11, 21×21, 35×35) megfelelő számú *Panel* reprezentálja, szintúgy a játékost. A játékmódok csakis a pálya megvílágításában különböznek, az alapkoncepció ugyanaz: a játékos lát x darab *cellát* ami segít neki abban, hogy minél előbb kijusson a labirintusból (a jobb felső sarokban lévő *cellára* lépve). 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.

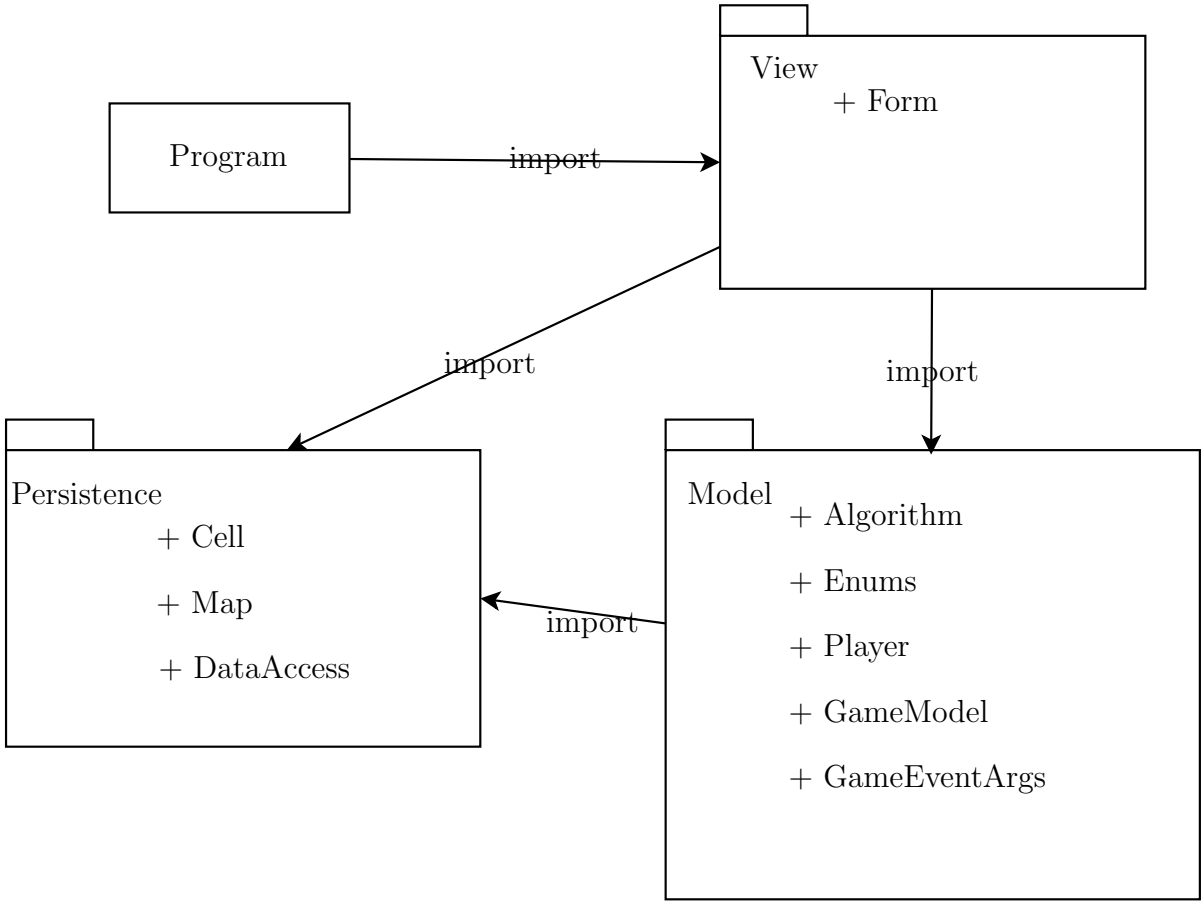
2.1 Felhasználói esetek diagramja



3 Tervezés

- Programszerkezet:
  - A programot 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 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 koordínátákat átadni.

