2. Beadandó feladat dokumentáció

Készítette:

Wágner András
B31ZE5@inf.elte.hu

Feladat:

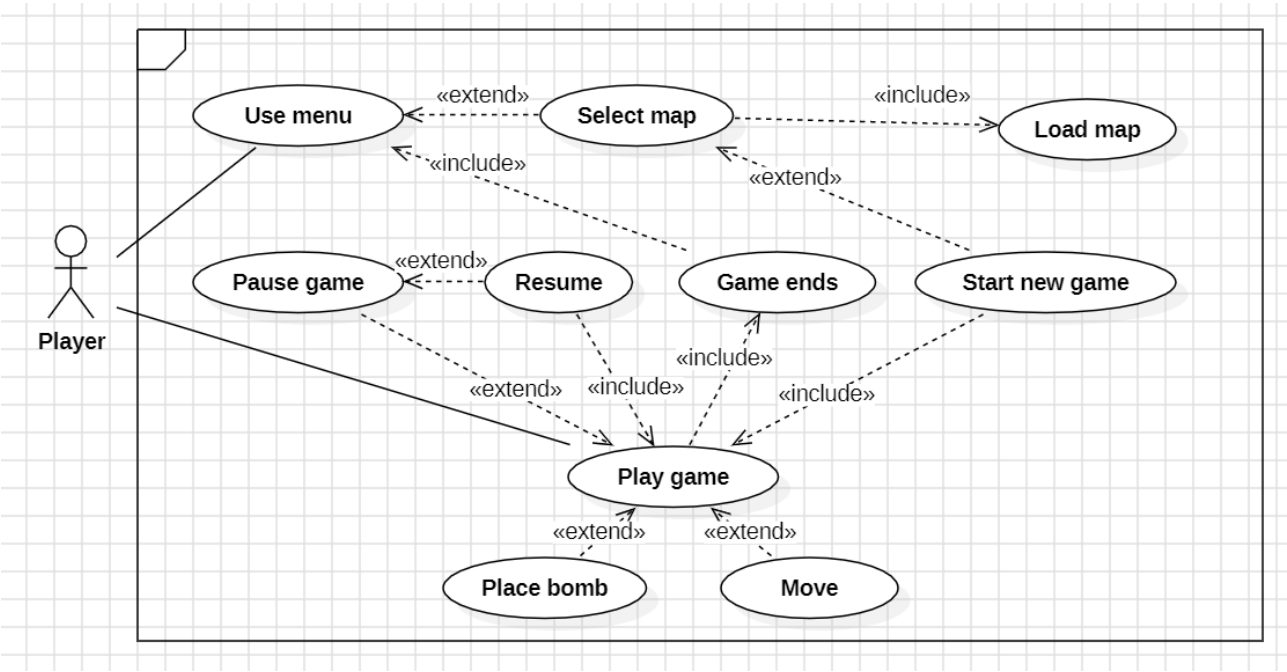
Bombázó

Készítsünk programot, amellyel a következő játékot játszhatjuk.
Adott egy n × 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 feladatot egyablakos asztali alkalmazásként WPF grafikus felülettel valósítjuk meg.
- A program futása során két grafikus felület váltja egymást az alkalmazás ablakában: egy menü és egy játéktér.
- A program indításakor a menü jelenik meg. Ezen a felületen gombok segítségével lépkedhetünk a játékpályák között, illetve egy Start gomb megnyomásával elindíthatunk egy játékot az aktuálisan kiválasztott pályán.
- A játéktér felületen egy a pálya méretei által meghatározott méretű, a mezőket tartalmazó rács és egy a játékidőt és a felrobbantott ellenségek számát mutató státuszsor található.
- A mezők (a játék kezdetén fájlból kiolvasott elrendezésben) falakat, ellenségeket, vagy a játékos karakterét, illetve a játékos által lerakott bombákat reprezentálhatják.
  - A falak pozíciója fix, mezőikre semmi más nem kerülhet.
  - Az ellenségek minden másodpercben egy szomszédos mezőre lépnek. Az egyes ellenségek addig haladnak egy irányba, míg falba, más ellenségbe, vagy bombába nem ütköznek, ekkor véletlenszerűen új, szabad irányt választanak, ha ez nem lehetséges, kihagynak egy kört.

