Prog3 NHF

A programozás alapjai 3 - NHF

Tartalom

Földi Balázs
2023.10.11.

I. SPECIFIKÁCIÓ

A feladat egy grafikus felülettel rendelkező amőba játék készítése.

1. Főmenü

A játék indításakor egy főmenüvel találkozunk. A főmenüben 3 opció lesz elérhető,

- a „*Start Game*”,
- „*Scoreboard*”
- és az „*Exit*” gombok.

Az utóbbi kilép a játékból, a „Scoreboard” pedig fájlból betölti az eddigi összes játékos pontszámát. Ha egy játékos nyer, az 2 pont, ha döntetlen, akkor mindkét játékos kap 1-1 pontot, vereség esetén pedig 0 pontot kap a vesztes fél. Pontokat csak „Player vs. Player” módban lehet szerezni.

2. Játékmódok

A „Start Game” opció választása után egy újabb menü jelenik meg, ahol kiválaszthatjuk, hogy ki ellen szeretnénk játszani:

- „*Player vs. Player*”, vagy
- „*Player vs. AI*”.

Az elsővel egy másik játékos ellen játszhatunk, ugyan azon kijelző előtt. Ha ezt az opciót választjuk, akkor megjelenik két szövegdoboz, ahova a játékosok beírhatják a nevüket, ami majd a Scoreboardban megjelenik.
Ha a „Player vs. AI” -t választjuk, akkor megjelenik egy újabb menü, ahol az AI nehézségét tudjuk kiválasztani:

- „*Difficulty 1*” és
- „*Difficulty 2*”.

Majd mielőtt ténylegesen kezdetét venné a játék, minden esetben megjelenik egy felület, ahol beállíthatjuk és a tábla méretét. Alapértelmezetten ez 3x3-as, de van lehetőség 5x5-ös és 7x7-es táblán is játszani.
Ha a játéknak vége, akkor a program visszaugrik a főmenübe. Amíg tart egy játszma, addig új játékot nem lehet kezdeni. Tehát ha játék közben lépünk ki, akkor az állás mentésre kerül és következő indításkor automatikusan az töltődik be.

3. Szabályok

A játékosok felváltva rakják le a O vagy X alakú jelölőjüket.
Mindig az X játékos kezd, AI ellen mindig O az AI.
A nyeréshez 3x3 tábla esetén 3 azonos jelölőt kell egymás mellé, vagy átlósan lerakni, míg a másik kettőben négyet.
A játékosok nem rakhatják a jelölőjüket olyan mezőbe, ahol már a másik játékosé szerepel. Ha ezt megpróbálják, akkor egyszerűen a program nem fogja engedni, viszont a felhasználó nem lát majd semmilyen hibaüzenetet.
A játéknak akkor van vége, ha valakinek sikerül megfelelő sorrendbe leraknia a jelölőit, vagy ha elfogynak az üres cellák, ekkor a játék döntetlennel ér véget.