– A jobb olvashatóság szempontjából az alábbi *enum* osztályokat hoztam létre, amiket a nevükből könnyedén rájöhetünk milyen célt fognak szolgálni:

  * *Gamemode* (eltárolja a játékmódot, pl. lézer)
  * *MapSize* (eltárolja a pálya méretét, pl. kicsi)
  * *Arrow* (eltárolja az irályt amerre a játékos lépni szeretne, pl. bal)

- Nézet:

  – A nézetet a *Form1* osztály biztosítja, amely tárolja a modell egy példányát (*game*). A *Form1* osztály hívja a *GameModel* osztály metódusait, hogy az adatelérés sikeresen megtörténjen.

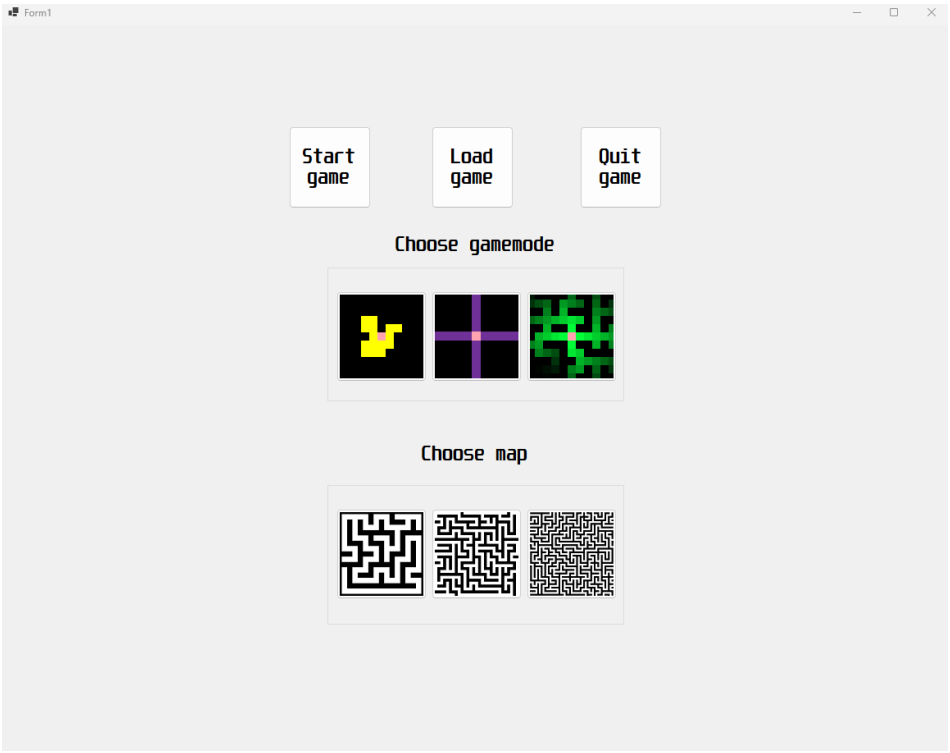
  – A nézet két fő részből áll:

    * Menü (itt tudja a felhasználó kiválasztani a játékmódot, pályát vagy egyszerűen betölteni a játékot)
    * Játék (labirintus és a játékos feldolgozása *Panel*-ek segítségével)

    Ezek közül a játék részét hozzuk létre dinamikusan, amikor a felhasználó elkezdett egy játékot.

  – A játék kijelzi az eltelt időt, amit a nézetben egy szövegdoboz segítségével valósítunk meg.

3.1.1 Menü tényleges kinézete



3.1.2 Játék tényleges kinézete

3.2 Statikus szerkezet

Az osztályok az alábbi módon épülnek fel:

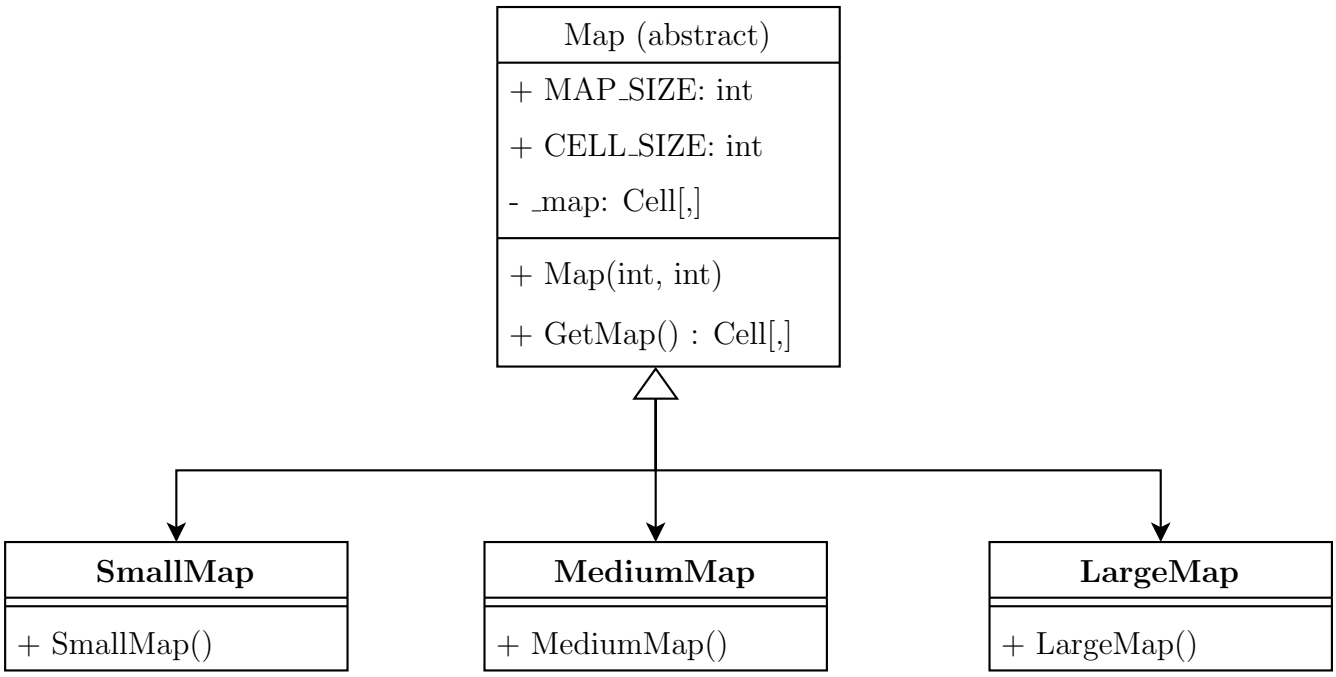
3.2.1 Adatelérés osztály

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

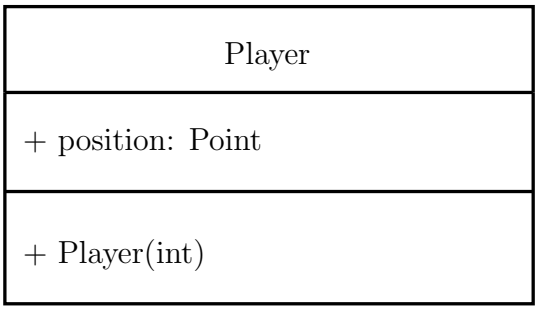