    - A játékos karaktere a nyíl billentyűk lenyomásakor lép a megfelelő irányban található szomszédos mezőre, amennyiben az nem fal.
    - A space billentyű lenyomásakor a játékos pozíciójában létrejön egy bomba. A bomba 5 másodperc múlva felrobban, ekkor a hatókörében, a bomba középpontú 7×7 mezős négyzetben található karakterek - ellenségek illetve a játékos karaktere – megsemmisülnek.
    - Ha a játékos karaktere és egy ellenségel egy mezőre kerülne (a játékos vagy az ellenség lépése után) megsemmisül.
- Ha a játékos karaktere megsemmisül, a játékos veszít és a játék véget ér.
- Ha az összes ellenség megsemmisül, a játékos győz és a játék véget ér.
- A játék végéről előugró ablak tájékoztatja a felhasználót, az ablak tartalma pedig visszavált a menü felületre.
- Játék közben az escape billentyűt megnyomva a felhasználó megállíthatja, ismét megnyomva pedig folytathatja a játékot.



Tervezés:

Programszerkezet:

- A programot négyrétegű MVVM architektúrában valósítjuk meg:
    - A megjelenítés struktúráját a `WPFView`
    - A megjelenítéshez szükséges logikát a `ViewModel`
    - A modellt a `Model`
    - A perzisztenciát a `Persistence`

  névterek biztosítják.
- A programot a névterek mentén négy projektre osztjuk, így mind a WPF-től függő megjelenítés, illetve az azt biztosító logika, mind a lokális `txt` fájlokat olvasó perzisztencia könnyen lecserélhető, míg a működés logikáját biztosító modell teljesen rendszerfüggetlen, így más megjelenítéssel és perzisztenciával is konzisztens működést biztosít.

Perzisztencia:

- A perzisztencia feladata a játéktáblák adatainak betöltése.

- Az `IMapReader` interfész biztosítja, hogy minden a játék indításához szükséges adat elérhető:
  - A `Map` a játéktér dimenzióit és a falak elhelyezkedését tartalmazza, utóbbit két dimenziós boolean tömbben, mivel valószínűleg nagy mennyiségű fal található a játéktéren, amik közül gyakran kell lekérdezni, a tároló méretén pedig a játék során nem kell változtatni.
  - Az `EnemiesStart` az ellenségek kiinduló pozíciói lista formájában, mivel az ellenségek száma változhat, a lassabb elérés pedig várhatóan kisebb számuk miatt nem jelent problémát.
  - A `PlayerStart` a játékos karakterének kiinduló pozíciója.
- A `TXTMapReader` osztály a fenti interfészt implementálja, `txt` fájlból olvasva a tábla adatait, melyben minden karakter egy-egy mező tartalmát jelzi a kezdeti állapotban.
- A `TXTMapReader` teszteket is végez a fájlból beolvasott pálya validálására. n × n-es dimenziók helyett a bővíthetőség és az eredeti NES játékhoz való hűség jegyében csak téglalap alakot követel meg, illetve megadható a bal felső saroktól eltérő kezdőpozíció is megadható a játékosnak. A programban szereplő pályák azonban a feladatnak megfelelőek.

