Koczkás Péter
QCB102

NHF Dokumentáció

1. Felhasználói útmutató

A játék célja

Az összes akna megtalálása a legrövidebb idő alatt anélkül, hogy felfednénk az aknákat.

A játék menete

Indítás után ki kell választani, hogy új játékot akarunk-e indítani, a ranglistát akarjuk-e megtekinteni vagy esetleg ki akarunk lépni.

A ranglista megtekintése esetén először ki kell választani, hogy melyik nehézségű ranglistára vagyunk kíváncsiak. A kiválasztás után megjelenik az adott ranglista (maximum 10 legjobb idő + név). A ranglista alatt található Back gombbal visszaléphetünk a főmenübe.

Új játék esetén először meg kell adni a nevet, amit, ha elég jó időt érünk el a ranglistában láthatunk. A név beírása után az Enter gombbal léphetünk tovább. Ekkor 3 nehézségi szint közül választhatunk (Easy/Medium/Hard). A nehézségi szint kiválasztása után megjelenik a játéktér.

Alaphelyzetben a mezők felderítetlenek. Továbbá egy mezőnek még lehet 3 másik állapota:

- felderített és nincs a szomszédos mezőkön akna
- felderített, de van a szomszédos mezőkön akna
- megjelölt (elképzelésünk szerint akna van az adott mezőn, de nem tártuk fel)
- feltárt akna (játék vége, a játékos a menetet elveszítette).

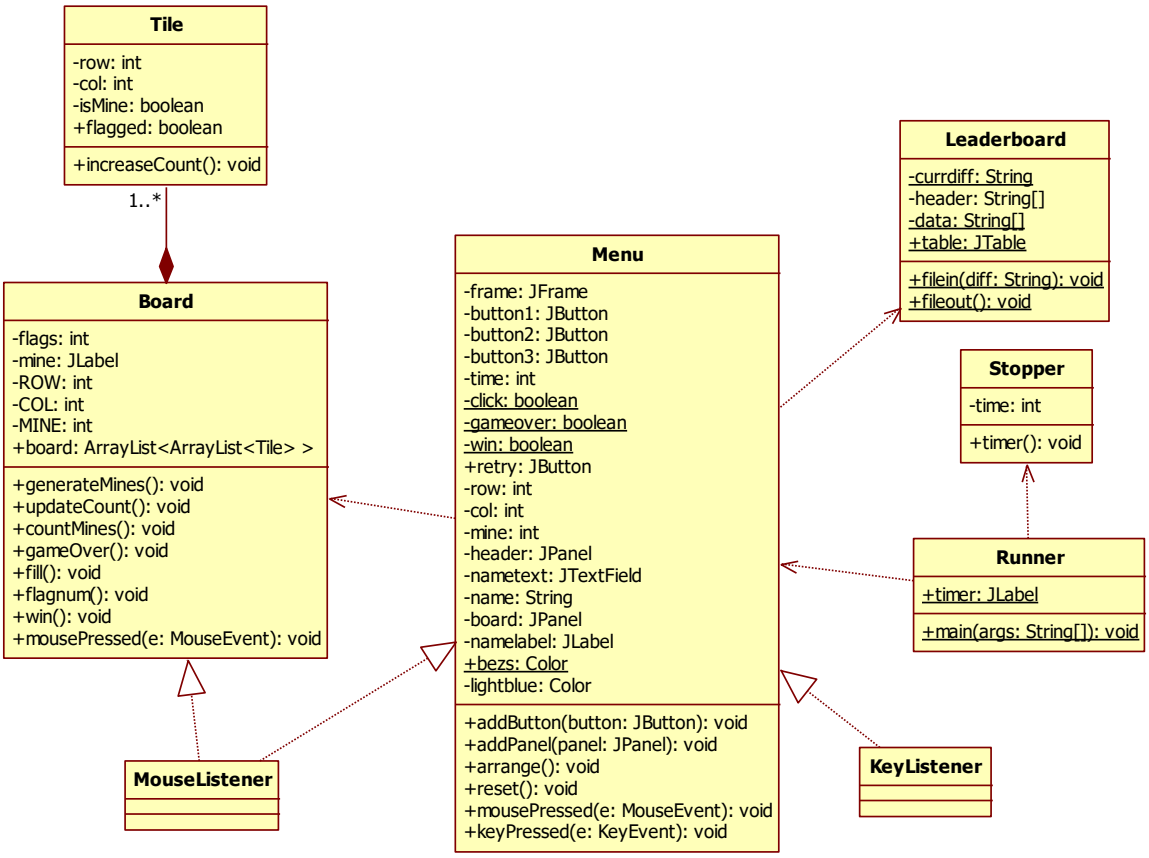
A megjelölt állapot az egér jobb gombjával érhető el. A játék megnyeréséhez nem kötelező használni a megjelölt állapotot, ez csupán segítséget nyújt a játékmenet során.

