2. Beadandó feladat

Készítette:

Név: Kékesi Ádám
Neptun-kód: DPTEH5
Email: kekesi.adam2003@gmail.com

Feladat:

Készítsünk programot, amellyel a következő játékot játszhatjuk.

Adott egy $n \times n$ mezőből álló játékpálya, amelyeken falak, illetve járható mezők helyezkednek el, valamint ellenfelek járőröznek. A játékos célja, hogy ellenfeleit bombák segítségével minél gyorsabban legyőzze.

Az ellenfelek adott időközönként lépnek egy mezőt (vízszintesen, vagy függőlegesen) úgy, hogy folyamatosan előre haladnak egészen addig, amíg falba nem ütköznek. Ekkor véletlenszerűen választanak egy új irányt, és arra haladnak tovább.

A játékos figurája kezdetben a bal felső sarokban helyezkedik el, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán, de ha találkozik (egy pozíciót foglal el) valamely ellenféllel, akkor meghal.

A játékos bombát rakhat le az aktuális pozíciójára, amely rövid időn belül robban megsemmisítve a 3 sugáron belül (azaz egy 7×7-es négyzetben) található ellenfeleket (falon át is), illetve magát a játékost is, ha nem menekül onnan.

A pályák méretét, illetve felépítését (falak, ellenfelek kezdőpozíciója) tároljuk fájlban. A program legalább 3 különböző méretű pályát tartalmazzon.

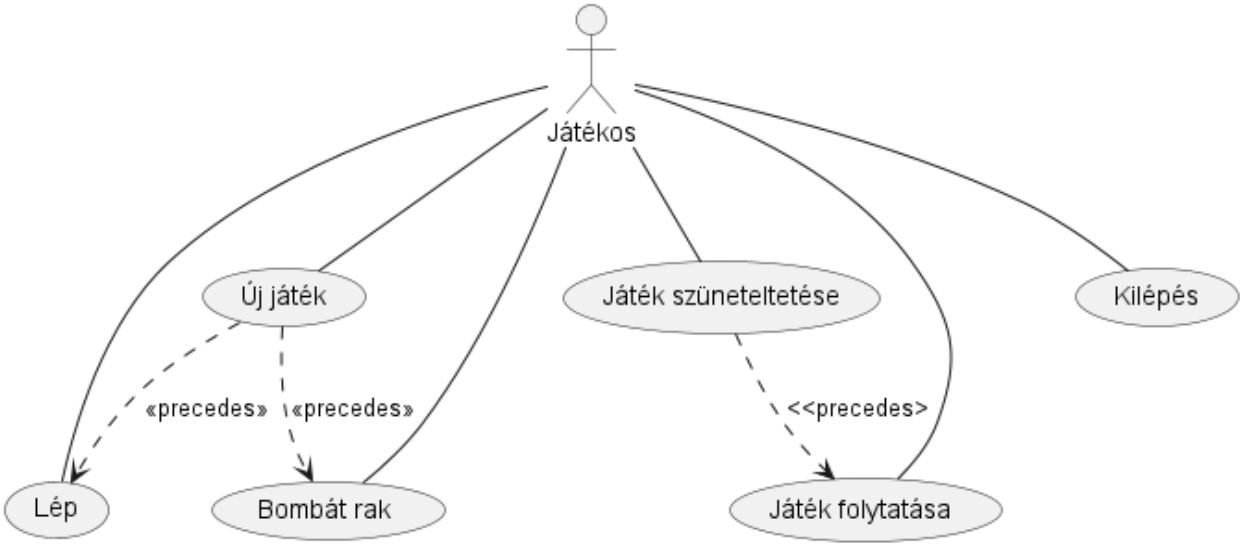
A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet a játékos). Ismerje fel, ha vége a játéknak, és jelezze, győzött, vagy veszített a játékos. A program játék közben folyamatosan jelezze ki a játékidőt, valamint a felrobbantott ellenfelek számát.

Elemzés:

- A játék pályáját egy fájlból tölthetjük be, amely tartalmazza, hogy hol helyezkednek el az ellenfelek, falak, járható mezők. A pálya $n \times n$-es, a betöltött fájl alapjáb

- A feladatot .NET Avalonia alkalmazásként, elsődlegesen asztali alkalmazásként és Android platformon valósítjuk meg, amely egy fő nézetből fog állni.
- Az ablakban elhelyezünk egy menüt a következő menüpontokkal: File (Choose game map). Az ablak alján megjelenítünk egy státuszsort, amely a megölt ellenfelek számát, illetve az eltelt időt jelzi. Ugyanitt lehet a játékot szüneteltetni, elindítani.
- A játéktáblát egy $n \times n$ képekből álló rács reprezentálja, amelyen az adott mező tartalmát lehet látni (Üres, Játékos, Ellenség, Fal, Bomba)
- Az ellenfeleket a játék automatikusan lépteti, adott időközönként
- A játékost a WASD gombokkal lehet léptetni
- Bombát a Space-cel lehet lerakni
- A lerakott bombák automatikusan felrobbannak egy megadott idő után, megsemmisítve ellenfelet és játékost egyaránt egy 7x7-es területen
- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (meghalt a játékos, vagy megölte az összes ellenfelet). Szintén dialógusablakkal végezzük el a játékpálya betöltését.

