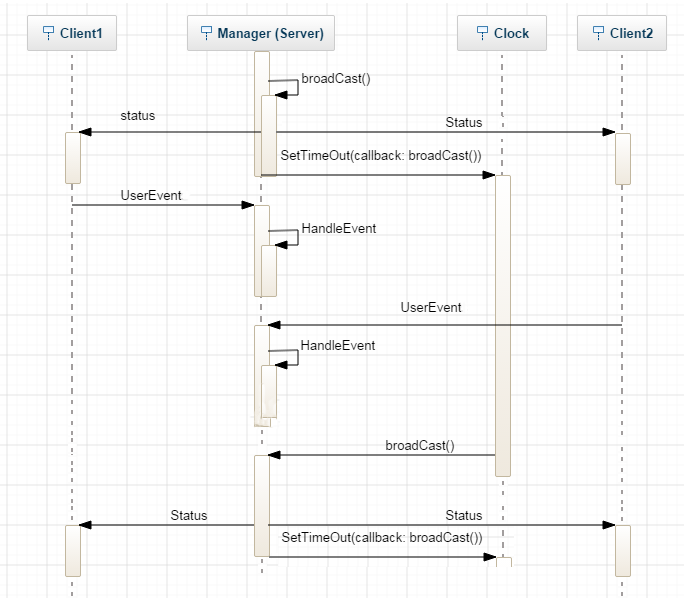
Modell

## Architektúra jellegű áttekintő szekvencia

****

## Objektumok – Szerver oldali működés

### Player

Egy játékos, mozogni és bombákat lerakni képes

### WallBomb

Bomba, mely időzítve felrobban, majd hatósugarában falat épít

### FireBomb

Bomba, mely időzítve felrobban, majd hatósugarában falat rombol, és Fire objektumot hagy maga után.

### Wall

A WallBomb ilyet épít, illetve alapból is része a pályának. A játékosok számára áthatolhatatlan, amíg nem foglalnak területet.

* 1. **Cell**

A pálya egy kis része, mely tovább nem osztható. Tudja, hogy ki áll rajta, és ütközés esetén értesíti az illetékeseket, ismeri a szomszédos Cell objektumokat, és tudja van-e rajta fal, és el tudja pusztítani azt szükség esetén, illetve, tudja, hogy hol van.

* 1. **Fire**

A FireBomb robbanásakor keletkező tűzcsóva, mely kis ideig ott marad a hatósugarában paralizálva a játékosokat, akik belelépnek.

* 1. **Property**

Egy játékos egy birtoka (több is lehet), fallal elkerített területe, melyen az elkerítés pillanatában egyedül tartózkodott. A játékos a saját Property-ét körbevevő falon áthatolhat. Az a játékos nyer, akinek a legtöbb cellája van az összes Property-én együttvéve, a játék végén, vagy aki megszerzi az összes cella felét Property-ként.

## Osztálydiagram - Szerver oldali működés

## Osztályleírások - Szerver oldali működés

### <<Abstract>> GameElement

* **Attribútumok:**
  + **impassable: boolean –** Megmondja hogy játékelem áthatolható-e, praktikusan, hogy rá tud-e lépni a játékos
  + **position: Cell –** A játékelem melyik cellán van
* **Metódusok**
  + **isImpassable(): boolean –** Lekérdezi, hogy a játékelem áthatolható-e
  + **getPosition(): Cell –** Visszatér a cellával ahol a játékelem található
  + **setPosition(c: Cell): void –** Beállítja a kapott cellát a játékelem cellájául

### <<Interface>> ICollidable

*(visitor pattern)*

* **Metódusok**
  + **collide(Bomb): void –** Ütközés, amikor Bomb érkezik a Cellára.
  + **collide(Player): void –** Ütközés, amikor Player érkezik a Cellára.
  + **collide(Fire): void –** Ütközés, amikor Fire érkezik a Cellára.