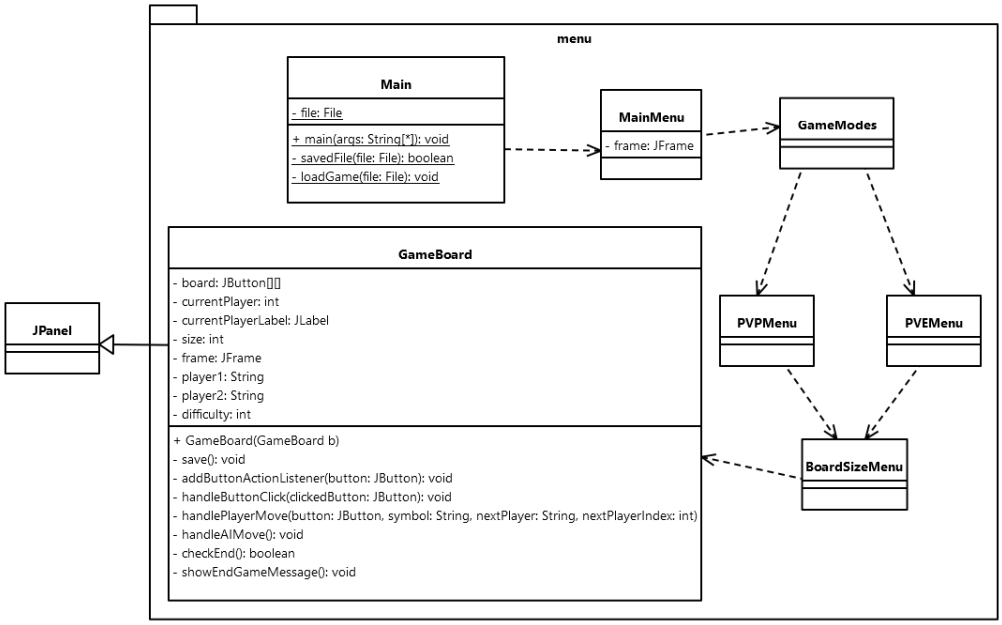
4. Technikai elemek

A program a grafikus megjelenítéshez SwingGUI-t fog használni[1]. A Scoreboard mindig pont szerint rendezetten fogja mutatni az aktuális állást, ezt a rendezést java.util-beli osztályok segítségével érem majd el. A program a játék állását kilépéskor fájlba menti, illetve a scoreboard is egy fájlban fog tárolódni. A játék állását minden indításkor megpróbálja majd betölteni, a scoreboard-ot csak akkor, ha a felhasználó szeretné azt megtekinteni. A program tesztélést JUnit segítségével fogom megvalósítani[1].

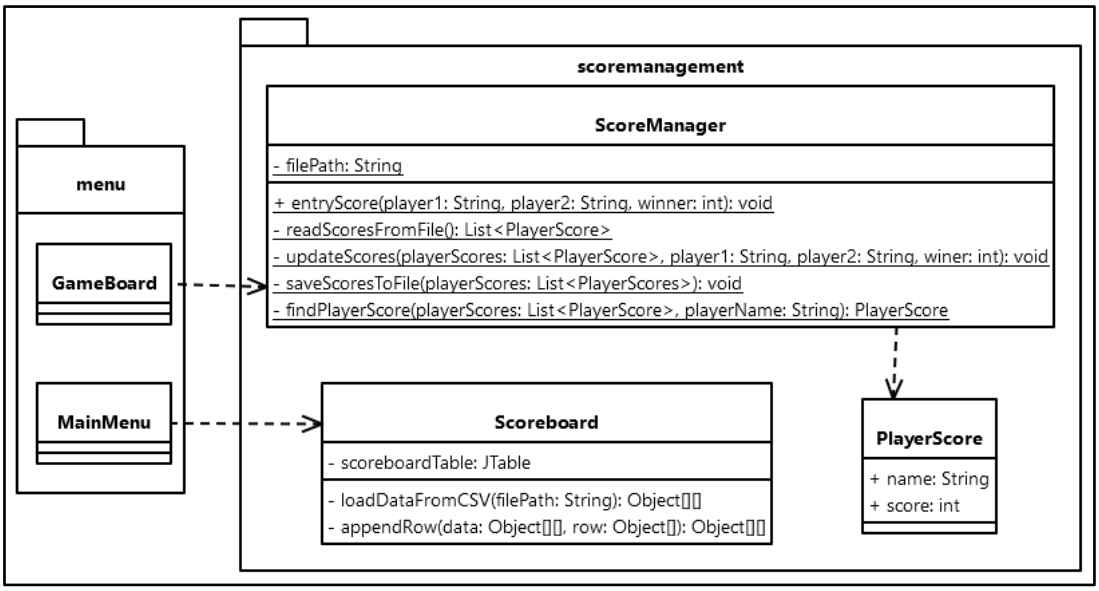
[1] Ezekről majd az előadások után, mikor lesz róluk ismeretem, még bővebben írok.

II. FEJLESZTŐI DOKUMENTÁCIÓ

A játék elkészítésénél az osztályokat a logikai összetartozásuk végett és a jobb átláthatóság miatt 4 csomagba rendeztem. Ennek a 4 csomagnak az osztálydiagramjai láthatóak az alábbi képeken.[2]

