

II. Beadandó

Habzda Fruzsina (XNUHTE)

Tartalomjegyzék

Feladat.....	3
Tervezés.....	3
Specifikáció.....	3
Osztály diagram	4
Részletezett UML diagrammok	4
Megvalósítás.....	6
Főprogram	6
GameLogic	6
GameBoard.....	11
FieldButtonActionListener.....	12
Tesztelés	13
Elvégzett tesztesetek.....	13

Feladat

4. feladat: Kiszúrós amőba

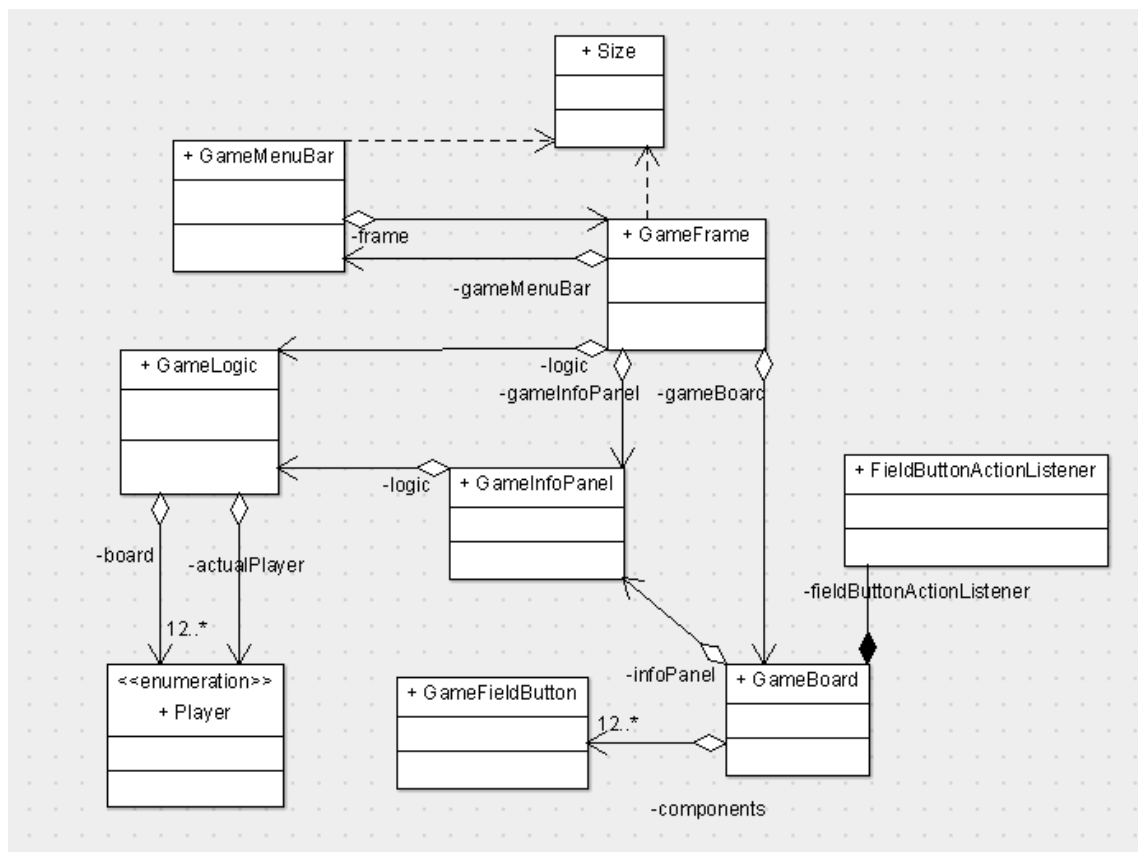
Készítsünk programot, amellyel a közismert amőba játék következő változatát játszhatjuk. Adott egy $n \times n$ -es tábla, amelyen a két játékos felváltva X, illetve O jeleket helyez el. Csak olyan mezőre tehetünk jelet, amely még üres. A játék akkor ér véget, ha betelik a tábla (döntetlen), vagy valamelyik 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 hármashból), 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.