### Player

* **Ősosztályok:**
  + GameElement
* **Interfacek:**
  + ICollidable
* **Attribútumok:**
  + **color: Color –** A játékos színe
  + **c: Clock –** Az időzítő, erre tudja feliratkoztatni egyes funkcióit, hogy időzítve hajtódjanak végre (deparalizáció például)
  + **avilableBombs: int –** A játékos rendelkezésére álló bombák száma
  + **isParalized: boolean –** Megmondja, hogy a játékos paralizálva van-e (amíg ez az állapot fennáll nem rakhat bombát)
  + **properties: List<Property> –** A játékos birtokainak listája
  + **paralizedTime: int –** Statikus tag, mely megmondja a player osztály mennyi ideig paralizálódik, ha paralizálják
  + **setupAvailableBombTime: int –** Statikus tag, mely megmondja a player osztály milyen gyakran kap új bombát, melyet lerakhat (milyen sűrűn hívódjon a **setupAvailableBomb()** függvény)
* **Metódusok**
  + **canMove(d: Direction): boolean –** Megmondja, hogy a felhasználó mozoghat-e az adott irányba, itt a Direction belül a szokásos irányokra nézve, megvizsgálja a szomszédos cellákon van-e Wall objektum mely nem property rész, és ha az adott irányban van, hamis értékkel tér vissza, különben igazzal.
  + **move(d: Direction): void –** Lépteti a játékost az adott irányba
  + **getColor(): color –** Visszaadja a játékos színét
  + **makeFireBomb(): void –** Ha a játékos nincs paralizálva (isParalized == false), létrehoz egy FireBomb objektumot a Cell-en ahol a felhasználó van, egyúttal feliratkoztatja a Clock objektumra a bomba explode() metódusának átadásával (callback)
  + **makeWallBomb(): void –** Ha a játékos nincs paralizálva (isParalized == false),létrehoz egy WallBomb objektumot a Cell-en ahol a felhasználó van, egyúttal feliratkoztatja a Clock objektumra a bomba explode() metódusának átadásával (callback)
  + **setupAvailableBomb(): void –** Hozzáad egyet az availableBomb változóhoz, s egyúttal újrairatkozatja magát a Clock-ra, a s**etupAvailableBombTime**-ban tárolt időtartammal
  + **countAllProperty(): int –** Megszámolja a Játékos összes Property-én lévő cellákat, és visszetér az összegükkel
  + **paralize(): void –** Beállítja a paralized jelzőt, ezzel meggátolva hogy bombát tegyen a játékos, és beregisztrálja a Clock-hoz a deParalize() metódust, a statikus **paralizedTime** attribútumban meghatározott delay-el.
  + **deParalize(): void –** Beállítja az isParalized változó értékét hamisra, ezzel felhatalmazva, hogy újra rakhasson bombákat

### Manager

* **Ősosztályok:**
  + GameElement
* **Interfacek:**
  + ICollidable
* **Attribútumok:**
  + **clients: List<Client> –** szerverhez kapcsolódó kliensek listája
  + **c: Clock –** Az időzítő, erre tudja feliratkoztatni egyes funkcióit, hogy időzítve hajtódjanak végre
  + **map: Map<Point, Cell> –** A pálya, pontokat kulcsként használva egy-egy cellát tárolva
  + **status: Map<Point, String[]> –** A státusz, mely minden ponthoz tárolja, milyen objektumok van ott, a nevével azonosítva (pl ’Wall’). Ez a státusz kerül minden adott egységnyi és oszthatatlannak tekintett idő leteltével ki az összes klienshez, akik ebből a pályát a felhasználók számára megjelenítik
* **Metódusok**
  + **broadcastClients(): void –** Szétszórja a kliensek közt broadcastszerűen a státuszobjektumot
  + **makeStatusFromMap(): void –** Előállítja a státuszobjektumot a cellák lekérdezésével
  + **makeProperty(): void –** elkészíti a Property objektumot a játékos számára ha a felhasználó bekerített egy területet
  + **findProperty(): void –** Ellenőrzi létrejöhetett-e Property
  + **checkPropertyCellsCount(): boolean –** Ellenőrzi megszerezte-e valaki a győzelemhez szükséges (legalább a mezők fele) mennyiségű cellát, Propertikben. Csak akkor hívódik ha valaki épp Property-t szerzett.
  + **endGame(): void –** Győztest hirdet, lezárja a játékot
  + **catchEvents(): void –** Elkapja a felhasználói eventeket és továbbítja a Player objektumoknak

