

1.	Stellung im Unterricht, das Spiel .....	1
2.	Programmeigenschaften .....	3
2.1	Ein interaktives Spielfeld .....	3
2.2	Programmfragment .....	4
3.	Arbeit mit Arrays .....	6
4.	Spielablauf - grobe Planung .....	8
4.1	Verfeinerung (1) .....	9
4.2	Verfeinerung (2) .....	10
4.3	Verfeinerung (3) .....	11
5.	Programmlisting .....	13
6.	Verbesserungen und Aufgaben .....	17

## Stellung im Unterricht

Das Spiel TIC TAC TOE ist eine gute Möglichkeit z.B. im 11. Jahrgang in Form einer Projektarbeit Schülerinnen und Schüler an einem größeren Problem arbeiten zu lassen. Dazu brauchen sie keine zu umfangreichen Vorkenntnisse. Der einzige komplexe Datentyp, der benötigt wird, sind Arrays. Das Spiel bietet verschiedenste Lösungsansätze und ermöglicht es damit Schülerinnen und Schülern, gemeinsam mit Anderen ihren Weg zu einer für sie annehmbaren Lösung zu finden. Nicht alle Ergebnisse werden vollständig oder elegant sein. Jede Lösung läßt auch immer noch eine Verbesserung oder Verschönerung zu.

Die nachfolgenden Betrachtungen sollen zeigen, dass es nicht nur einen Weg zu einer richtigen Lösung gibt, sondern viele Ansatzmöglichkeiten existieren. Auch langsamere Schülerinnen und Schüler werden ihren Lösungsweg mit ein wenig Hilfe finden können. Man muss nur genügend Zeit geben und nicht zu sehr steuern wollen. Für sehr schnelle und programmiererfahrenen Schülerinnen und Schüler wird es bei erfolgreicher Lösung am Ende eher um Verschönerungen, ein nettes Erscheinungsbild oder die Erweiterung auf ein 4 x 4 Spielfeld gehen. Alles das ist in wenigen Unterrichtswochen zu erreichen.

### Das Spiel

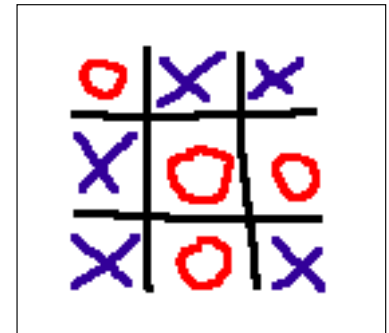
Für das Spiel TIC-TAC-TOE gibt es in fast jeder Programmiersprache Lösungsbeispiele. Auch in JAVA kann man z.B. eine Lösung der Firma SUN selbst bewundern. Klein, elegant und ohne jeden Kommentar. Für den Unterricht nützt sie uns so nicht viel.

Für alle aber, die das Spiel nicht kennen, hier ganz kurz die Regeln:

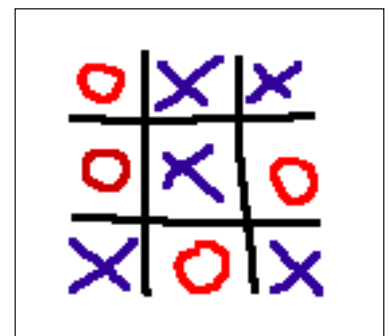
Es wird auf einem 3 x 3 Felder großen Bereich gespielt. Abwechselnd zeichnen zwei Spielerinnen / zwei Spieler entweder ein Kreuz oder einen Kringel in ein leeres Feld. Wem es gelingt, zuerst eine Zeile oder eine Spalte oder eine Diagonale mit dem eigenen Symbol zu füllen, der hat gewonnen.

Wir wollen nun ein Programm entwickeln, das mit einem Spieler / mit einer Spielerin spielt. Dabei soll es folgenden Anforderungen genügen:

- 1) Es soll interaktiv sein. Die Spielerin/ der Spieler soll mit der Maus sein Symbol auf ein freies Feld "klicken" können.
- 2) Dabei sollen unerlaubte Züge verhindert werden ( wenn ein Feld schon besetzt ist, darf man dort nicht mehr hinsetzen).
- 3) Das Programm soll , wenn es dran ist, auf ein freies Feld setzen.
- 4) Das Programm soll feststellen, wenn ein Sieg ( für wen auch immer ) vorliegt.
- 5) Das Programm soll stoppen, wenn es einen Sieg gibt oder wenn nach 9 Zügen unentschieden gespielt wurde.



unentschieden



da hat aber Kringel nicht aufgepasst !