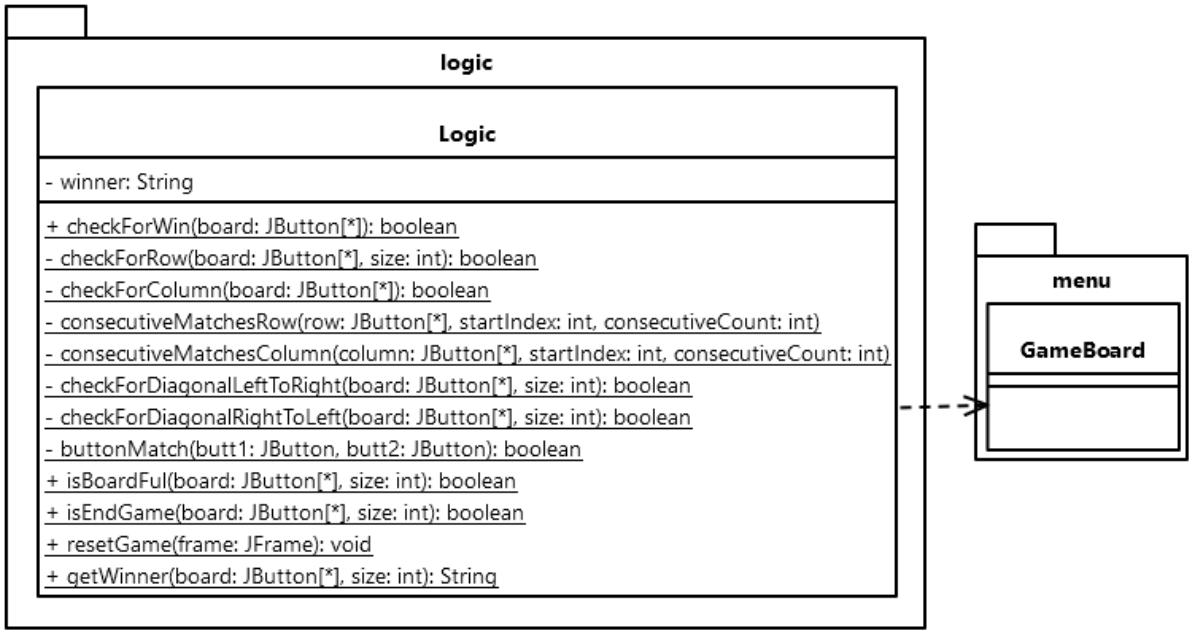


1. ábra: menu package

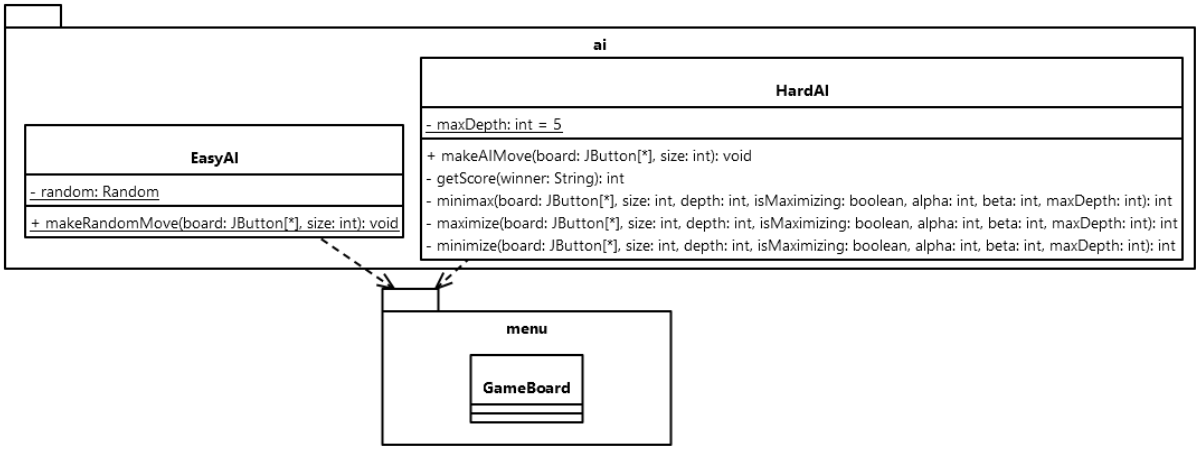


2. ábra: scoremanager package

[2] A diagramokon ahol paraméterként „board: JButton[*]” áll, az „board: JButton[][]”. valamiért a program összevonta ezeket. De ez csak ott fontos, ahol „board” a paraméter neve. Máshol csak „JButton[]”