Felhasználó eset diagram:



Tervezés:

- Programszerkezet:
  - A szoftvert négy projektből építjük fel: a modellt és a perzisztenciát tartalmazó osztálykönyvtárból (.NET Standard Class Library), valamint a .NET Avalonia projektjeiből (platformfüggetlen osztálykönyvtár és platformfüggő

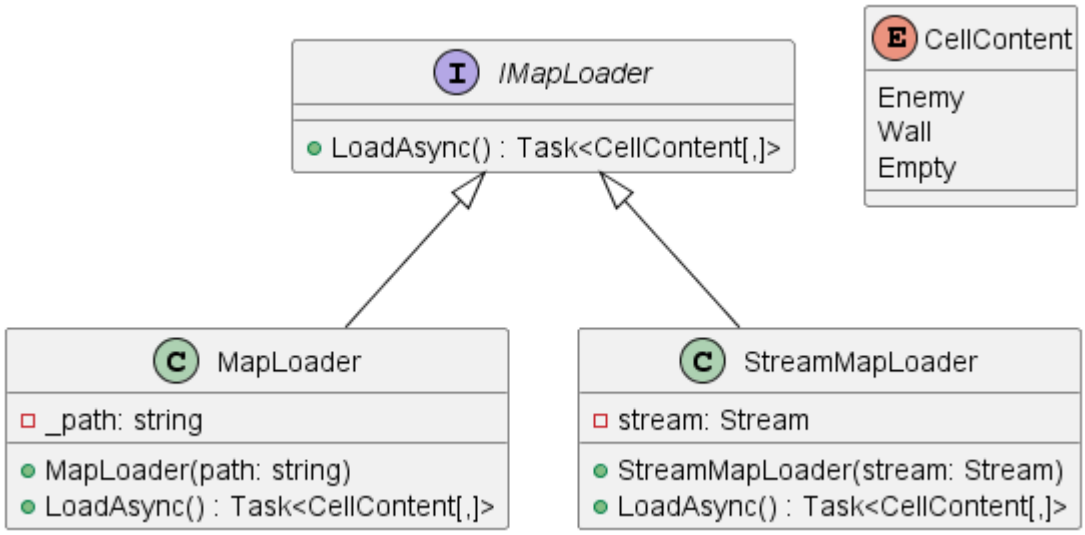
végrehajtható projektek), amelyet így Windows és Android operációs rendszerekre is le tudunk fordítani.

    - A programot MVVM architektúrában valósítjuk meg, ennek megfelelően View, Model, ViewModel és Persistence névtereket valósítunk meg az alkalmazáson belül.
    - A program vezérlését az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést. Az alkalmazás környezeti réteg feladata a platformfüggő alkalmazás életciklus helyénvaló kezelése is.
    - A program szerkezetét két projektre osztjuk implementációs megfontolásból: a Persistence és Model csomagok a program ViewModel és View csomagok a WPF függő projektjében kapnak helyet.
- Perzisztencia:
    - Az adatkezelés feladata a kezdeti játékpálya betöltése
    - A játékpályát egy **CellContent[,]** matrix reprezentálja. A **CellContent** enum az **Enemy**, **Wall**, **Empty** értékeket tartalmazza
    - Az **IMapLoader** interfész írja le a játékpályát betölteni képes osztályokat. Ennek az interfésznek jelenleg két megvalósítása van, a **MapLoader**, amely .txt fájlból tölti be a játékpályát, és a **StreamMapLoader**, amely egy Stream-ből tölti be a pályát
    - A pálya fájl a 0,1,2 karakterekből áll, egy sora a játékpálya egy sorát jelenti. A 0,1,2 karakterek az Enemy, Wall, Empty enum értékeknek felelnek meg
    - Ha a nem megfelelő formátumú pályát töltenénk be, a **MapLoader** **IncorrectMapFileException** kivételt dob
- Modell:
    - A modell rétegben a **Player**, **Enemy**, és a **Wall** osztályok reprezentálják a játékpálya mezőit
    - Ezek az osztályok implementálják az **IField** interfészt, amelyben az ütközéseket lekezelő **OnCollision** metódus van leírva
    - A mozgásra képes, “halandó” játékmezők ősosztálya a **Unit** absztrakt osztály, amely szintén implementálja az **IField**-et. A **Unit** ősosztály kezeli a mozgás, és az elhalálozás logikáját. A halál a Died eseményt váltja ki. A mozgást az egység a vele asszociációban lévő IMap-el valósítja meg.
    - A **Player** osztály a **Unit** funkcionalitásán túl a bomba lerakását kezelő **PlantBomb** metódust tartalmazza. Az **IField**-ben lévő **OnCollision** metódust üres metódussal implementálja, nem határoz meg játékos-specifikus funkcionalitást más mezővel történő ütközés esetére