### Zusatzanforderungen:

- 6) Das Programm soll nicht mehr gegen eine Spielerin/ einen Spieler verlieren können ( also mindestens unentschieden spielen).
- 7) Das Programm soll mit einer passendes Meldung beenden, wenn nach einer gewissen Anzahl von Zügen nur noch ein Unentschieden möglich ist.

Die Zusatzanforderungen machen besonders deutlich, dass es für die Lösung sehr unterschiedliche Anspruchsebenen gibt.

Wir werden und zuerst mit der Darstellung des Spielfeldes am Bildschirm befassen und erst danach die eigentliche Problemlösung - nämlich das Spiel - angehen.

### Das Spielfeld - interaktiv

Nach den Vorüberlegungen in vorangegangenen Abschnitten ist eine Lösung die Darstellung des Spielfeldes durch Buttons, die anfangs leer ( unbeschriftet ) sind und dann - je nach dem wer dran ist - mit einem Kreuz oder Kringel besetzt werden.

Dabei wird es für die Knöpfe eine ActionListener geben, der dann aktiv ist, wenn die Spielerin/ der Spieler an der Reihe ist, denn der Rechner drückt die Maustaste nur virtuell.

### Aufgabe:

Erzeuge ein Appletwindow mit einem passenden Layout, das 9 Spielbuttons, ein Start/Löschbutton und zwei Label enthält. Eines für eine Überschrift und eines für Mitteilungen freundlicher Natur an die Spielerin / den Spieler.

Eine mögliche Lösung ( als Kontrolle und Hilfe ):

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class TICTAC1 extends Applet implements ActionListener
{
    //*****
    // Zuerst die Objekte und Variablen
    //*****
    Button b1,b2,b3,b4,b5,b6,b7,b8,b9;           // können wir das wollen , 9 Variable ?
    Label ueberschrift,nachricht;              // zwei Labels
    Button start;                               // Startknopf
    Panel nord,mitte,sued;                     // Panels
    int weristdran,zugzahl;                    // Spieler=1 und Computer =2, Zugzahl

    public void init()
    {
        ueberschrift= new Label(" TIC- TAC - TOE");
    }
}
```

```

ueberschrift.setFont(new Font("SanSerif", 1, 20));
ueberschrift.setAlignment(Label.CENTER);
nord=new Panel();
nord.add(ueberschrift);

b1= new Button(" "); //ohne Beschriftung also mit Leerzeichen
b1.setFont(new Font("SanSerif", 1, 28));
b1.addActionListener(this); // Zuweisung des actionListeners, der dem ganzen Applet gehört

```

Das muss nun für alle 9 Knöpfe so gemacht werden. Mit *copy* und *paste* sicher nicht so sehr schwer. Auch wenn der Programmtext sehr lang wird, ist das Ganze recht gut lesbar. Dann muss nun noch für den ActionListener die Methode *actionPerformed* her. Darin könnte man abfragen, welcher Button geklickt wurde, ob der noch leer ist und wer dran ist (Spieler oder Rechner)? Ob das am Ende wirklich alles bedacht werden muss, ist jetzt noch unklar.

