**Beadandó feladat dokumentáció**

Készítette: Varga Gergely (WGHIQQ)

e-mail: [wghiqq@int.elte.hu](mailto:wghiqq@int.elte.hu)

**Feladat: Aknakereső**

Készítsünk programot, amellyel az aknakereső játék két személyes változatát játszhatjuk.

Adott egy 𝑛 × 𝑛 mezőből álló tábla, amelyen rejtett aknákat helyezünk el. A többi mező szintén elrejtve tárolják, hogy a velük szomszédos 8 mezőn hány akna helyezkedik el. A játékosok felváltva léphetnek. Egy mező felfedjük annak tartalmát. Ha az akna, a játékos veszített. Amennyiben a mező nullát rejt, akkor a vele szomszédos mezők is automatikusan felfedésre kerülnek (és ha a szomszédos is nulla, akkor annak a szomszédai is, és így tovább). A játék addig tart, amíg valamelyik játékos aknára nem lép, vagy fel nem fedték az összes nem akna mezőt (ekkor döntetlen lesz a játék). A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (6 × 6, 10 × 10, 16 × 16), valamint játék mentésére és betöltésére. Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött (ha nem döntetlen).

**Elemzés:**

* A játékot három különböző méretű táblával tudjuk játszani (6 × 6, 10 × 10, 16 × 16). Természetesen ahogy nő a pályaméret, nő a bombák száma is. A program indításakor a Menu-be lépünk, ahol választhatunk, hogy játékot betölteni akarunk-e vagy újat kezdeni.
* A feladatot két ablakkal tervezem megoldani. Egy menüvel és egy játéktáblával.
* A menüben négy gombot tervezek elhelyezni. A betöltéshez tartozó gombot aminek a hatására egy dialógusból kiválaszthatunk egy fájlt, ami alapján egy mentett játékot tudunk betölteni. A másik három rendre a 6, 10, 16 széles táblán történő új játék indítására szolgál.
* A játéktbla felett elhelyezek egy feliratot, amely jelzi, hogy ki a következő játékos.
* A játéktábla alatt elhelyezek egy gombot, ami az aktuális játékállapot mentésére szolgál.
* Az 𝑛 × 𝑛-es játéktáblát egy ugyanekkora gombokból álló rács reprezentálja. Alapállapotban egyik sem rendelkezik felirattal. Ha jobb klikkel kattintunk rá, egy piros B betűt írok a gombra, jelezvén, hogy jeleztem magamnak, hogy ez egy bomba. Ez természetesen csak nekünk segít. Magát a játék állapotát nem érdekli. A bal klikkel kattintás hatására, ha bombát találtunk, kírjuk egy ablakban, hogy a játék véget ért, és hogy ki veszítette el ezáltal a játékot.
* Ha nem bombára kattintunk, akkor a gomb disabled-dé válik és (ha nem nulla) kiírjuk, hány vele szomszédos bomba van (ha nulla, üresen hagyjuk a gomb feliratmezőjét).
* Amennyiben nullás cellára kattintunk, végignyitjuk a feladaleírásnak megfelelően, rekurzívan a felfedezendő területet automatikusan
* Minden kinyitott mező esetén regisztráljuk, hogy eggyel több ellát nyitottunk.
* Döntetlent akkor állapítok meg ha az ÖsszesMező – BombákSzáma értéket megüti a kinyitott mezők száma.
* Ha döntetlen, a játék kiírja, hogy szépen dolgoztak a játékosok, bezárja a játékteret és visszatér a főmenübe.
* A felhasználói esetek az alábbi ábrán láthatóak.

Diagram

Description automatically generated

**Tervezés:**

* Programszerkezet:
  + A programot ***MVVM*** architektúrában valósítjuk meg, ennek megfelelően View, Model, ViewModel és Persistence névtereket valósítunk meg az alkalmazáson belül. A program környezetét az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést. A program csomagszerkezete a következő. ábrán látható.
  + A Nézethez külön projektet hozunk létre (***AknakeresőWPF***). A modell és a perzisztencia réteg szintén külön projektet kap (***MineField***).
  + A ***MineField*** projekten belül természetesen külön namespaceben (***Model***, ***Persistence***) helyezkednek el a modellhez és perzisztenciához tartozó fájlok, így a háttérlogika felületfüggetlen marad.
* Perzisztencia:
  + Arra használom, hogy tartósan tudjak a játék futása nélkül játékállapotot tárolni, illetve igény esetén azt teljes egészében visszaállítani.
  + Az adatokat a gépen kiválasztott helyre egy .txt fájlba mentem.
  + A fájl első sora pontosvesszővel elválasztva tárolja a tábla szélességét (n), a bombák számát és a már kinyitott mezők számát.
  + A következő sorban egy bool érték string reprezentatívjával tárolom, hogy következőnek az első, vagy a második játékos jön-e
  + Ezután 𝑛 × 𝑛 soron keresztül tárolom az aknamező minden mezőjének pontos állapotát (Bomba-e vagy sem, pozíciója a táblán, ki van-e nyitva, van-e rajta zászló, és hogy a környezetében hány bomba tárolható). Az adatokat pontosvesszővel választom el. Bombaságot egy enum string reprezentációjával teszem meg, a pozíciót két int-el tárolom, vesszővel elválasztva, a zászlót és nyitottságot bool-ok string reprezentációjával tárolom, a szomszédos bombák számát int-ként.
  + Az írást és olvasást fájl elérési út alapján a ***MineFieldPersistence***  osztály ***SaveGame***és***LoadGame*** metódusok biztosítják.
  + Beolvasási hiba esetén ***MineFieldDataException-***t dob az adott metódus.