3. ábra: logic package



4. ábra: ai package

Az **1. ábra** tartalmazza a *menu* nevű csomagom osztálydiagramját. A Main osztály felelős a program indításáért és az esetleges, korábbi mentés betöltéséért. A savedFile függvény segítségével ellenőrzi, hogy létezik-e mentés, ha pedig létezik akkor a loadGame-mel betölti. Ebben az esetben rögtön meghívja a GameBoard osztály copy konstruktorát, és megy is tovább a játék. Minden más esetben egy hosszú lánc vezet a GameBoard osztályig, ami ténylegesen elindítja a játékot. A diagramon látszik, hogyan megyünk végig ezen a láncon, minden „láncszem" meghívja a következő konstruktorát, így változtatva a menüket. A GameBoard osztály a JPanel leszármazottja, ezért implementálja a Serializable interfészt, így a fájlba mentése egyszerűvé válik. Ezt az osztály save metódusa végzi el. Ez az osztály építi fel a játéktáblát is, ezt JButton elemekkel teszi. Az addButtonActionlistener a paraméterként kapott JButton elemre elvégzi az event kezelését. Ebben segít a handleButtonClick, handlePlayerMove, handleAIMove, checkEnd és showEndMeddage metódusok. Az utóbbi végzi el a játékosok felvételét a rekordtáblába.

A **2. ábrán** látható a *scoremanager* csomag felépítése. Az egyik osztálya a Scoreboard. Ő felelős a főmenüből megjeleníteni és fájlból beolvasni a rekordtáblát. 2 tagfüggvénye van: az első függvény loadDataFromCSV felelős a CSV fájl beolvasásáért, míg a második függvény, az appendRow, az adatsorok hozzáadását végzi a táblázathoz. A loadDataFromCSV függvény egyszerűen végig olvassa a CSV fájlt soronként, majd a split függvény segítségével szétválasztja a sort a pontosvessző karakter alapján. Ha a szétválasztott részek száma pontosan 2, akkor egy új Object tömbbe helyezi ezeket az értékeket, és a appendRow függvény segítségével hozzáadja ezt a sort a táblázathoz. Ha a részek száma nem 2, akkor a konzolra kiír egy hibaüzenetet. Végül a függvény visszaadja a feltöltött adatokat. Az appendRow függvény létrehoz egy új tömböt, amely egy sorral hosszabb, mint a meglévő adattáblázat. A System.arraycopy függvény segítségével másolja át az eredeti adatokat az új tömbbe, majd az utolsó helyre beszúrja az új sort. A függvény visszaadja az új adattáblázatot.
A másik osztály a ScoreManager. Ez az osztály felelős annak a függvénynek (entryScore()) a működéséért, amivel új rekordokat vehetünk fel, vagy frissíthetjük azok pontszámát. Az átláthatóság és kognitív komplexitás csökkentése érdekében további privát tagfüggvényeket használ. Az updateScores() metódus felel azért is, hogy az adatok pontszám szerint rendezve legyenek, azon belül pedig név szerint. A privát a függvények gyakran használnak egy PlayerScore nevű osztályt. Ez egy roppant egyszerű osztály: 2 adattagja van, egy név és a hozzá tartozó pontszám, illetve setter-ek és getter-ek. A Gameboard osztály a ScoreManager osztályt használja, míg a Mainmenu a Scoreboard osztályt.

A **3. ábra** tartalma a *logic* csomag felépítése. Ennek az egyetlen eleme a Logic nevű osztály, amiben alapvetően olyan függvények vannak implementálva, amelyek a játék szabályainak a „betartását" végzik. Ezeket a metódusokat főként a GameBoard osztály használja. A tagfüggvényeit egyszerű algoritmusok valósítják meg, amik további magyarázatra nem szorulnak.