Tervezés

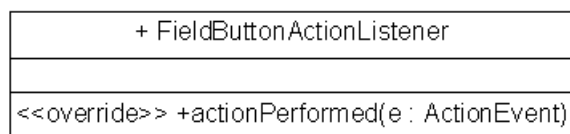
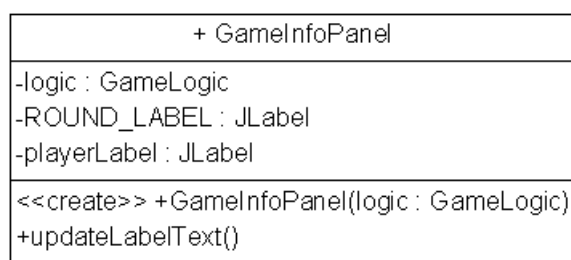
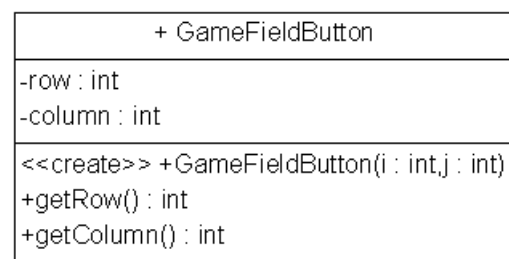
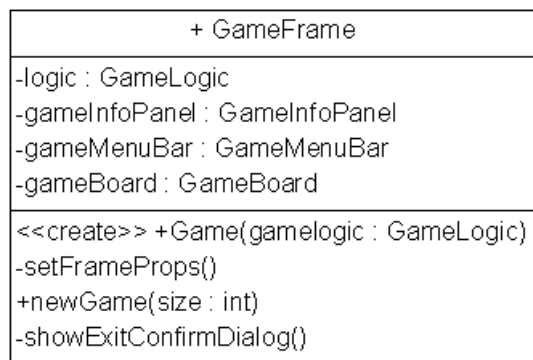
Specifikáció

A game mappán belül két package lett létrehozva. A gui, mely tartalmazza a GameFrame.java, GameBoard.java, GameFieldButton.java, GameInfoPanel.java, GameMenuBar.java állományokat, és a logic mely tartalmazza a GameLogic.java, Player.java, Size.java állományokat. A GUI alapja a GameFrame, amit JFrame-ből származtattam. Tartalmaz egy menüt (GameMenuBar, JMenuBar-ből), egy információs panelt (GameInfoPanel, JPanel-ből), és a játék táblát (GameBoard). A menüből új játék indítható, választható táblamérettel. Az információs panel kiírja a soron lévő játékos (logic.actualPlayer) jelét. (Mindig X kezdi a játékot, a játék 6*6-os táblával indul.) A játék táblán gombok (GameFieldButton) vannak elhelyezve, melyek először üres mezőket reprezentálnak, majd ha a soron következő játékos rákattint, annak a jelét jelenítik meg. Két féle játékkal lehet játszani (Player.X, Player.Y). A táblaméret konstans változóit a Size nevű osztályban tárolom (6 féle van: SMALL, MEDUIM, LARGE). A játékhoz szükséges számításokat a GameLogic osztály végzi. Mindegyik osztályból egyetlen objektum van létrehozva.