3.2.2 Cella osztály

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

3.2.3 Pálya osztály

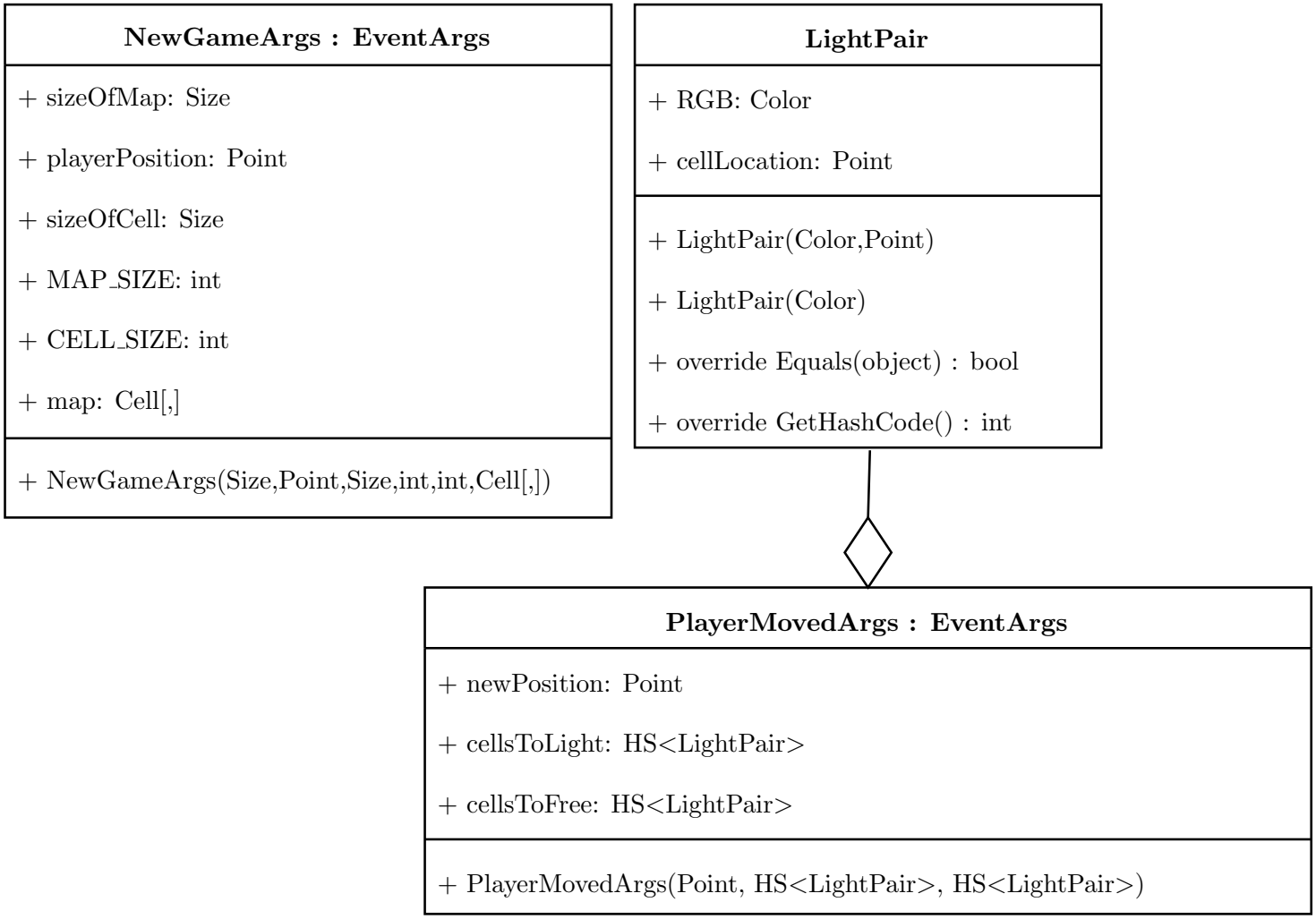