A **4. ábra** az *ai* csomag felépítését szemlélteti. Ebben két osztály található.
Az EasyAI egy egyszerű osztály. Egyetlen metódusával képes egy véletlenszerűen kiválasztott helyre rakni a jelölőjét.
A HardAI az egész projekt legkomplexebb osztálya. Az egyetlen publikus metódusát a GameBoard osztály hívja meg. Ez a függvény felelős a 2-es nehézségű AI lépéseiért. A lépést egy „minimax algoritmus" alapján számolja. Röviden: „A minimax elv a döntéselméletben, a játékelméletben és a statisztikában alkalmazott döntési szabály, ami szerint azt a lehetőséget kell választani, ami minimalizálja a maximális veszteséget. Felfogható a minimális nyereség maximalizálásaként is." (Wikipédia, Minimax elv, 2023)

```
if(size == 3) maxDepth = Integer.MAX_VALUE;
if(size == 5) maxDepth = 6;
if(size == 7) maxDepth = 5;
```

5. ábra

A minimax tulajdonképpen egy rekurzív algoritmus a következő lépés meghatározására. A játékosok száma többnyire kettő. A játék minden állásához tartozik egy érték, amit egy kiértékelő függvény (getScore) segítségével számolunk. Ez jelzi, hogy mennyire kedvez egy játékosnak az az állás. A soron következő játékos maximalizálja az ellenfelek lépései után elérhető minimális értéket. A függvény a győzelmet 1-nek, döntetlent 0-nak, a vereséget pedig -1-nek értékeli ki. Az algoritmus a játékfa csúcsainak felkutatásaként képzelhető el. Az egy állásban megtehető lépések átlagos száma jó elágazási tényező. A csúcsok száma rendszerint exponenciálisan nő, ezért nem hatékony a teljes játék végigelemzéséhez. A komplexitás nagyjából $O(n^{2d})$ egy amőba játék esetén, ahol az n a tábla mérete (jelen esetben 3, 5, vagy 7), a d pedig a maximális mélység. A „makeAIMove" függvény elején ez a paraméter a 5. ábrán látható módon van definiálva. Ez igazából teljesen heurisztikus, tesztelgettem és úgy döntöttem, hogy ilyen paraméterek mellet az adott tábla méretéhez képest még tűrhető időn belül lép az algoritmus. Ezen paraméterek mellett még mindig rengeteg idő lenne a kiértékelés, ezért az algoritmus alfa-béta vágást használ, aminek „alapötlete azon nyugszik, hogy ha a játékfában az éppen vizsgált lépésünkre az ellenfélnek van egy olyan erős lépése, ami miatt ezt a lépést úgyse választanánk (mivel a vizsgálat korábbi részéből már van jobb választásunk), akkor az erre a lépésre az ellenfél által adható további lépéseket nem szükséges megvizsgálni. (Más szóval: ha az ellenfél válaszlépése túl jó, akkor úgyse fogjuk meglépni az azt lehetővé tévő lépésünket.) Az algoritmusban ezen részjátékfák fölösleges vizsgálatának kihagyását hívjuk alfa-, illetve béta vágásnak. Az algoritmus a játékfa bejárása közben két értéket tart karban, az ún. alfa és béta értéket, az alfa azt a minimum értéket jelenti, amit az ún. 'maximalizáló' játékos már biztosított magának, illetve a béta azt a maximum értéket, amit a 'minimalizáló' játékos biztosított magának. Az algoritmus induláskor a két értéket mínusz végtelen, illetve plusz végtelenre inicializálja, majd a fa rekurzív vizsgálata közben folyamatosan állítgatja értéküket a két játékos által garantáltan elérhető értékekre. Ezáltal folyamatosan szűkül ez az alfa-béta ablak. Amint a béta kisebbé válik az alfánál, az azt jelenti, hogy ez az állás – ha mind a két játékos részéről a legjobb játékot tételezzük fel – nem állhat elő, így nincs is értelme tovább vizsgálni." (Wikipédia, Alfa-Béta vágás, 2023) Ezt az algoritmust valósítja meg a minimax nevű, privát függvény, ami meghívja a maximizie és minimize függvényeket a fentieknek megfelelően. Az algoritmus a 3x3-as táblán tökéletesen működik, míg a 7x7-esen már követ el triviális hibákat, a maximális mélység miatt. Viszont, ha ez nem lenne, akkor nagyon sok ideig tartana a kiértékelés. Ezek a 3 opciónak megfelelően a következőképpen alakulnak/alakulnának, ahol d a maximális, a kódban is implementált mélység.[3]

n	d	n^{2d}	$n^{2(d+1)}$
3x3	Nem érdemes vizsgálni, mivel optimális a működése		
5x5	6	244.140.625	≈6 millió
7x7	5	282.475.249	≈13 millió

[3] Az algoritmus implementálásához használt források:

- Tic Tac Toe: Understanding the Minimax Algorithm – Never Stop Building
- Finding optimal move in Tic-Tac-Toe using Minimax Algorithm in Game Theory – Geeks for Geeks
- Coding Challenge 154: Tic Tac Toe AI with Minimax Algorithm – The Coding Train (YouTube)

A tesztek implementálását a Java JUnit környezetével végeztem el. Összesen 10 metódus van tesztelve. A HardAI osztályból 1 a makeAiMove függvény, a ScoreManager osztályból az entryScore metódus, a maradék 8 pedig a Logic osztályból kerültek ki.