Osztály diagram



Részletezett UML diagrammok



+ GameBoard
-infoPanel : GameInfoPanel -logic : GameLogic -fieldButtonListener : fieldButtonListener
<<create>> +GameBoard(size : int, logic : GameLogic, infoPanel : GameInfoPanel) +newGame(size : int) -setUpBoard(size : int) -addButton(i : int, j : int) -refreshButtonTexts() -showGameOverMessage(winner : String)

+ GameMenuBar
-frame : GameFrame -newGame : JMenu -small : JMenuItem -medium : JMenuItem -large : JMenuItem -newGameAction : Action
<<create>> +GameMenuBar(frame : GameFrame)

<<enumeration>> + Player
NOBODY
X
O

+ Size
+SMALL : int
+MEDIUM : int
+LARGE : int

+ GameLogic
-rand : Random -board : Player -size : int -actualPlayer : Player
<<create>> +GameLogic(size : int) +newGame(size : int) +step(row : int, column : int) : String -maxLengthOfAdjacentSigns(row : int, column : int) : int -sameSignInRow(row : int) : int -sameSignInColumn(column : int) : int -sameSignAcrossBackslash(row : int, column : int) : int -firstHalfOfBackslash(row : int, column : int) : int -secondHalfOfBackslash(row : int, column : int) : int -sameSignAcrossSlash(row : int, column : int) : int -firstHalfOfSlash(row : int, column : int) : int -secondHalfOfSlash(row : int, column : int) : int -deleteRandomSign(sign : Player) -nextPlayer() -boardIsFull() : boolean +getActualPlayer() : Player +getBoardSign(i : int, j : int) : Player +endGame()

Megvalósítás

Főprogram

A főprogram (Boot) létrehoz egy új GameFrame-et, és láthatóvá teszi.

```
java.awt.EventQueue.invokeLater(new Runnable() {  
    public void run() {  
        new GameFrame(new GameLogic(Size.SMALL)).setVisible(true);  
    }  
});
```

GameLogic

A játék belső számításait végző osztály a GameLogic.

newGame:

```
public void newGame(int size){  
    actualPlayer = Player.X;  
    this.size = size;  
    board = new Player[size][size];  
    for (int row = 0; row < size; ++row) {  
        for (int column = 0; column < size; ++column) {  
            board[row][column] = Player.NOBODY;  
        }  
    }  
}
```

Step:

Ez a metódus végzi a „lépést”. Először is megvizsgálja, hogy egyáltalán rakhat-e a játékos az adott „mezőre”. Ha igen, rak. Ebben az esetben megnézni, hogy kell-e törölni, illetve ha igen, hányat, és töröl. Ha sikeresen elvégezte a lépést, vált a következő játékosra. Ha kijött az 5 ugyanolyan egymás mellett, az adott játékost visszaadja, és nem lép a következőre. Ha a tábla betelt „döntetlent” ad vissza.

```
public String step(int row, int column) {  
    if (board[row][column] == Player.NOBODY) {  
        board[row][column] = actualPlayer;  
        if (maxLengthOfAdjacentSigns(row, column) >= 5) {  
            return String.valueOf(actualPlayer);  
        } else if (maxLengthOfAdjacentSigns(row, column) == 4) {  
            deleteRandomSign(actualPlayer);  
            deleteRandomSign(actualPlayer);  
            nextPlayer();  
            return "";  
        } else if (maxLengthOfAdjacentSigns(row, column) == 3) {  
            deleteRandomSign(actualPlayer);  
            nextPlayer();  
            return "";  
        } else if (boardIsFull()) {  
            return " - . A meccs döntetlen.";  
        }  
        nextPlayer();  
    }  
    return "";  
}
```

maxLengthOfAdjacentSigns:

Kiválasztja a maximumot a sameSignInColumn, sameSignInRow, sameSignAcrossBackslash, sameSignAcrossSlash metódusok visszatérési értékei közül.

```
private int maxLengthOfAdjacentSigns(int row, int column){
    int max = sameSignInRow(row);
    if (max < sameSignInColumn(column)){
        max = sameSignInColumn(column);
    }
    if (max < sameSignAcrossBackslash(row, column)){
        max = sameSignAcrossBackslash(row, column);
    }
    if (max < sameSignAcrossSlash(row, column)){
        max = sameSignAcrossSlash(row, column);
    }
    return max;
}
```

sameSignInColumn:

Végig iterál azon az oszlopon, mely meg lett adva paraméterként, és visszaadja a leghosszabb olyan összefüggő oszlop hosszát, mely az aktuális játékos jeleit tartalmazza.

```
private int sameSignInColumn(final int column){
    int max = 0;
    for (int i=0; i<size; i++){
        int sum = 0;
        while (i<size && board[i][column]==actualPlayer){
            sum++;
            i++;
        }
        if (max<sum){
            max = sum;
        }
    }
    return max;
}
```

sameSignInRow:

Végig iterál azon a soron, mely meg lett adva paraméterként, és visszaadja a leghosszabb olyan összefüggő sor hosszát, mely az aktuális játékos jeleit tartalmazza.

```
private int sameSignInRow(final int row){
    int max = 0;
    for (int i=0; i<size; i++){
        int sum = 0;
        while (i<size && board[row][i]==actualPlayer){
            sum++;
            i++;
        }
        if (max<sum){
            max = sum;
        }
    }
    return max;
}
```

sameSignAcrossBackslash:

Összeadja a fordított perjel irányban (bal felső -> jobb alsó irány) megszámlolt elemek első és második szakaszát, és hozzáadja a középső elemet. Összegezve tehát a fordított perjel irányú azonos jelek számával tér vissza.

```
private int sameSignAcrossBackslash(int row, int column){
    return (1 + firstHalfOfBackslash(row,column) +
secondHalfOfBackslash(row,column));
}
```

firstHalfOfBackslash:

A megadott elem pozíciójától (paraméterben megadott sor, oszlop) jobbra lefelé számolja meg az azonos jeleket.

```
private int firstHalfOfBackslash(int row, int column){
    int sum = 0;
    int i=row+1;
    int j=column+1;
    while((i < size && j < size) && (i >= 0 && j >= 0) &&
board[i][j]==actualPlayer)
    {
        i++;
        j++;
        sum++;
    }
    return sum;
}
```

secondHalfOfBackslash:

A megadott elem pozíciójától (paraméterben megadott sor, oszlop) balra felfelé számolja meg az azonos jeleket.

```
private int secondHalfOfBackslash(int row, int column){
    int sum = 0;
    int i=row-1;
    int j=column-1;
    while((i >= 0 && j >= 0) && (i < size && j < size) &&
board[i][j]==actualPlayer)
    {
        i--;
        j--;
        sum++;
    }
    return sum;
}
```


sameSignAcrossSlash:

Összeadja a perjel irányban (jobb felső -> bal alsó irány) megszámlolt elemek első és második szakaszát, és hozzáadja a középső elemet. Összegezve tehát a perjel irányú azonos jelek számával tér vissza.

```
private int sameSignAcrossSlash(int row, int column){
    return (1 + firstHalfOfSlash(row,column) +
secondHalfOfSlash(row,column));
}
```

firstHalfOfSlash:

A megadott elem pozíciójától (paraméterben megadott sor, oszlop) balra lefelé számolja meg az azonos jeleket.

```
private int firstHalfOfSlash(int row, int column){
    int sum = 0;
    int i=row+1;
    int j=column-1;
    while((i >= 0 && j < size) && (i < size && j >= 0) &&
board[i][j]==actualPlayer)
    {
        i++;
        j--;
        sum++;
    }
    return sum;
}
```

secondHalfOfSlash:

A megadott elem pozíciójától (paraméterben megadott sor, oszlop) jobbra felfelé számolja meg az azonos jeleket.

```
private int secondHalfOfSlash(int row, int column){
    int sum = 0;
    int i=row-1;
    int j=column+1;
    while((i >= 0 && j < size) && (i < size && j >= 0) &&
board[i][j]==actualPlayer)
    {
        i--;
        j++;
        sum++;
    }
    return sum;
}
```

deleteRandomSign:

```
private void deleteRandomSign(Player sign){
    int randomRow = rand.nextInt(size);
    int randomColumn = rand.nextInt(size);
    while(board[randomRow][randomColumn] != sign){
        randomRow = rand.nextInt(size);
        randomColumn = rand.nextInt(size);
    }
    board[randomRow][randomColumn] = Player.NOBODY;
}
```

boardIsFull:

Végignézi a tábla elemeit, és ha nem talál NOBODY játékost (tehát olyan helyet, ahova lehetne rakni), igaz értékkel tér vissza (ami azt jelenti, hogy a tábla megtelt).

```
private boolean boardIsFull() {  
    for(int row=0; row<size; row++){  
        for(int column=0; column<size; column++){  
            if(board[row][column]==Player.NOBODY) {  
                return false;  
            }  
        }  
    }  
    return true;  
}
```

GameBoard

A GameBoard végzi a játéktábla megjelenítését. JPanel-ből lett származtatva.

newGame:

```
public void newGame(int size){
    removeAll();
    setUpBoard(size);
    infoPanel.updateLabelText();
}
```

setUpBoard:

Elkészíti a táblát.

```
private void setUpBoard(int size) {
    setLayout(new GridLayout(size, size));
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            addButton(i, j);
        }
    }
}
```

addButton:

Létrehozza a gombokat, pozícióval, mérettel, és hozzáadja a táblához.

```
private void addButton(int i, int j){
    final GameFieldButton button = new GameFieldButton(i, j);
    button.addActionListener(fieldButtonActionListener);
    button.setPreferredSize(new Dimension(45, 45));
    add(button);
}
```

refreshButtonTexts:

A táblán elhelyezett gombok feliratát változtatja meg a GameLogic által kezelt board nevű tömb alapján.

```
private void refreshButtonTexts(){
    for(Component component : getComponents()){
        GameFieldButton button = (GameFieldButton) component;
        int row = button.getRow();
        int column = button.getColumn();
        if (logic.getBoardSign(row, column) != Player.NOBODY){
            button.setText(String.valueOf(logic.getBoardSign(row,
column)));
        } else {
            button.setText("");
        }
    }
}
```

showGameOverMessage:

```
private void showGameOverMessage(String winner) {
    JOptionPane.showMessageDialog(this, "A nyertes: " + winner);
    logic.endGame();
}
```

```
refreshButtonTexts();  
}
```

FieldButtonActionListener

actionPerformed:

Elvégzi a lépést az adott gombon („mezőn”), frissíti az információ panelt, és ha a visszakapott érték egy játékos vagy döntetlen, megjeleníti a játék vége üzenetet.

```
@Override  
public void actionPerformed(ActionEvent e) {  
    GameFieldButton button = (GameFieldButton) e.getSource();  
    String winner = logic.step(button.getRow(), button.getColumn());  
    refreshButtonTexts();  
    infoPanel.updateLabelText();  
    if (winner != ""){  
        showGameOverMessage(winner);  
    }  
}
```