Diagram

Description automatically generated

* Modell:
  + A modellben a játé állapotát és az összes mezőt a ***MineFieldState*** osztály kezeli.
    - A *\_cells*  kétdimenziós tömbb kezeil a mezőket.
    - A játék végét a *GameOver* event jelzi.
    - A játék alapinfóit a *BoardSize* és *BomCount* tartalmazza.
    - A játékosok a *FirstPlayer*, *SecondPlayer* fieldekben érhetők el.
    - Azt, hogy ki jön, a *FirstPlayerIsNext* bool változó tárolja.
    - A már kinyitott mezők számát a *RevealedCells* tárolja.
  + Mezőtípusokat a ***CellType*** felsoroló típus reprezentálja, segíti a mentését.
  + Egy-egy mezőt egy ***Cell***ősosztályból leszármazott ***Bomb*** vagy ***SafeCell*** objektum reprezentál.
    - Egy cella tudja magáról, hogy melyik táblához tartozik ***Field*** propertiján át.
    - Azt is tudja, azon belül hol található a VerticalPosition, HorizontalPosition számokon át.
    - Ezen felül tudja, hogy ő ki van e nyitva (***IsRevealed***) és hogy raktak-e rá zászlót (***IsFlagged***).
    - A megnyitását, annak a rekurzív továbbhívását a ***Reveal***  funkció kezeli.
    - Ugyanez a bomba esetén meghívja a játék ***GameOver*** eventet a játáktáren elsütő, ***InvokeGameOverOnDeath*** függvényt, hogy a játék véget ért.

**Diagram

Description automatically generated**

* Nézetmodell:
  + A nézetmodell megvalósításához felhasználunk egy általános utasítás (***DelegateCommand***), valamint egy ős változásjelző (***ViewModelBase***) osztályt.
  + A nézetmodell feladatait a ***MineFieldViewModel*** 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 vezérlőnek. A nézetmodell tárolja a modell egy hivatkozását (***\_model***), de csupán információkat kér le tőle. Direkt nem avatkozik a játék futtatásába.
  + Tartalmaz egy (***\_persistence***) hitvatkozást egy perzisztenciát kezelő ***MineFieldPersistence*** objektumra, amelyen keresztül a mentés és a betöltés kezelhető, szintén parancsokon keresztül.
  + A játékmező számára egy külön mezőt biztosítunk (***MineCellViewModel***), amely eltárolja a pozíciót, felfedett vagy zászlózott-e az adott mező, ha nyitott, hogy hány bomba van körülötte (szövegként), valamint a felfedés és zászlózás parancsát (***FlagCellCommand, RevealCommand***). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (***Fields***).

**Diagram, timeline

Description automatically generated**

* Nézet:
  + A nézet csak egy képernyőt tartalmaz, a ***MainWindow*** osztályt. A nézet egy rácsban tárolja a játékmezőt, a menüt és a státuszsort. A játékmező egy ***ItemsControl*** vezérlő, ahol dinamikusan felépítünk egy rácsot (***UniformGrid***), amely gombokból áll. Minden adatot adatkötéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a gombok színét is.
  + A fájlnév 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, a perzisztencia kezelése és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.

**Graphical user interface

Description automatically generated with low confidence**

**Tesztelés:**

* A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a ***TestForMineField*** osztályban
* Az alábbi tesztek kerültek megvalósításra:
  + MineField konstruálás
    - ***MineField\_BasicConstructTest\_Done*** amely ellenőrzi, hogy megfelelő mennyiségű bomba lett-e elhelyezve a pályán, a tábla megfelelő méretű-e és hogy alapból minden mező elrejtett és zászlómentes.
    - ***MineField\_BasicConstructTest\_NotLegitBoardSize***  amely ellenőrzi, hogy amennyiben nem megengedett méretű táblát hoznánk létre, dob-e Error-t a konstruktor.
  + MineField játék vége, lépések
    - ***MineFieldBombClickTest*** amely ellenőrzi, hogy bombára kattintás esetén, a GameOver event elsül-e, illetve hogy a játék helyesen jelzi-e, hogy nem döntetlen, hanem vesztes a játék.
    - ***MineField\_SuccessTest*** amely ellenőrzi, hogy amennyiben minden egyes nem bomba mező megnyitásra került, a játék végetér-e és ha igen, helyesen mondja-e hogy a játék döntetlen.
  + Mezők fontos működése
    - ***Cell\_GetNeighbourCells\_Center***  illetve ***Cell\_GetNeighbourCells\_Edge***  ellenőrzi, hogy akörnyezetét helyesen szedi-e össze a mező, mind a térképen belül, mind a sarokban.
    - ***Cell\_DoesNotRevealOthersWhenHasBombNextToIt***  ellenőrzi, hogy amennyiben van a mező környkén bomba, nem-e indítja el a eurziót mint az üreseknél, hanem csak saját magát nyitja meg.