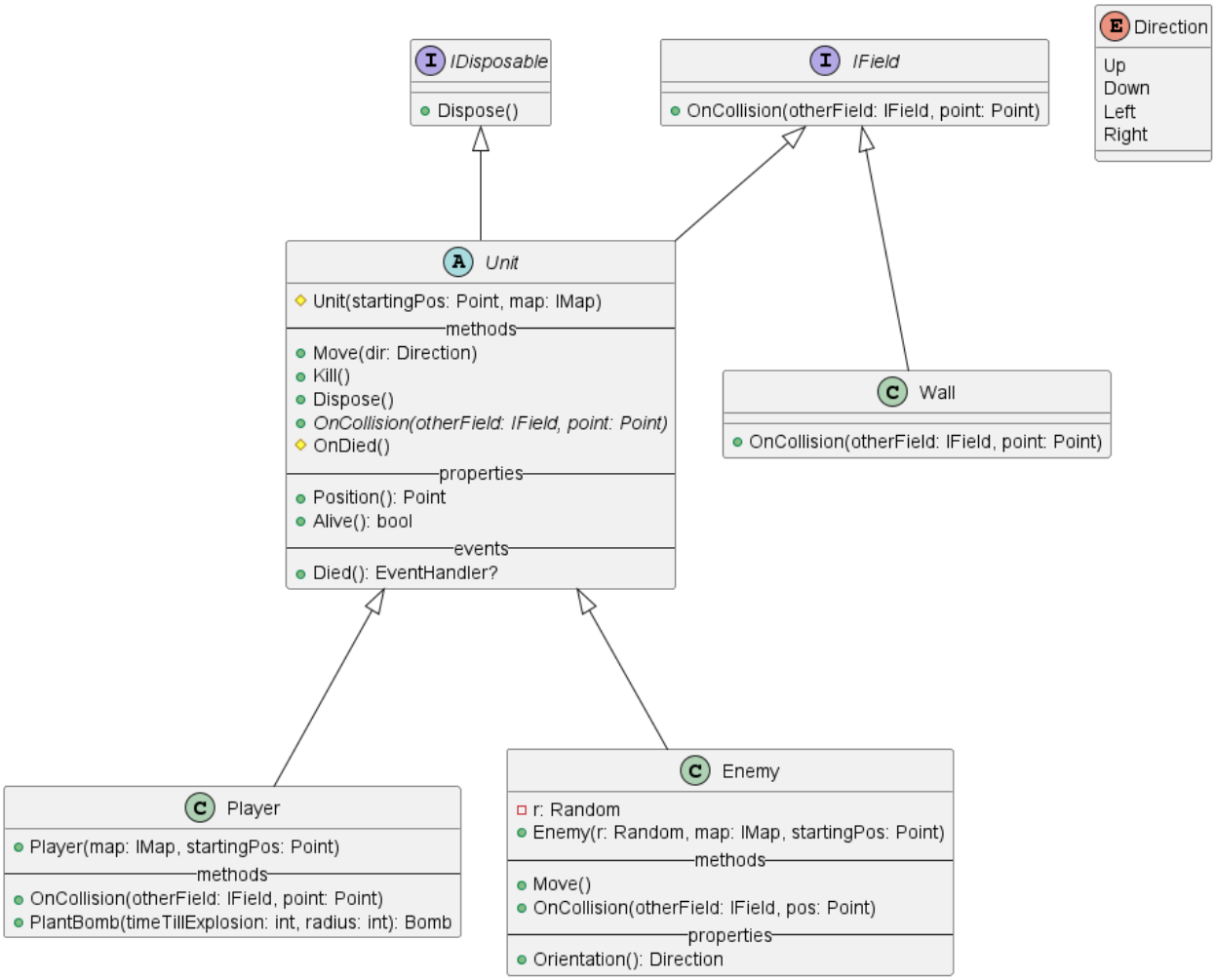
    - Az **Enemy** osztály a **Unit** funkcionalitásán túl kezeli az ellenségek véletlenszerű irányváltoztatását falnak ütközéskor, és a játékos megölését találkozáskor. Ezeket a funkciókat **OnCollison** metódusban valósítja meg.
    - A **Map** osztály felel a mozgások megvalósításáért a 2D-s táblán. Kiszámolja az adott irányba történő lépés utáni célmezőt, és kezeli az ütközést a lépő mező, és a cél mező között. A bomba robbanáshoz segédmetódust biztosít **ForEachInArea** néven, amely egy adott középpont körüli radius sugarú területen járja be a mezőket.
    - A **BombCollection** osztály felel a jelenleg elhelyezett bombák nyilvántartásáért. A **BombsChanged** eseményt váltja ki, ha új bomba került a pályára, vagy pedig felrobbant egy. Bomba robbanásakor a vele asszociációban lévő IMap-től lekérdezi az érintett mezőket, és az érintett Unit-okat megöli
    - A **Game** osztály valósítja meg a különböző játékelemek közötti interakciókat. Fogadja a játékos parancsait, a **PlayerStep**, **PlantBomb**, **PauseToggle** metódusaival. Kezeli az eltelt idő és a megölt ellenfelek számának nyilvántartartását, amelyek változásáról eseményeket is vált ki, továbbá ha vége lenne a játéknak, a **GameOver** eseményt váltja ki.
- Nézetmodell:
    - A nézetmodell megvalósításához felhasználjuk az *MVVM Toolkit* csomagból elérhető általános utasítás (**RelayCommand**), valamint egy ős változásjelző (**ObservableObject**) osztályt.
    - A nézetmodell feladatait a **BomberViewModel** osztály látja el, amely parancsokat biztosít új játék kezdéséhez, mozgáshoz, bomba lerakáshoz, és a játék szüneteltetéséhez. A parancsokhoz eseményeket kötünk, amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja a modell egy hivatkozását (**model**), és a parancsok hatására interakcióba lép vele.
    - A játékmező számára egy ViewModelt biztosítunk (**CellViewModel**), amely eltárolja a pozíciót, adott cella tartalmát, és hogy van-e bomba lerakva a cellára. A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (**Cells**).

- Nézet:
    - A nézetet ablakok, egyedi vezérlők és dialógusablakok használatával valósítjuk meg.
    - A **MainView** osztály, mint **UserControl** leszármazott tartalmazza a játéktáblát, amelyet egy **Grid** segítségével valósítunk meg.