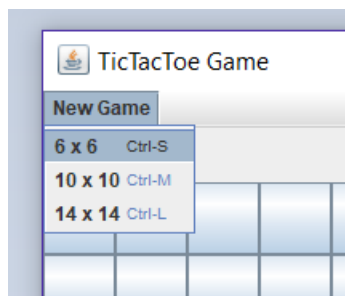
Tesztelés

Elvégzett tesztesetek


- Menügombok működése
 - új játék indításánál jó méret töltődik be, lenullázódik a tábla (`logic.newGame()`, `gameBoard.newGame()`) (1.-5.)
- `step()` metódus tesztelése
 - akkor változik meg a pálya, ha a játékos nem lefoglalt mezőre rak, különben nem (6.)
 - ha 5 ugyanolyan keletkezik egy sorban, oszlopban, átlósan, akkor a jelenlegi játékos nyer (9.-10.)
 - ha 4 ugyanolyan keletkezik egy sorban, oszlopban, átlósan, akkor 2 jele a pályáról törlődik (`deleteRandomSign()`) (8.)
 - ha 3 ugyanolyan keletkezik egy sorban, oszlopban, átlósan, akkor 1 jele a pályáról törlődik (`deleteRandomSign()`) (7.)
 - ha betelt a pálya (nincs lehetőség sehova se rakni, `boardIsFull()`), befejeződik a játék
 - akkor változik az aktuális játékos, ha az előző tudott rakni (`nextPlayer()`) (6.-8.)
 - jól határozza meg az oszlopokban, sorokban, és átlósan az egymás melletti ugyanolyan jelek számát - `maxLengthOfAdjacentSigns()` rekurzívan (`sameSignInRow()` 9., `sameSignInColumn()` 8., `sameSignAcrossSlash()` 10., `sameSignAcrossBackslash()` 7.)
- `FieldButtonActionListener :: actionPerformed()` tesztelése
 - helyesen változnak meg a gombok feliratai, ha törlés volt eltűnik, ha lépés volt, „lerak” (`refreshButtonTexts()`) (6.-8.)
 - megváltozik az aktuálisan soron következő szerint az infoPanel (`updateLabel()`) (6.-10.)
 - `showGameOverMessage()` (9.-10.)
- kilépés megerősítés ablak működése (11.)

(Megjegyzés: a kék számok a teszt sorszámaára utalnak.)


A menü:



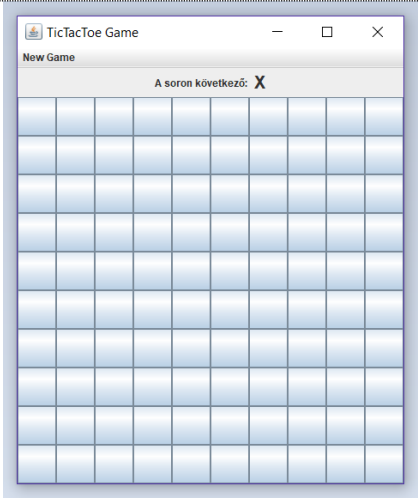
1. Játék megnyitása

Logic	GUI
<pre> GameLogic{ size=6 actualPlayer=X board= - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - } </pre>	

2. Új játék kezdése (6*6)

Logic	GUI
<pre> GameLogic{ size=6 actualPlayer=X board= - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - } </pre>	

3. Új játék indítása (10*10)

Logic	GUI
<pre> GameLogic{ size=10 actualPlayer=X board= - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - } </pre>	


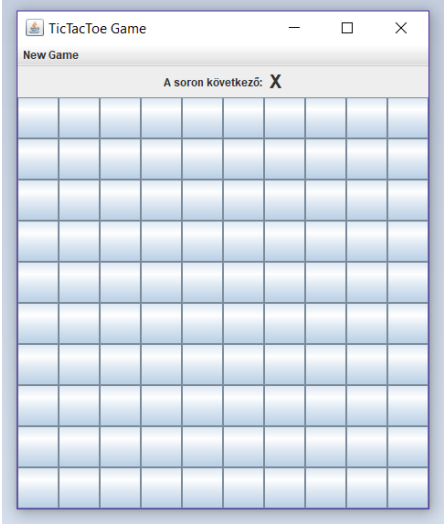
4. Új játék kezdése (14*14)

[illegible]



5. Új játék kezdése nem üres tábla esetén (10*10)

	Logic	GUI
E l ő tt e	<pre>GameLogic{ size=6 actualPlayer=0 board= - - - - X - X 0 - - - - - X - - - - - X - - 0 - 0 - - - - - - - }</pre>	
U t á n a	<pre>GameLogic{ size=6 actualPlayer=X board= - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }</pre>	