### Cell

* **Attribútumok:**
  + **position: Point –** A pályán a helyét tartja ebben számon
  + **visitors: List<ICollidable> –** A cellán lévő ütközésre hajlamos objektumokat tartja itt számon
  + **w: Wall –** Ha van fal objektum a cellán, az ebben van számon tartva
  + **neighbours: List<Cell>–** A cellával szomszédos cellák listája
* **Metódusok**
  + **add(p: Player) –** Player hozzáadása a cellán lévőkhöz, előtte pedig collide(Player) jelzés kiküldése mindenkinek
  + **add(f: Fire) –** Fire hozzáadása a cellán lévőkhöz, előtte pedig collide(Fire) jelzés kiküldése mindenkinek
  + **add(b: Bomb) –** Player hozzáadása a cellán lévőkhöz, előtte pedig collide(Bomb) jelzés kiküldése mindenkinek (ez azért fontos, mert a bombák esetleg felrobbannak ha tűz éri őket, így regisztrálniuk kell)
  + **add(w: Wall) –** Wall objektum felvétele a cellára
  + **remove(ic: ICollidable) –** Töröl a listából egy elemet
  + **removeWall() –** Törli a fal objektumot, jellemzően az add(f: Fire) hívja, ugyanis a FireBomb robbanásakor keletkező Fire eltünteti a falakat
  + **setNeighbours(top: Cell, left: Cell, bottom: Cell, right: Cell) –** Beállítja a szomszédos cellákat
  + **getNeighbours(): List<Cell> –** visszaadja a szomszédos cellákat

### Wall

* **Ősosztályok**

GameElement

* **Metódusok**
  + **destroy(): void –** Törli a falat a Cellából, annak removeWall() függvényének hívásával

### <<Abstract>> Bomb

* **Attribútumok:**
  + **scope: int –** A bomba hatósugara, ahol hatását kifejti
  + **c:Clock –** Az időzítő melyet használ, hogy időzítve robbanjon
  + **delayToExplode: int –** Az idő, melyet az egyes bombáknak időzítésül használniuk kell
* **Metódusok**
  + **explode(): void –** Absztrakt metódus a gyerekei implementálják, az időzítés végeztével hívja vissza a Clock objektum, s elindít egy folyamatot mely során a bomba kifejti hatását, létrehozza a megfelelő Wall vagy Fire objektumokat.
  + **getScope(): int –** Visszatér a bomba hatáskörét leíró számmal

### FireBomb

* **Ősosztályok**

Bomb

### WallBomb

* **Ősosztályok**

Bomb

### Fire

* **Ősosztályok:**

GameElement

* **Interface:**

ICollidable

* **Attribútumok:**
  + **delayFireBurns: int –** A statikus változó mely tárolja, egy Fire objektum meddig kell a pályán tartózkodjon
  + **c: Clock –** A Clock, mely figyelmezteti majd, hogy már ne létezzen
* **Metódusok**
  + **burn(p: Player) –** Megégeti és ezáltal paralizálja majd a játékost.

### Clock

* **Metódusok**
  + **add(delay: int, callback: function) –** A paraméterként kapott idő elteltével visszahívja a második paraméterben kapott függvényt

### Property

* **Attribútumok:**
  + **player: Player –** A játékos a birtokolja a Property-t
  + **cells: List<Cell> –** A Cellák melyek a Property részei
* **Metódusok**
  + **getCellsCount() –** Megszámolja a Propertyn lévő Cellákat és visszatér az összeggel

### Client

* **Attribútumok:**
  + **player: Player –** A játékos akit a kliens kezel
  + **sessionID: int –** A session id-je mely a klienshez tartozik és azonosítja
* **Metódusok**
  + **getPlayer(): Player –** Visszatér a kliens Player objektumával