3.2.4 Játékos osztály

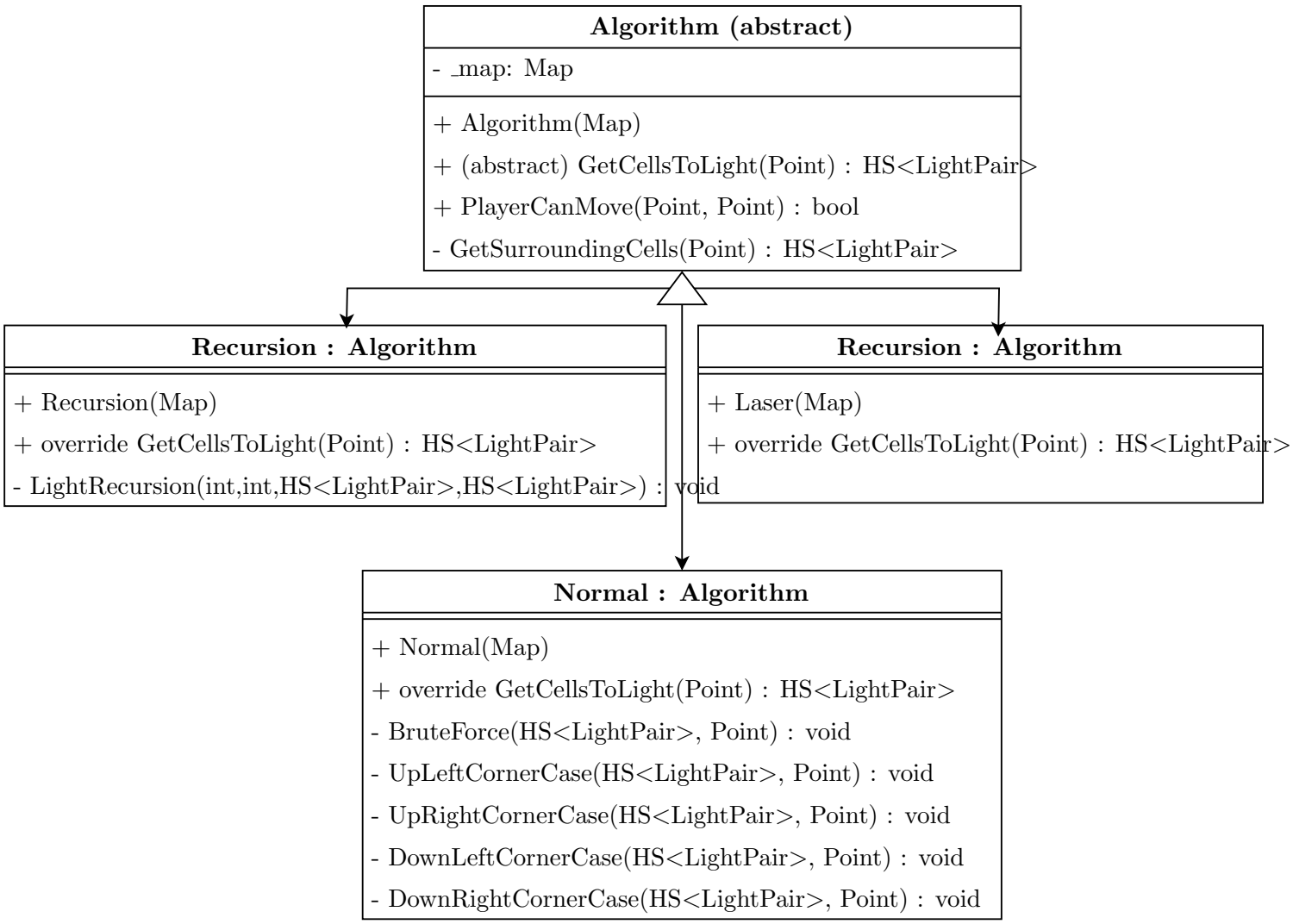


3.2.5 Esemény argumentum osztály

A "HS" rövidítése a HashSet típusnak.