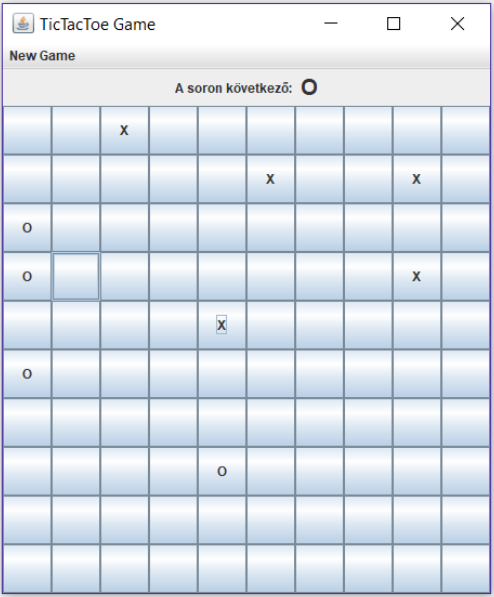
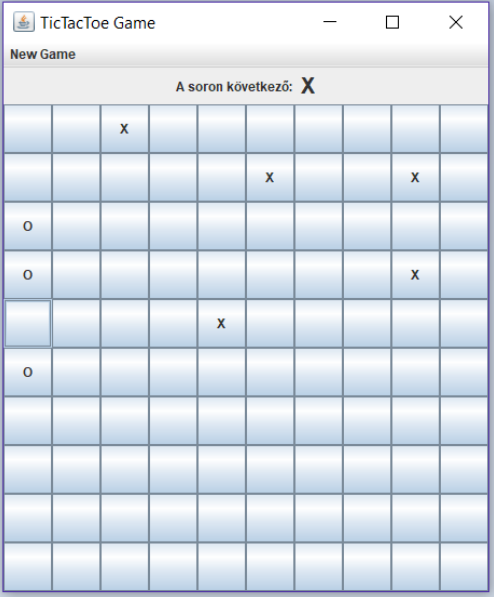
6. Ha nem „rossz” helyre rak az aktuális játékos, megint ő következik

	Logic	GUI
E l ő tt e	<pre> GameLogic{ size=6 actualPlayer=0 board= - - - - - X - X 0 - - - - - - X - - - - - - X - - 0 - 0 - - - - - - - - } </pre>	
U t á n a	<pre> GameLogic{ size=10 actualPlayer=X board= - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - } </pre>	

7. 3 ugyanolyan jel egymás mellett

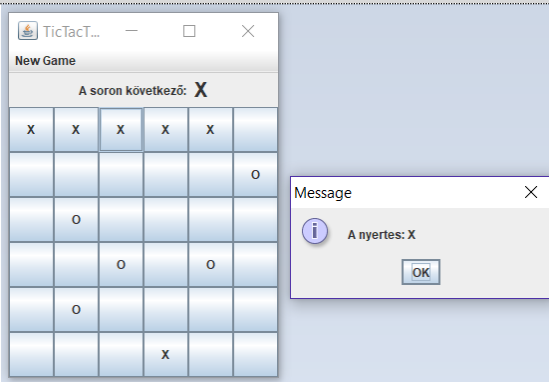
	Logic	GUI
E l ő tt e	<pre> GameLogic{ size=6 actualPlayer=X board= - - - 0 - - - 0 - X - X - - - - - - X - - 0 - - - - - - - - X - 0 - } </pre>	
U t á n a	<pre> GameLogic{ size=6 actualPlayer=0 board= - - - 0 - - - 0 - X - - - - X - - - - X - - 0 - - - - - - - - X - 0 - } </pre>	

8. 4 ugyanolyan jel egymás mellett

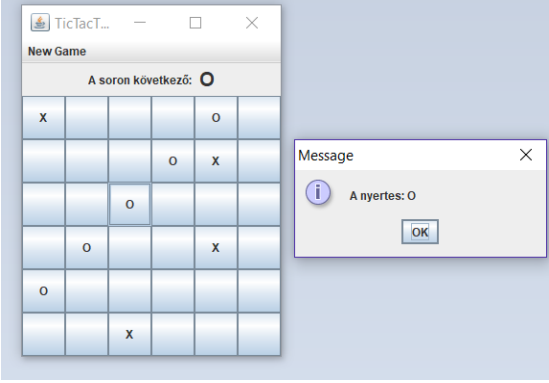
	Logic	GUI
E l ő tt e	<pre> GameLogic{ size=10 actualPlayer=0 board= - - X - - - - - - - - - - X - - X - 0 - - - - - - - - 0 - - - - - - - X - - - - - X - - - - - 0 - - - - - - - - - - - - - - - - - - - - 0 - - - - - - - - - - - - - - - - - - - - } </pre>	
U t á n a	<pre> GameLogic{ size=10 actualPlayer=X board= - - X - - - - - - - - - - X - - X - 0 - - - - - - - - 0 - - - - - - - X - - - - - X - - - - - 0 - - - - - - - - - - - - - - - - - - - - 0 - - - - - - - - - - - - - - - - - - - - } </pre>	