A makeAIMove függvény tesztje létrehoz egy táblát, ahol a következő körben az X (tehát nem az AI) nyerne, ez az actualBoard nevű tábla. Létrehoz egy expectedBoard nevű táblát, ahol az X-ek végén egy O áll, mivel ez a legoptimálisabb eset, mivel ekkor nem veszít az AI. Az actualBoard-ra meghívjuk a makeAIMove metódust, majd a végén ellenőrizzük, hogy a két tábla megegyezik-e.

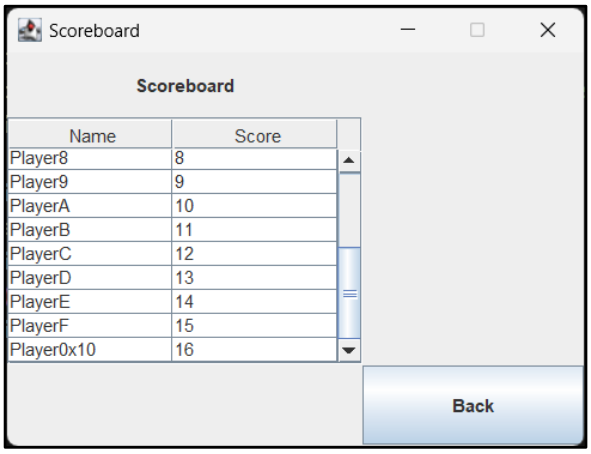
Az entryScore függvény tesztelésénél létrehozunk 2 új bejegyzést a függvény meghívásával, a test.csv fájlba, amit a teszt végén törlünk. Mindkét függvényhívás paraméterei megegyeznek, így a várt kimenet, hogy valakinek 4, míg a másiknak 2 pontja van.

A Logic osztályban a checkFor kezdetű függvények is tesztelve vannak többek között. Ezek algoritmusa nagyon hasonló. Létrehoz egy táblát, ahol a függvénynek megfelelően létrehoz egy alakzatot (sor, oszlop, átlók), majd ellenőrzi, hogy a függvény igaz értékkel tér-e vissza.
Van teszt a buttonMatch-re. Ez létrehoz 2 gombot különböző felirattal, így a várható kimenetel a hamis érték, ezt vizsgálja a tesztfüggvény. Az isBoardFull és isEndGame tesztjei létrehoznak egy telített táblát, a végén pedig vizsgálják, hogy a függvény kimenetele igaz-e. Ugyan ezen az elven működik a getWinner függvény tesztje. Itt annyi a különbség, hogy a végén azt vizsgáljuk, hogy a függvény a megfelelő szimbólummal tér-e vissza (a teszteset környezte miatt ez O).

III. FELHASZNÁLÓI KÉZIKÖNYV

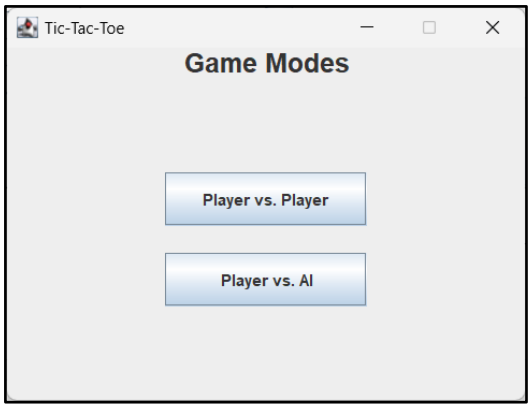
A felhasználó a legelső programindításkor a 6. ábra által szemléltetett felülettel találkozik.

