Dokumentáció

A feladat szövege:

Készítsünk programot, amellyel a közismert amőba játék következő változatát játszhatjuk. Adott egy n × n-es tábla, amelyen a két játékos felváltva X, illetve O jeleket helyez el. Csak olyan mezőre tehetünk jelet, amely még üres. A játék akkor ér véget, ha betelik a tábla (döntetlen), vagy valamelyik játékos kirak 5 egymással szomszédos jelet vízszintesen, függőlegesen vagy átlósan. A program minden lépésnél jelezze, hogy melyik játékos következik, és a tábla egy üres mezőjét kijelölve helyezhessük el a megfelelő jelet. A kiszúrás a játékban az, hogy ha egy játékos eléri a 3 egymással szomszédos jelet, akkor a program automatikusan törli egy jelét egy véletlenszerűen kiválasztott pozícióról (nem biztos, hogy a hármasból), ha pedig 4 egymással szomszédos jelet ér el, akkor pedig kettőt. A program biztosítson lehetőséget új játék kezdésére a táblaméret megadásával (6×6, 10×10, 14×14), és ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hogy melyik játékos győzött (ha nem lett döntetlen), majd kezdjen automatikusan új játékot.

A megoldás terve:

Maga a feladat egy nehezített tic-tac-too állítható pályamérettel. A feladatban fontos megoldás volt az hogy hatékonyan tudjuk ellenőrizni azt hogy hány elem van egymás mellet úgy hogy nyerjünk vagy hogy büntetést kapjunk. A feladatban minden elem gombbal van ábrázolva és hogy frissíteni tudjuk a büntetésnél a Gridet így mindig újra rendereljük minden egyes gomblenyomás után hogy ki tudjuk rajzolni a frissített elemeket.

Ha nincs győztes akkor azt kellene ellenőrizni hogy maradt e olyan gomb ami üres és ha igen akkor döntetlen lett az állás.

A megoldáshoz tartozik a pálya, egy ablak ugrik fel ahol kiválaszthatjuk hogy mekkora pályán szeretnénk játszani ami egy újabb ablakban jelenik meg maga a pálya amin játszhatunk és vége játéknál ezzel a pályamérettel indul újra a játék.

Metódus leírások:

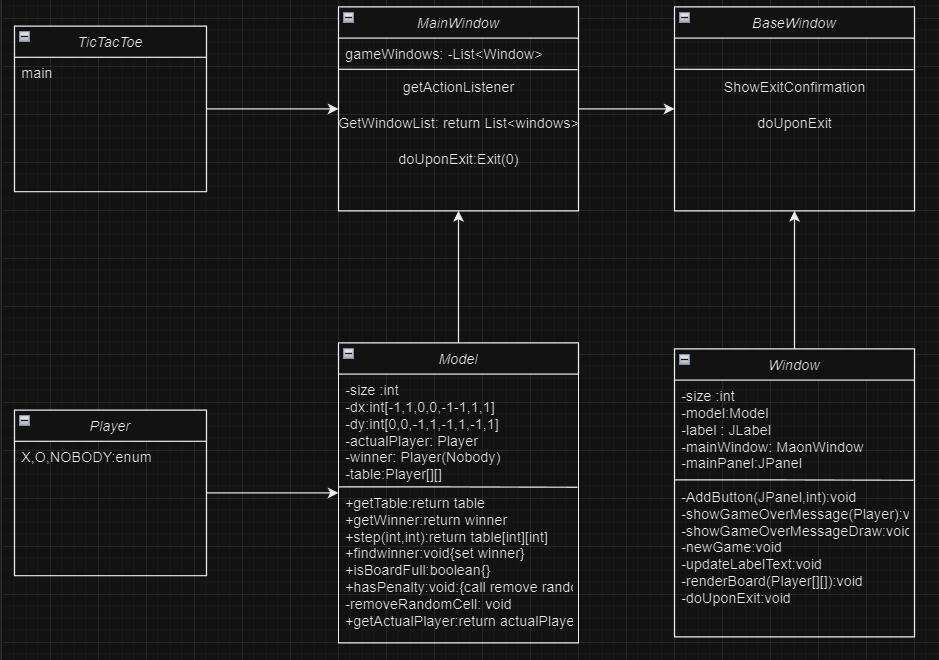
-Modell osztály:

* getTable: visszaadja a játékos táblánkat amiben nyomon követjük a pálya állását.
* getWinner: visszaadja a nyertes játékosunkat
* step: a lépésért felel ahol meghívjuk hogy van e nyertes a lépés után és hogy kap e büntetést
* findWinner: a step függvényben van meghívva hogy a lépésnél aki lépett az nyert e
* isBoardFull: azt ellenőrizzük a függvénnyel hogy a pálya tele van e
* hasPenalty: a büntetést ellenőrizzük, hogy elért e már 3 vagy 4 egymás melletti mezőt és ha igen akkor a removeRandomCell fügvénnyle töröljük a megfelelő számú mezőnket.
* removeRandomCell: törlünk egy véletlenszerüen választott mezőst a sajátjaink közül
* getActualPlayer: visszaadjuk hogy ki az aktuális játékos

-Window osztály:

* addButton:hozzáadjuk a gombokat a játékmezőhöz
* showGameOverMessage: a nyertes nevét kiírjuk
* showGameOverMessageDraw: döntetlent írunk ki
* newGame: új játékot hozunk létre
* updateLabelText: frissítjük az éppen aktuális játékos nevét
* renderBoard: rendereljük a griden található játékmezőt
* doUponExit: kilépés esetén mit csinálunk

Osztálydiagramm:



Esemény-Esemény kezelő leírása:

* Window actionPerformed: meghívjuk a newGame metódust hogy új játékot kezdjünk
* addButton actionPerformed: ebben az eseménykezelőben kezeljük le a mezők helyes működését és pálya újragenerálását lényegében az egész játék minden gombba van belekódolva
* doUponExit: Megadjuk hogy mit csináljon a program kilépésnél, ami a program leállítását eredményezi
* MainWindow actionPerformed: Új játékablak létrehozása.
* BaseWindow windowClosing: megjelenik a kilépés megerősítése ablak ha megpróbálunk kilépni

Tesztelési terv:

|  |  |  |
| --- | --- | --- |
| Teszteset rövid leírása | Teszt bemenet | Elvárt kimenet |
| Van három szimbólum egymás mellett | x,x,x  o,o,nobody | Kitörlődik egy mező |
| Van négy ugyanolyan szimbólum egymás mellett | x,x,x,x  o,o,o,nobody | Törlődik két mező |
| Van öt ugyanolyan szimbólum egymás mellett | x,x,x,x,x  o,o,o,o,nobody | Megkapjuk hogy ki nyert egy üzenetablakban |
| Már foglalt mezőre kattintunk | x,x,x | Nem történik semmi |
| Megtelt a pálya |  | Döntetlen üzenet üzenetablakban |