Egy mező felfedése az egér bal gombjával történik. Ha egy mező feltárult, és körülötte akna található, akkor annak a darabszámát egy számmal jelzi (1-8). Ha egy mező mellett nincs akna, akkor a mezővel szomszédos mezők is feltárulnak mindaddig, amíg azok a mezők is felfedetté válnak, melyek körül akna található.

A játék kezdeténél egy aknamentes mezőre kell kattintanunk, aztán logikai úton és néha szerencsével tudunk előre haladni a játék menetében. A játék folyamatosan jelzi a megjelöletlen aknák számát, illetve az eltelt időt. Ha aknára kattintunk, a játék véget ér, és az adott menetet elvesztettük. A játék csak akkor fejeződik be győzelemmel, ha felfedtük az összes olyan mezőt, ami nem tartalmaz aknát. Ekkor a játék véget ér, és ha elég jó időeredményt értünk el akkor a játék elején megadott névvel együtt az idő is bekerül a megfelelő ranglistába. A játék megnyerése nem függ attól, hogy hány aknát jelöltünk meg, illetve használtuk-e egyáltalán ezt a segítséget.

Új játékot úgy kezdhetünk, ha a fejlécben lévő Retry gombra kattintunk.

2. Osztálydiagram



3. Osztályok és a főbb függvények

class Tile: Egy Button, amely a mező a pályán. Privát adattagjaiban eltárolja:

- az adott sort és oszlopot, amiben elhelyezkedik
- hány bomba van körülötte
- bomba-e
- van-e rajta jelölés

- **public void increaseCount()**

  Növeli azt, hogy hány bomba van a mező körül.

class Board: Egy panel, magát a játékteret valósítja meg. Privát adattagjaiban eltárolja:

- zászlók számát, amennyit használhat a játékos
- sorok számát
- oszlopok számát
- bombák számát
- listát, ami a mezőket tárolja (class Tile)

- **public void generateMines()**

  Véletlenszerűen legenerálja a bombák helyét a táblán. A kisorsolt mezők isMine (bomba-e) attribútumát igazra állítja.

- **public void countMines()**

  Megszámolja, hogy egy mező körül hány bomba található.

- **public void updateCount(int r, int c)**

  Segédfüggvény a countMines-hoz, ami végigmegy a szomszédos mezőkön és megnézi, hogy bomba-e az adott mező.

- **public void gameOver()**

  Felfedi az összes mezőt, ha a játékos bombára kattint

- **public void fill(int r, int c)**

  Rekurzív függvény, felfedi azokat a mezőket és szomszédjait, amiknek a szomszédjában nincs bomba

- **public void flagnum(Tile t)**

  Ha a játékos használja a zászlókat, akkor csökkenti/növeli a számukat és megjeleníti

- **public void win()**

  Megvizsgálja, hogy a játékos nyert-e, azaz csak a bombák fedettek.

- **public void mousePressed(MouseEvent e)**

  Az egérrel való kattintást kezeli le ez a függvény. Csak akkor kezeli a kattintásokat amikor fut a játék.

  A bal klikk lehetséges kimenetelei (csak akkor nézzük a kattintást ha az adott mező nincs megjelölve):

  - első kattintás, ilyenkor a timer tudja, hogy elkezdhet számolni
  - ha bombára kattint a játék véget ér, különben tovább fut a játék

  A jobb klikk lehetséges kimenetelei:

  - ha megjelöl egy fedett mezőt akkor azt megjeleníti
  - ha megszüntet egy jelölést, akkor eltávolítja

  Eközben a program végig figyeli, hogy nyert-e a játékos.

class Stopper: Az időért felelős osztály, feladata kiszámítani az eltelt időt a játék kezdete óta, hogy nyerés esetén a program tudja, hogy mennyi idő alatt sikerült megnyerni a játékot. A privát adattagjában tárolja ezt az időt.

- **public void timer(Date date, Date startDate)**

  Kiszámítja a kezdés óta eltelt időt.

class Leaderboard: A ranglistát kezelő osztály, továbbá ez az osztály felelős a ranglista megfelelő megjelenítéséért is. Privát adattagjaiban eltárolja:

- aktuális nehézségi szintet
- a tábla fejlécét
- a tábla adatait

- **public static void filein(String diff)**

  Beolvassa a megfelelő .txt fájlból az adatokat.