(Megjegyzés: 2-t törölt, ez esetben az egyik ilyen elem, az lett, ami most lett lerakva, szóval a véletlenszerűen kiválasztott elem, lehet az imént lerakott is.)

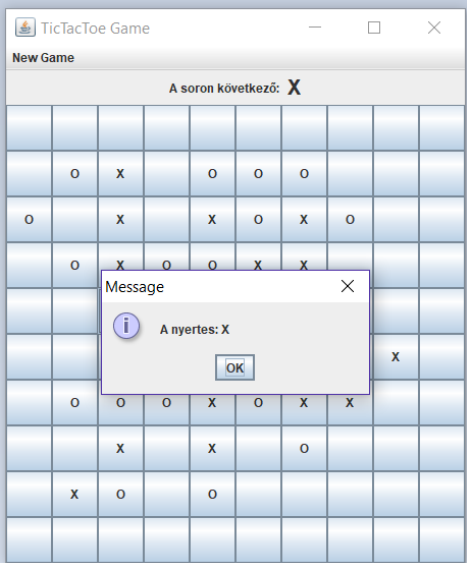
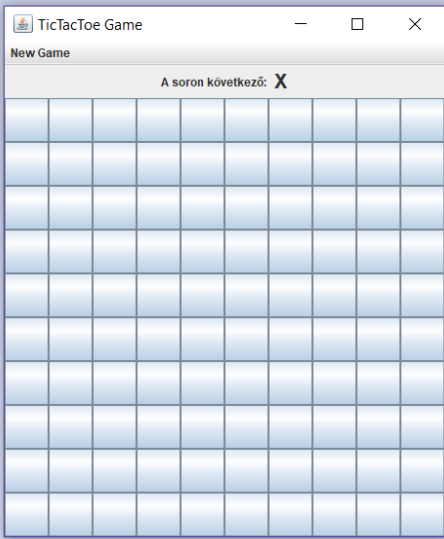
9. X nyert, üzenet

Logic	GUI
<pre> GameLogic{ size=6 actualPlayer=X board= X X X X X - - - - - - 0 - 0 - - - - - - 0 - 0 - - 0 - - - - - - - X - - } </pre>	

10. O nyert, üzenet

Logic	GUI
<pre> GameLogic{ size=6 actualPlayer=O board= - X - - 0 - - - - 0 X - - - 0 - - - - 0 - - X - 0 - - - - - - - X - - - } </pre>	

11. Játék vége után (OK gomb lenyomása után) új játék kezdése

	Logic	GUI
E l ő t t e	<pre> GameLogic{ size=10 actualPlayer=X board= - - - - - - - - - - 0 X - 0 0 0 - - - 0 - X - X 0 X 0 - - - 0 X 0 0 X X - - - - - - X 0 X 0 X - - - - X 0 X - 0 0 X - - 0 0 0 X 0 X X - - - - X - X - 0 - - - - X 0 - 0 - - - - - - - - - - - - - - } </pre>	
U t á n a	<pre> GameLogic{ size=10 actualPlayer=X board= - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - } </pre>	

(Megjegyzés: Az automatikus újratekés mérettartó.)

12. kilépés

	Logic	GUI
	<pre> GameLogic{ size=6 actualPlayer=0 board= 0 0 X X - 0 - - - 0 X - - X - - - - 0 0 - - X X 0 - 0 X - X X - - 0 - 0 } </pre>	