Modell:

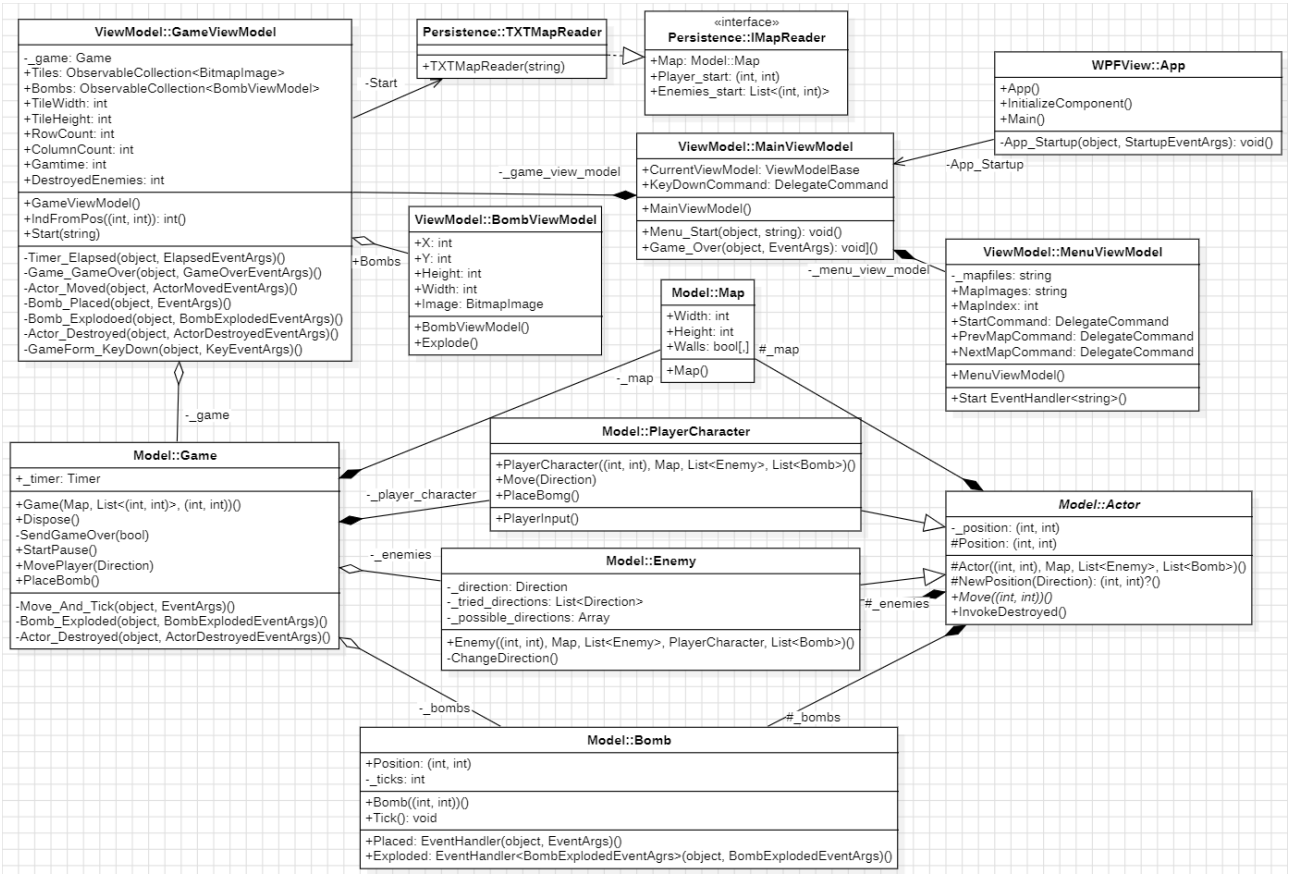
- A modell „fő osztálya” a `Game`, melynek egy példánya felelős egy játék levezényléséért. A `Game` a perzisztenciából kiolvasott adatokkal példányosítható és azoknak megfelelően konstruktorában példányosítja a többi modellbeli osztály objektumait (a bombák kivételével), átadva működésükhöz szükséges adatokat. A játék során a `Game` fogadja a felhasználó inputjai és az időzítő által kiváltott eseményeket, meghívva a többi objektum megfelelő metódusait.
- A játékos karaktere és az ellenségek sok funkcionalitáson osztoznak. Ezek, a mozgás, a megsemmisülés, és az ezekhez kapcsolódó események, melyeket a nézet és `Game` is figyel a közös `Actor` ősosztályban vannak megvalósítva, melynek leszármazottai a `PlayerCharacter` és az `Enemy`.
- A bombákat a `Bomb` osztály reprezentálja. Ez a `PlayerCharater` osztályból példányosítható. A `Game` objektum minden másodpercben meghívja minden bomba `Tick` metódusát. Az ötödik ilyen hívás után a bomba `Exploded` eseményt vált ki, mely tartalmazza a pozícióját.
  - A `Game` ez alapján ellenőrzi, mely aktorok vannak benne, a hatókörben, azaz megsemmisülnek-e. A megsemmisülteket és a küldő bombát pedig kiveszi a játékban lévő ellenségek, illetve bombák listájáról.
  - Azért nem az egyes szereplők fogadják a robbanás eseményt, hogy biztosan a játékos megsemmisülése kerüljön először tesztelésre, és ha megsemmisült, ki lehessen váltani a `GameOver` eseményt.
- A `Game` által létrehozott `System.Timers.Timer` másodpercenként kiváltott eseménye hatására a `Game` minden ellenségre meghívja annak `Move` függvényét, a `View`-ből érkező hívás alapján pedig a játékossal teszi ezt.
- A játék ennek a `Timer`-nek a megállítása, illetve a játékost irányító hívások továbbításának megtagadása által állítható meg.
- A játékos vagy az összes ellenség megsemmisülésekor a `Game` a `GameOver` eseménnyel jelez a nézetnek.