6. ábra

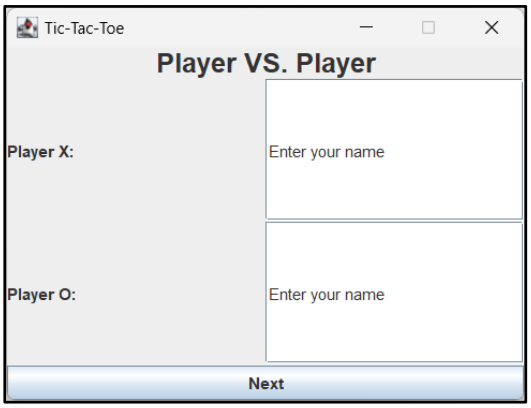


7. ábra

Innen a legrövidebb úttal rendelkező választás az „Exit”, ekkor ugyanis a program bezáródik. Ha a felhasználó a „Scoreboard” opciót választja, akkor a 7. ábrán látható felület fogadja, ahol a rekordtáblát lehet megtekinteni. A táblázat görgethető az annak jobb oldalán található csúszkával. Ha a felhasználó megnyomja a „Back” feliratú gombot, akkor újra a 6. ábra felülete jelenik meg.
Amennyiben a játékos a „Start Game” opciót választotta, akkor a 8. ábra felülete jelenik meg. Itt válik szét jobban a menürendszer. Először azt az utat járjuk be, ahol a felhasználó a „Player vs. Player” feliratú gombot választotta. Ezt követően a 9. ábra felület kell látnia. Ekkor más valaki ellen játszik.

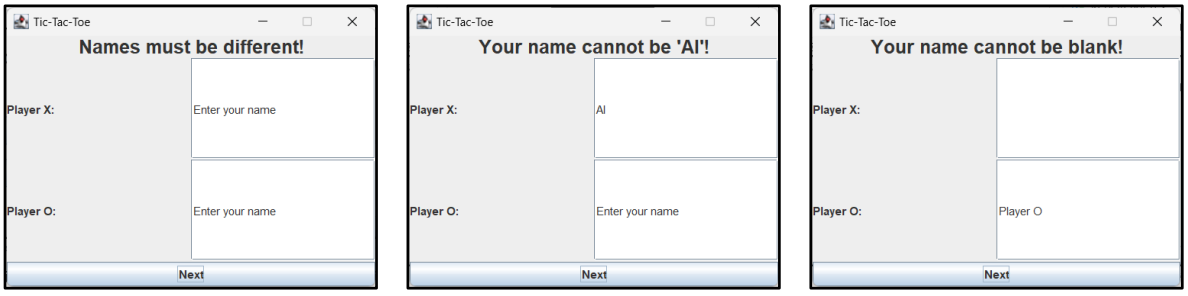


8. ábra



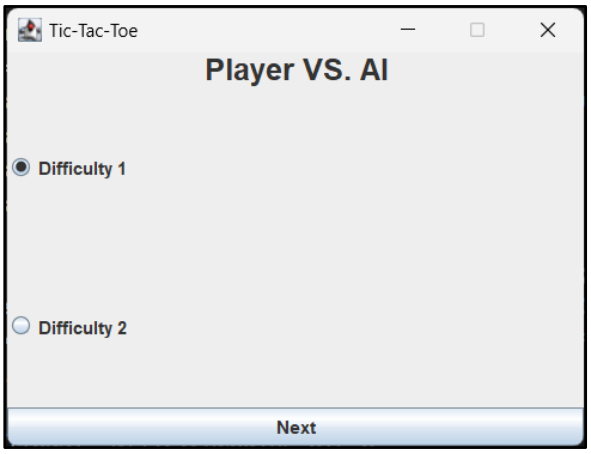
9. ábra

Itt adhatják meg a játékosok a neveiket. A program (ignorálva a kis- és nagybetűket, illetve a szóközöket) nem engedi az a következő három esetet: Ha a két játékos neve megegyezik, ha valakinek „AI”, vagy ha üres a szövegdoboz. Erre példák a 10. ábrán lathatóak, a fentebb is említett sorrendben.