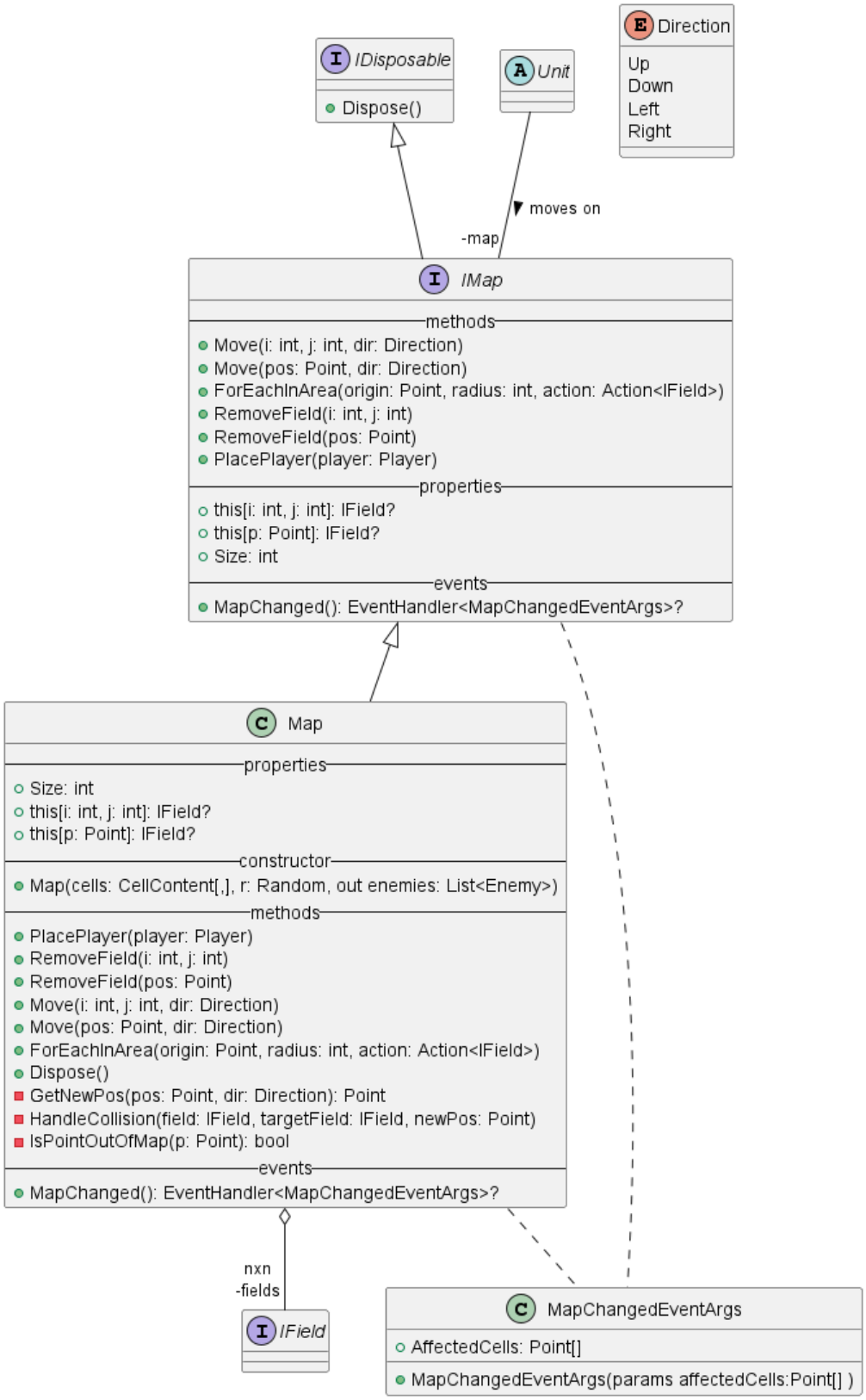
    - A **MainWindow** ablakba egyszerűen a **MainView** vezérlőt ágyazzuk be. Ilyen módon a felület asztali alkalmazásokban ablakos alkalmazásként, mobil platformon pedig lapként is megjeleníthető.
- Vezérlés:
    - Az App osztály feladata az alkalmazás vezérlése, a rétegek példányosítása és az események feldolgozása.
    - A OnFrameworkInitializationCompleted metódus felüldefiniálásával kezeljük a nézet platform specifikus megjelenítését.
    - A játék végét jelző, és a játékpálya betöltésére szolgáló dialógusablakokat is innen indítjuk.
- Az osztálydiagram:
  Perzisztencia:



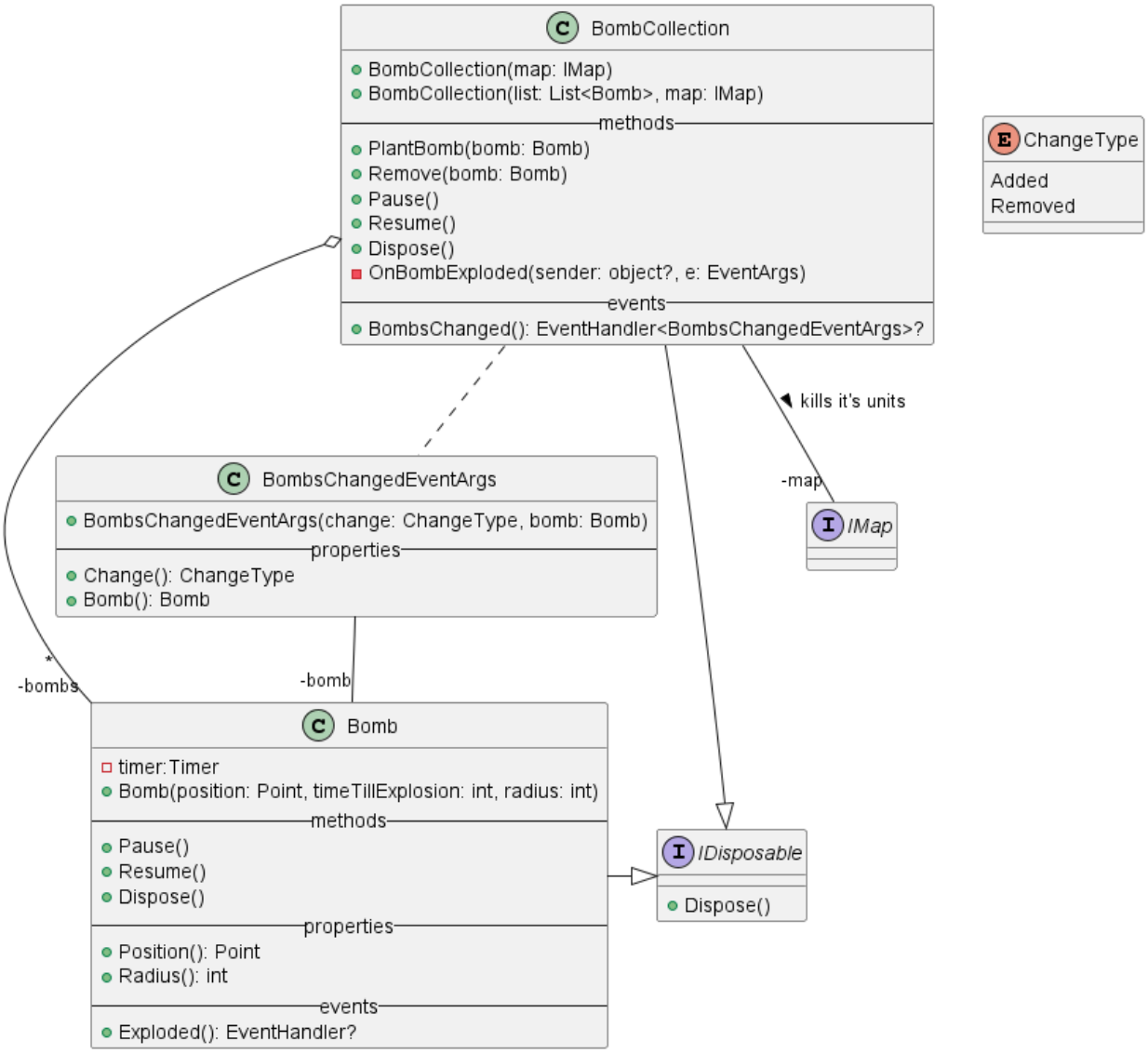
IField és leszármazottai:



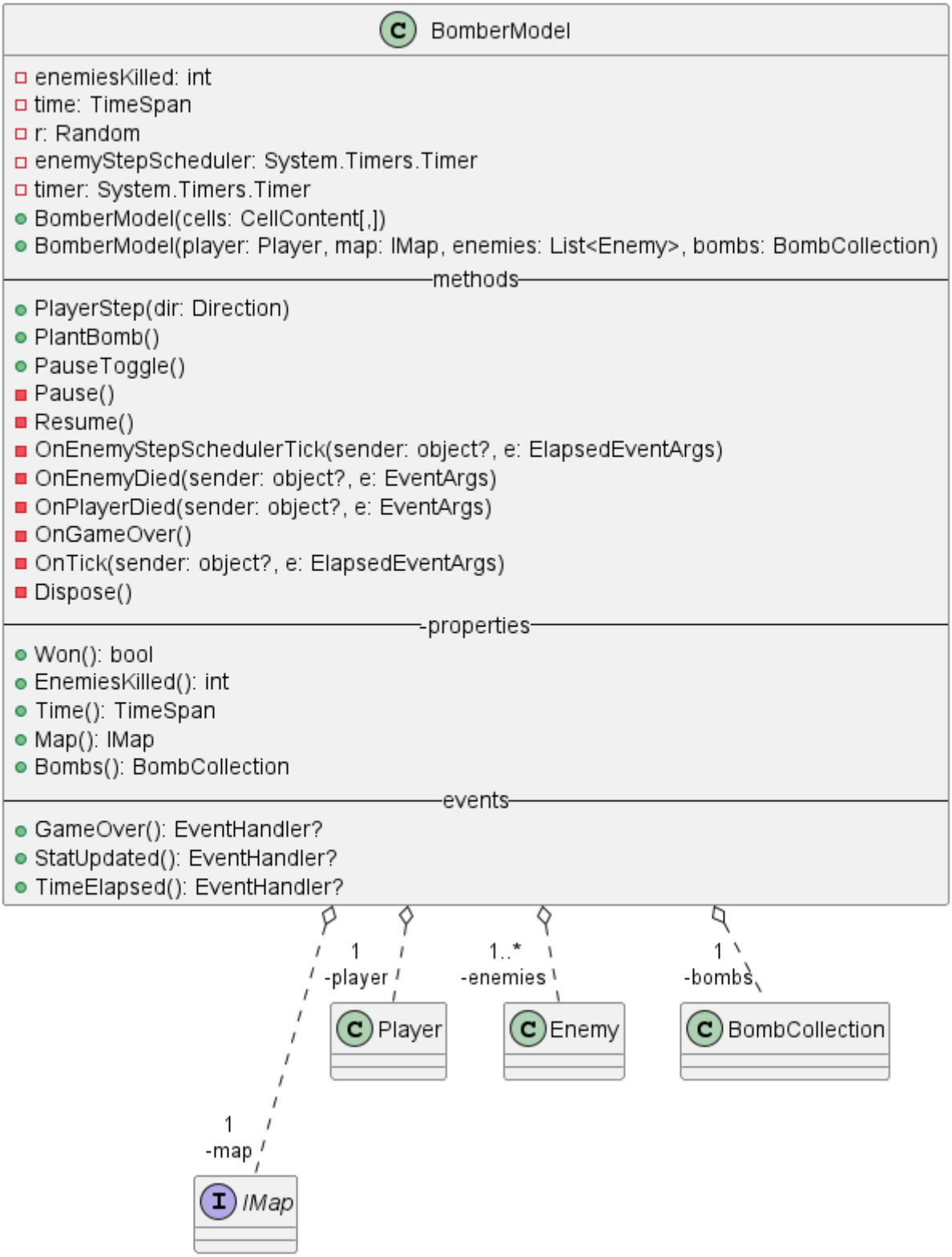
Map:


IDisposable
Dispose()
Unit
Direction
Up
Down
Left
Right
moves on
-map
IMap
methods
Move(i: int, j: int, dir: Direction)
Move(pos: Point, dir: Direction)
ForEachInArea(origin: Point, radius: int, action: Action<IField>)
RemoveField(i: int, j: int)
RemoveField(pos: Point)
PlacePlayer(player: Player)
properties
this[i: int, j: int]: IField?
this[p: Point]: IField?
Size: int
events
MapChanged(): EventHandler<MapChangedEventArgs>?
Map
properties
Size: int
this[i: int, j: int]: IField?
this[p: Point]: IField?
constructor
Map(cells: CellContent[,], r: Random, out enemies: List<Enemy>)
methods
PlacePlayer(player: Player)
RemoveField(i: int, j: int)
RemoveField(pos: Point)
Move(i: int, j: int, dir: Direction)
Move(pos: Point, dir: Direction)
ForEachInArea(origin: Point, radius: int, action: Action<IField>)
Dispose()
GetNewPos(pos: Point, dir: Direction): Point
HandleCollision(field: IField, targetField: IField, newPos: Point)
IsPointOutOfMap(p: Point): bool
events
MapChanged(): EventHandler<MapChangedEventArgs>?
nxn
-fields
IField
MapChangedEventArgs
AffectedCells: Point[]
MapChangedEventArgs(params affectedCells:Point[])

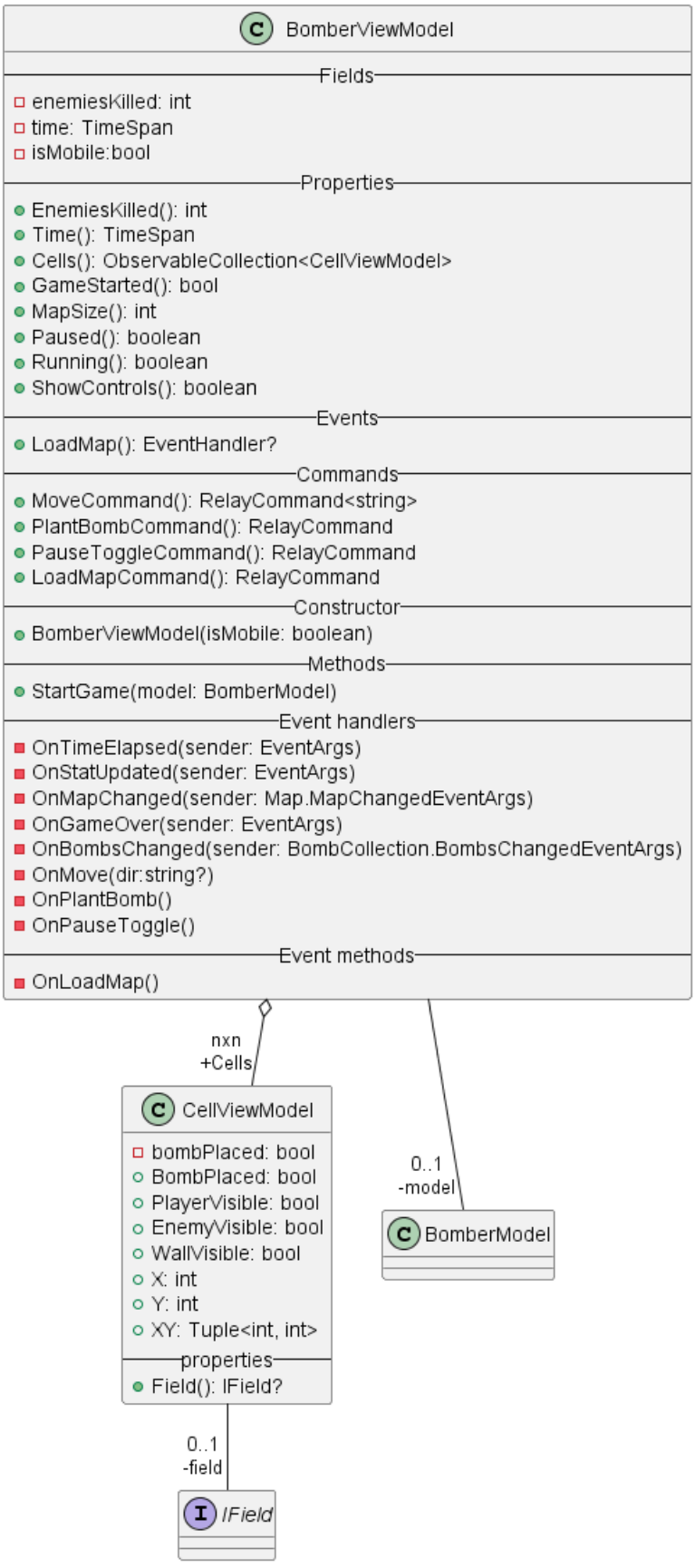
Bombák, BombCollection:



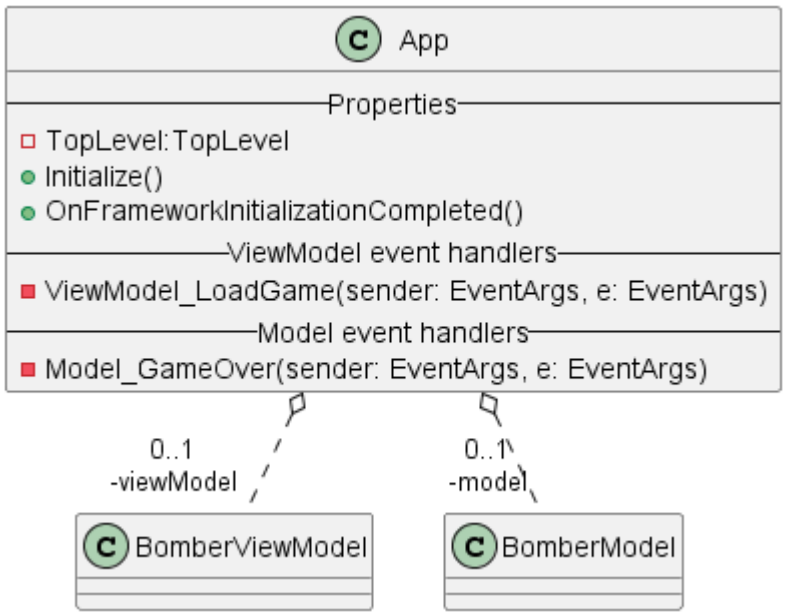
BomberModel:



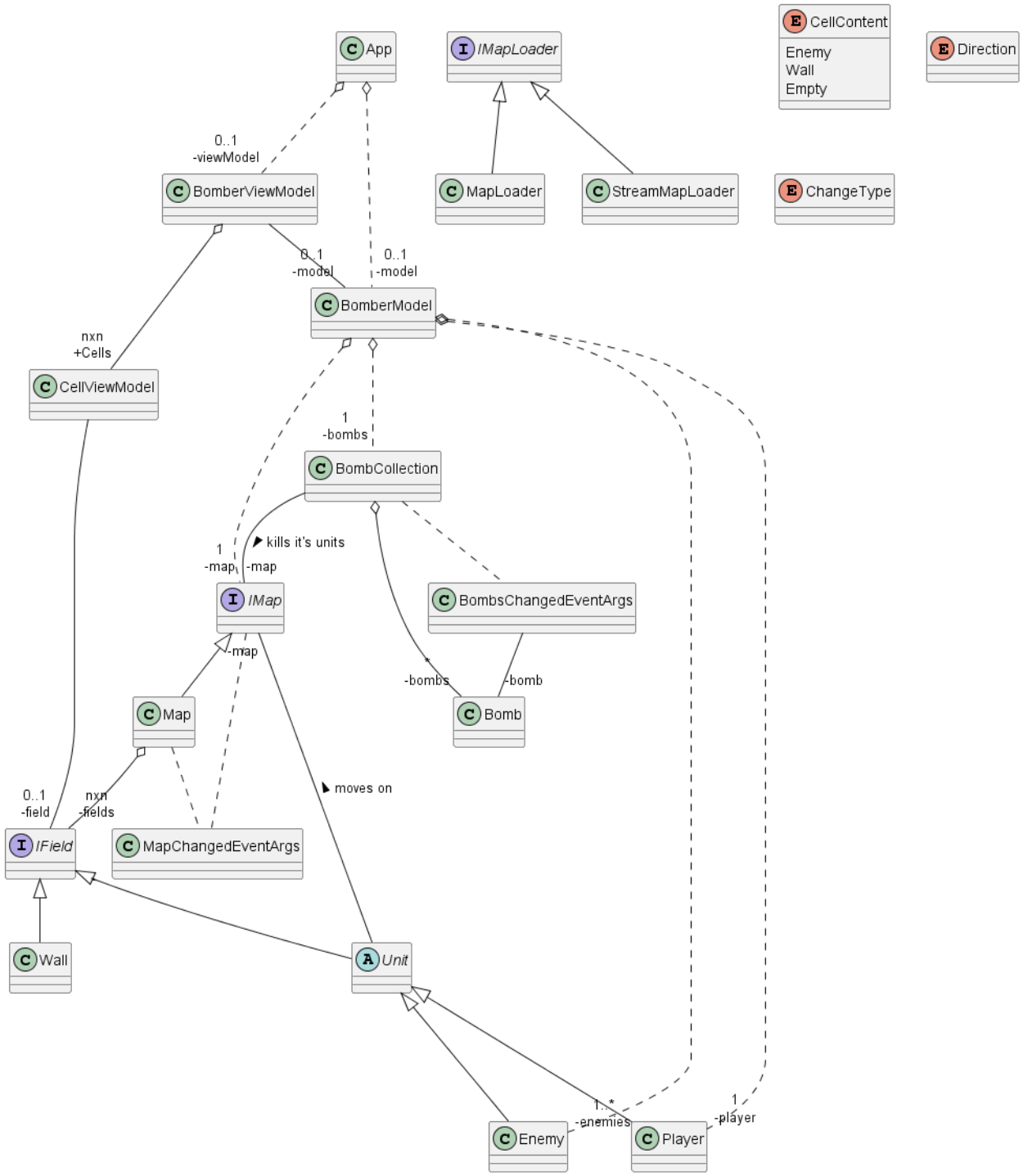
ViewModel:


BomberViewModel
Fields
enemiesKilled: int
time: TimeSpan
isMobile:bool
Properties
EnemiesKilled(): int
Time(): TimeSpan
Cells(): ObservableCollection<CellViewModel>
GameStarted(): bool
MapSize(): int
Paused(): boolean
Running(): boolean
ShowControls(): boolean
Events
LoadMap(): EventHandler?
Commands
MoveCommand(): RelayCommand<string>
PlantBombCommand(): RelayCommand
PauseToggleCommand(): RelayCommand
LoadMapCommand(): RelayCommand
Constructor
BomberViewModel(isMobile: boolean)
Methods
StartGame(model: BomberModel)
Event handlers
OnTimeElapsed(sender: EventArgs)
OnStatUpdated(sender: EventArgs)
OnMapChanged(sender: Map.MapChangedEventArgs)
OnGameOver(sender: EventArgs)
OnBombsChanged(sender: BombCollection.BombsChangedEventArgs)
OnMove(dir:string?)
OnPlantBomb()
OnPauseToggle()
Event methods
OnLoadMap()
nxn
+Cells
0..1
-model
CellViewModel
bombPlaced: bool
BombPlaced: bool
PlayerVisible: bool
EnemyVisible: bool
WallVisible: bool
X: int
Y: int
XY: Tuple<int, int>
properties
Field(): IField?
BomberModel
0..1
-field
IField