- **public static void fileout()**

  Kiírja a megfelelő .txt fájlba az adatokat.

class Menu: A játék indítása után megjelenő menürendszerért felelős osztály. Privát adattagjaiban eltárol pár megjelenítéshez szükséges GUI elemet, továbbá:

- az adott időt, ha a játékos nyert
- azt, hogy az első kattintás megtörtént-e
- azt, hogy a játék véget ért-e
- azt, hogy a játékos nyert-e
- a pálya létrehozásához szükséges adatokat (pálya nagysága, bombák száma)
- játékos nevét

- **public void arrange()**

  A ranglistát az elért idő szerint rendezi növekvő sorrendbe és elmenti az adott .txt fileba.

- **public void reset()**

  Új játéknál alaphelyzetbe állítja a szükséges változókat.

- **public void mousePressed(MouseEvent e)**

  Az egérrel való kattintást kezeli le ez a függvény, ahogy a menüben navigál a felhasználó.

  A bal klikk lehetséges kimenetelei:

  - Exit esetén kilépés, adatok elmentése.
  - Új játék esetén átnevezi a gombokat, hogy ki lehessen választani a nehézségi szintet, de előbb meg kell adnia a felhasználónak egy nevet, és csak ez után lehet nehézségi szintet választani. a választás után megjelenik a megfelelő játéktér.
  - A ranglista megnézéséhez először ki kell választani a megtekinteni kívánt ranglistát. A kiválasztás után megjeleníti a megfelelő ranglistát. Ilyenkor az ablak alján lévő Back gombra kattintás esetén visszalép a főmenübe (átnevezi a gombokat)
  - A játék közben a fejlécben található Retry gombot is a Menu kezeli

- **public void keyPressed(KeyEvent e)**

  Az Enter megnyomásával lehet elmenteni a nevet, ilyenkor elérhetővé válik a nehézségi szint kiválasztása.

A megjelenítéshez használt függvényeket nem részletezem, mivel ezek nem tartoznak a program főbb függvényei közé. A programkódban ezekhez a függvényekhez is írtam kommentet.

class Runner: A program futásáért és az idő megjelenítéséért felelős. Létrehozza a játék futtatásához szükséges osztályokat, változókat. Továbbá nyerés esetén belerakja a ranglistába az elért eredményt névvel együtt, ha az jogosult az ottlétére.

4. Tesztprogram

A tesztelő osztály JUnit segítségével valósul meg. A program futása szempontjából fontos függvényekre, metódusokra van teszteset. Az osztályban létre van hozva pár, a teszteléshez szükséges osztály,változó, melyeket a tesztesetek használnak fel.

- **public void testStopperTimer()**

  Leteszteli, hogy a stopper tényleg másodpercenként vált-e a Stopper osztály segítségével.

- **public void testLeaderboardExeption()**

  Teszteli, hogy ha a Leaderboardnak hibás .txt fájlnevet adnánk meg, akkor megfelelő errort kapnánk-e. Mivel a programnak adott fájlt kell megnyitni, így ez az eset csak akkor áll fent, ha az adott fájl eltűnik a programfáljokból.

- **public void testLeaderboardFilein()**

  Ellenőrzi, hogy sikeres volt-e a fájlból való beolvasás.

- **public void testLeaderboardFileout()**

  Teszteli, hogy tényleg megtörtént-e az adatok fájlba való kiírása és annak sikerességét is vizsgálja.

- **public void testBoardMinenum()**

  Ellenőrzi, hogy a Board osztály konstruktora megfelelően adta-e át a megadott adatokat a privát adattagoknak, jelen esetben bombaszámot.

- **public void testFlagnum()**

  Vizsgálja, hogy a program a bombaszámból át tudta-e kalkulálni, hogy mennyi a használható zászlók száma

- **public void testBoardMinecount()**

  Teszteli, hogy sikeres volt-e a mezőket megjelölni, hogy azokon a helyeken helyezkednek el a bombák

- **public void testBoardCountMines()**

  Ellenőrzi, hogy megfelelően működik-e annak a kiszámítása, hogy hány bomba van az adott mező körül

- **public void testBoardaftergameover()**

  Vesztés esetén vizsgálja, hogy valóban felfedésre került-e az összes mező a gameOver függvény segítségével.

- **public void testMenuArrange()**

  Megnézi, hogy valóban jól működik-e a ranglista növekvő sorrendbe helyezése időeredmény alapján az arrange függvény segítségével.

Az adott tesztesetek végrehajtása után a tesztelő osztály az undo függvény segítségével visszavonja a módosításokat, amelyeket a metódusok megfelelő teszteléséhez kellett végezni.