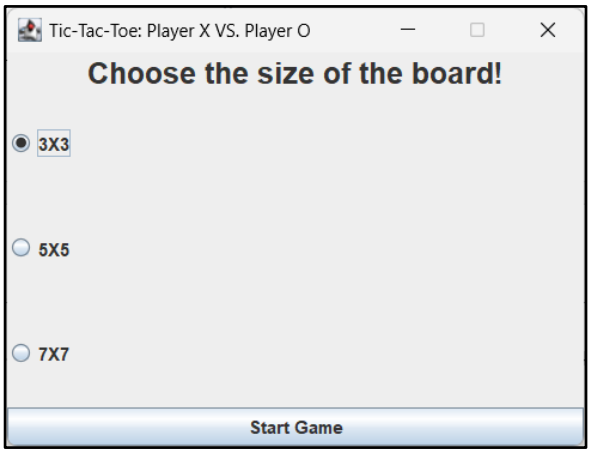


10. ábra: Exceptions

Miután sikerült két helyes nevet megadni, - amik majd megjelennek a rekordtáblában is – akkor a „Next” feliratú gomb megnyomását követően, a felhasználó a 12. ábra felületét kell, hogy lássa. Ugyan ezt a felületet kell majd látnia, ha a „Player vs AI” feliratú gombot választotta a 8. ábra menüjéből, viszont előtte a 11. ábra menüjével találkozik. A 11. ábra felületén a játékos ki tudja választani, a számítógép nehézségi szintjét, ugyanis ilyenkor a gép lesz a másik játékos. Alap esetben ez a Difficulty 1. Miután sikerült kiválasztani és rányomtunk a „Next” gombra, akkor innen is a 12. ábra felületére lyukadunk ki. Itt kiválaszthatjuk, hogy mekkora méretű táblán szeretnénk játszani, ez alapesetben 3x3. Ha ezt kiválasztottuk kezdődhet is a játék, ugyanis a következő felület, amit látnunk kell már maga a játéktábla. Ezt a felületet látjuk a 13. ábrán is, miután megnyomtuk a „Start Game” gombot. (A példán egy 3x3-as tábla). A játékot mindig az X szimbólumot használó játékos kezdi. Amikor a másik játékos az „AI”, akkor az mindig a O szimbólummal játszik.



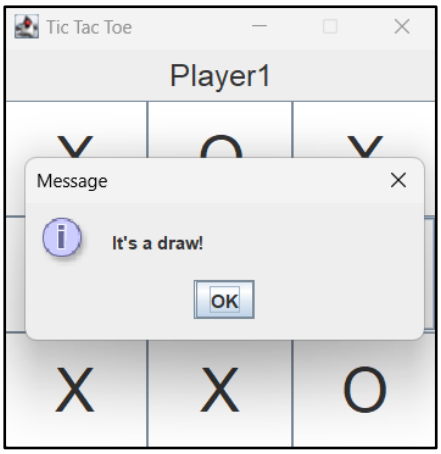
11. ábra: Difficulty menu



12. ábra: Boardsize menu

13. *ábra: Board GUI*

A játékmenet kezelése egyszerű és egyértelmű. A játékosok felváltva (az ablak tetején is jelzett névnek megfelelően) kiválasztanak egy rubrikát, amire rákattintanak. Ha ez bármilyen okból szabálytalan lenne, akkor nem kapunk semmilyen visszajelzést, a lépés egyszerűen nem kerül végrehajtásra. A játék kétféleképpen érhet véget. Ha elfogy minden mező és már nincs hova rakni jelölőt, akkor döntetlen. A másik eset, ha valaki megnyeri. Erre példák a 14. és 15. ábrák. Miután a felhasználó rákattint az „OK” feliratú gombra, máris újra a főmenüben találja magát. Ha valamelyik fél játék közben bezárja az ablakot, akkor a játék állása mentésre kerül, amikor a következő indításkor betöltődik. (Ez azt jelenti, hogy a felhasználó nem a 6. ábra menüjét, azaz a főmenüt fogja látni, hanem a félbehagyott játék tábláját. Ha ez a játék elején történt, még nem történt lépés, és 3x3-as volt a játéktábla, akkor a 13. ábra felületéhez hasonlót kell látnia.) Miután a játék véget ért és nem a számítógép ellen történt a játék, úgy a rekordtáblába bekerülnek a megfelelő nevek a megfelelő pontokkal. A nyertes 2 pontot kap, a vesztesnek pedig 0 pont jár. Amennyiben a játék döntetlennel zárult, úgy mindkét fél 1-1 pontot kap. Ezek az új/frissített rekordok rögtön már a játék után látszanak a 9. ábra felületén.



14. ábra: Draw

15. ábra: PlayerD wins