3.2.6 Algoritmus osztály



3.2.7 Modell osztály

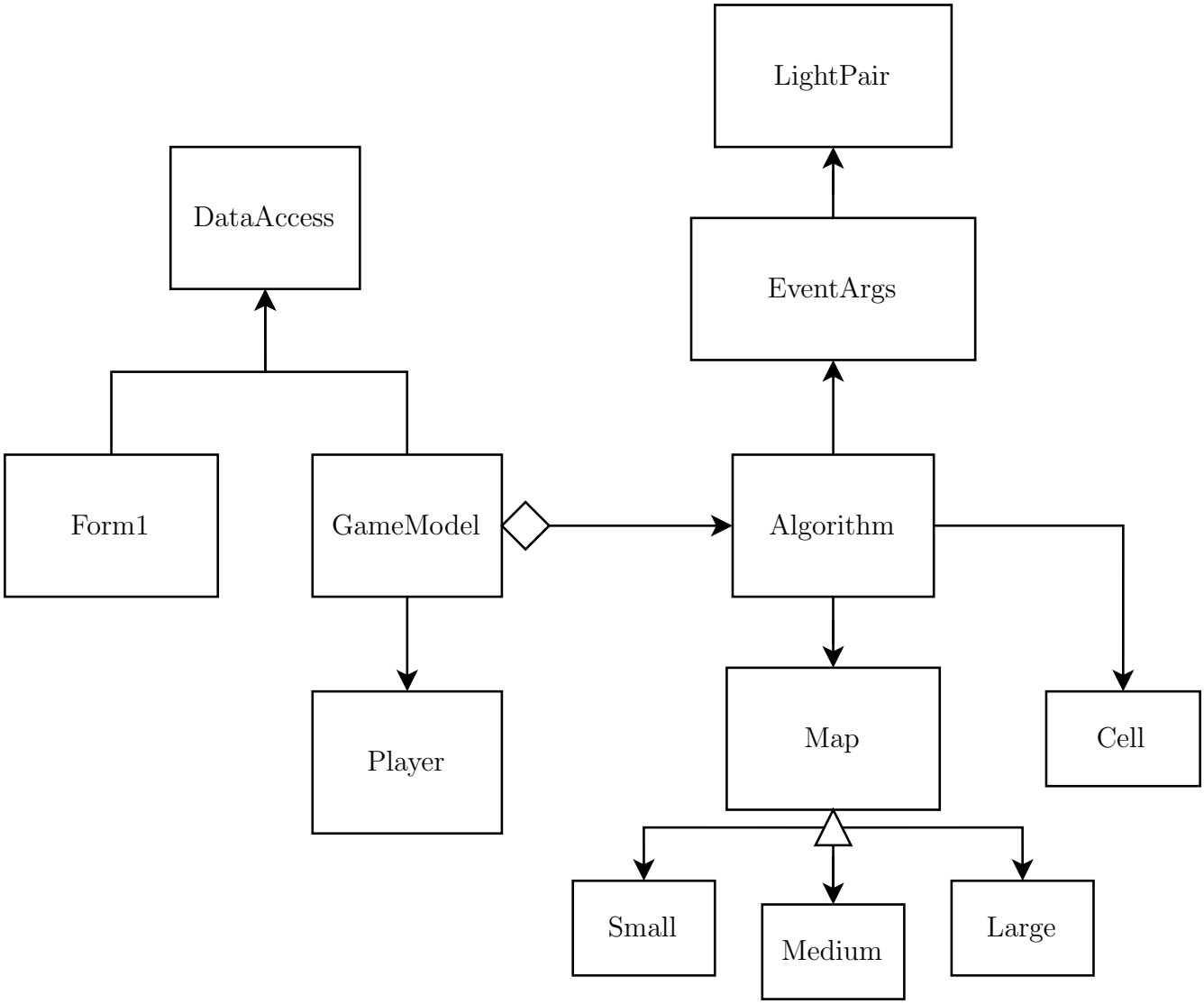
GameModel
+ newGame: event<NewGameArgs>
+ playerMoved: event<PlayerMovedArgs>
+ playerWon: event<EventArgs>
- _map: Map
- _player: Player
- _algo: Algorithm
- _cellsToFree: HS<Point>
- _gamemode: enum Gamemode
+ GameModel()
+ StartNewGame(enum MapSize, enum Gamemode) : void
+ LoadNewGame(string) : void
- 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 Nézet osztály

Azokat a almetúdusokat kihagytam, amik lényegi funkciót nem töltenek be, szimplán "szebbé" teszik a kódot. Pont ezek azok az almetódusok, amiket ezen osztálydiagramon lévők hívnak.

Form1 : Form
- _game: GameModel - _background: Panel - _player: Panel - _cells: Panel[,] - _timer: Timer - _elapsedTime: TimeSpan - _elapsedTimeLabel: Label - _saveGameButton: Button - _backToMenuButton: Button - _pauseTimeButton: CheckBox
+ Form1() - FormLoadEventHandler(object, EventArgs) : void - StartHitEventHandler(object, EventArgs) : void - QuitHitEventHandler(object, EventArgs) : void - LoadGameEventHandler(object, EventArgs) : void - Game_GameStarted__SettingUpPanels(object, NewGameArgs) : void - KeyHitEventHandler(object, KeyEventArgs) : void - TimerTickEventHandler(object,EventArgs) : void - Game_PlayerMoved__ActivateLights(object, PlayerMovedArgs) : void - Game_PlayerWon__ShowDialogWindow(object, EventArgs) : void - BackToMenu() : void - SaveGameEventHandler(object, EventArgs) : void - BackToMenuEventHandler(object, EventArgs) : void - PauseTimeEventHandler(object, EventArgs) : void

3.3 Egész osztálydiagram egyszerű képe



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, játékmódot)
- A pálya betöltése sikeresen megtörténik