Vezérlés:


App
Properties
TopLevel:TopLevel
Initialize()
OnFrameworkInitializationCompleted()
ViewModel event handlers
ViewModel_LoadGame(sender: EventArgs, e: EventArgs)
Model event handlers
Model_GameOver(sender: EventArgs, e: EventArgs)
0..1
-viewModel
0..1
-model
BomberViewModel
BomberModel

Áttekintés:


App
IMapLoader
CellContent
Enemy
Wall
Empty
Direction
0..1
-viewModel
BomberViewModel
MapLoader
StreamMapLoader
ChangeType
0..1
-model
0..1
-model
BomberModel
nxn
+Cells
CellViewModel
1
-bombs
BombCollection
1
-map
kills it's units
-map
IMap
BombsChangedEventArgs
-map
*
-bombs
-bomb
Bomb
Map
0..1
-field
nxn
-fields
moves on
IField
MapChangedEventArgs
Wall
Unit
1
-enemies
*
1
-player
Enemy
Player

Tesztelés:

- A modell osztályaihoz egységtesztek készültek
- BombTest:
    - TestExplode: Ellenőrzi, hogy a bomba felrobban-e a beállított idő után.
    - TestPause: Ellenőrzi, hogy a bomba felrobban-e, amikor a játék szüneteltetve van.
    - TestResume: Ellenőrzi, hogy a bomba felrobban-e, miután folytatódik a játék.
    - TestDispose: Ellenőrzi, hogy a bomba helyesen kerül-e felszámolásra.
- BombCollectionTest:
    - TestPlantBomb: Ellenőrzi, hogy a bomba megfelelően hozzáadódik-e a gyűjteményhez, és az esemény kiváltódik-e.
    - TestPlantBombWithMultipleBombs: Ellenőrzi, hogy több bomba is megfelelően hozzáadódik a gyűjteményhez, és minden bomba hozzáadásakor esemény kiváltódik.
    - TestBombExplosion: Ellenőrzi, hogy egy bomba felrobbanása után eltávolításra kerül a gyűjteményből.
    - TestRemove: Ellenőrzi, hogy egy bomba eltávolítása után a gyűjteményből az esemény megfelelően kiváltódik, és a bomba törlődik.
    - TestPause: Ellenőrzi, hogy az osztály szüneteltetés során a bombái is szüneteltetésre kerülnek.
    - TestResume: Ellenőrzi, hogy az osztályra hívott Resume hatására a bombái is megkapják-e a Resume-ot.
    - TestDispose: Ellenőrzi, hogy az osztály felszámolása során a bombák is megfelelően törlődnek.
- EnemyTest:
    - TestMove: Ellenőrzi, hogy az ellenség helyesen mozog-e
    - TestCollisionWithPlayer: Ellenőrzi, hogy az ellenséggel történő ütközés során a játékos meghal-e.
    - TestCollisionWithWall: Ellenőrzi, hogy az ellenség haladási iránya megváltozik, amikor falnak ütközik.
- MapTest:
    - TestInitialization: Ellenőrzi a térkép inicializálását, és hogy a megfelelő objektumok vannak a megadott mezőkön.
    - TestRemoveFieldOutOfMap1: Ellenőrzi, hogy egy térképen kívüli mező eltávolításakor kivétel keletkezik.
    - TestRemoveFieldOutOfMap2: Ellenőrzi, hogy egy térképen kívüli mező eltávolításakor kivétel keletkezik.

    - TestRemoveField: Ellenőrzi, hogy egy adott mező eltávolítása után a megfelelő események kiváltódnak.
    - TestMovementAgainstWall: Ellenőrzi, hogy a játékos nem tud a falon átmenni
    - TestMovementOutOfMap: Ellenőrzi, hogy a játékos nem tud a térképről kilépni.
    - TestMovementAgainstEmptySpaceRight: Jobbra mozgás ellenőrzése
    - TestMovementAgainstEmptySpaceLeft: Balra mozgás ellenőrzése.
    - TestMovementAgainstEmptySpaceUp: Felfele mozgás ellenőrzése
    - TestMovementAgainstEmptySpaceDown: Lefele mozgás ellenőrzése
    - TestCollisions: Ellenőrzi a mezők közti ütközéseket
    - TestForeachRadius1: Ellenőrzi, hogy a ForEachInArea metódus helyesen iterál a megadott sugárban lévő mezőkön.
    - TestForeachRadius2: Ellenőrzi, hogy a ForEachInArea metódus helyesen iterál a megadott nagyobb sugárban lévő mezőkön.
- PlayerTest:
    - TestPlantBombAlive: Ellenőrzi, hogy élő játékos helyesen rak-e le bombát
    - TestPlantBombDead: Ellenőrzi, hogy halott játékos ne tudjon bombát lerakni
- UnitTest:
    - TestMovement: Ellenőrzi, hogy az egység mozgása megfelelően működik.
    - TestMovementDead: Ellenőrzi, hogy a halott egység ne tudjon mozogni
    - TestKill: Ellenőrzi, hogy az egység halála kiváltja a Died eseményt.
- GameTest:
    - TestPlayerStep: Ellenőrzi, hogy a játékos lépése helyesen hajtódik végre, és a játékos, valamint a térkép lépési műveletei megfelelően kerülnek meghívásra.
    - TestPlantBomb: Ellenőrzi, hogy a játékos sikeresen rak le bombát, és a bomba hozzáadódik a BombCollection-hoz.
    - TestPause: Ellenőrzi, hogy a játék szünetelése leállítja az ellenségek mozgását és a bombák detonálását, valamint a játék időzítését.
    - TestEnemySteps: Ellenőrzi, hogy az ellenségek megfelelő időközönként mozognak a játékban.
    - TestTimeElapsed: Ellenőrzi, hogy a játék számolja az eltelt időt.