Nézet:

- Minden a `ViewModel`-ben található osztály a `ViewModelBase` leszármazottja, mely az `InotifyPropertyChanged` interfészt megvalósítva lehetővé teszi a `WPFView` elemeinek frissítését. A felhasznáó inputjait az `ICommand` interfészt megvalósító `DelegateCommand`-segítségével kezeljük.
- Az ablak megjelenítéséért és a menü és játék közti váltásért a `MainWindow.xaml` és a

`MainViewModel` felelősek.

- A `MenuView` User Control és a hozzá tartozó `MenuVeiwModel` alkotják a játék menüjét, ahol a pályák között lehet választani, illetve a kiválasztott pályán játékot indítani. Utóbbit a `Menu` eseménnyel jelzi, melyben a megfelelő pályát tartalmazó fájlt is megosztja. A `MainViewModel` ezzel az elérési úttal hívja meg a `GameViewModel Start` metódusát, és lecseréli a megjelenített `User Control`-t a `MenuView` a `GameView`-ra.
- A `GameViewModel Start` metódusa példányosítja a TXTMapFileReader-t majd az az által a perzisztenciából visszadott adatokkal példányosítja a `Game`-et, illeve feltölti a `GameView`-ban kötött kollekciót az egyes mezők képeivel.
- A nézet modell innentől a modelltől kapott eseményekre reagálva a megjelenítés frissítéséért felel. Ezt az `Actor_Moved`, `Actor_Destroyed`, `Bomb_Placed`, és `Bomb_Exploded` függvények végzik.
- A `GameOver` esemény hatására előugró ablakon közöljük a játékossal az eredményt, a `GameViewModel` pedig saját esemény vált ki, aminek hatására a `MainViewModel` visszaállítja az ablak tartalmát a `MenuView` user control-ra; a `Game` objektumát tartalmazó `_game`-et pedig `null`-ra állítjuk felkészülve új játék indítására.



Tesztelés:

- A modell funkcionalitásának ellenőrzésére egységteszteket végzünk.
- A tesztek a `ModelTest` projektben kerülnek megvalósításra.
- A tesztelésre kerülő osztályoka a következők:
  - `Game`
  - `PlayerCharacter`
  - `Enemy`
  - `Bomb`
- Ezek teszteléséért az értelemszerűen elnevezett osztályok felelnek.