```

public void actionPerformed(ActionEvent meinEvent)
{
    Object welcherButton; //welcher Knopf war gedrückt
    String wasstehtdrauf = meinEvent.getActionCommand(); //Beschriftung holen
    welcherButton=meinEvent.getSource(); // welcher Button war gedrückt ?
    nachricht.setText(" Mitteilungen : ");
    if (welcherButton==b1) // Wenn Button b1 geklickt war...
    {
        if (wasstehtdrauf==" ") // Feld( oder Button ) noch frei ?
            {if (weristdran==1) // wenn wir das brauchen ?
                {
                    b1.setLabel("X"); // weristdran = 1 = Spieler = X und
                    weristdran=0; // Rechner ist dran
                }
                else
                {
                    b1.setLabel("O"); // weristdran =0 : Rechner setzt "O" und
                    weristdran=1; // nun kommt der Spieler dran
                }
                // wenn wir das brauchen ?
            }
        else // Feld schon belegt .....
            nachricht.setText("Feld ist schon belegt");
    } // Ende von welcherButton = b1
}

```

Und das nun für alle 9 Buttons.

Auch hier wird der Text lang aber nicht unübersichtlich. Bleibt am Ende noch der Startknopf. Er soll alle Buttons wieder leer machen, die Zugzahl auf Null zurücksetzen und eventuell noch eine Nachricht ins Fenster schreiben.

```

if (welcherButton==start)
{
    b1.setLabel(" ");
    b2.setLabel(" ");
    b3.setLabel(" ");
    b4.setLabel(" ");
    b5.setLabel(" ");
    b6.setLabel(" ");
    b7.setLabel(" ");
    b8.setLabel(" ");
    b9.setLabel(" ");
    weristdran=1;
    zugzahl=0;
    nachricht.setText("Neues Spiel kann beginnen.....");
}

```

Dieser Teil könnte aus *actionPerformed* herausgenommen werden und als eigene Methode in der Klasse TICTACTOE1 erscheinen:

```

public void myReset()
{
    b1.setLabel(" ");

    .... u.s.w.

    b9.setLabel(" ");
    weristdran=1;
    zugzahl=0;
    nachricht.setText("Neues Spiel kann
                      beginnen.....");
}

```

Dann wäre der Code in *actionPerformed* sehr viel kleiner:

```

if (welcherButton==start)
{
    myReset();
}

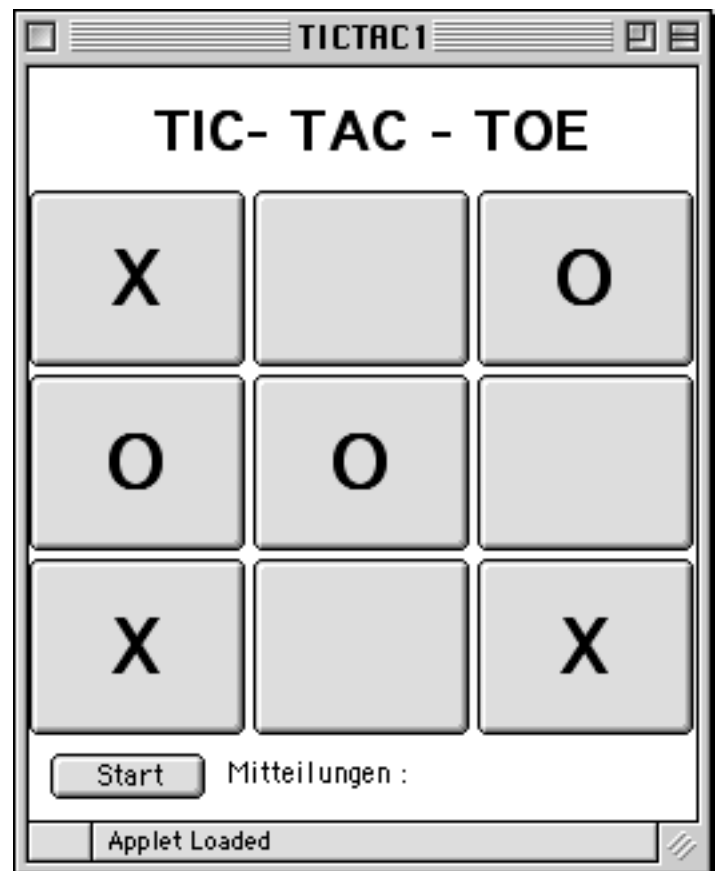
```

Nicht vergessen, auch dem Button Start muss in der Methode *init()* der ActionListener zugewiesen worden sein mit

```

start=new Button(" Start");
start.addActionListener(this);

```



Wenn alles geklappt hat, sollte das Appletwindow etwa so aussehen.

Wie man sieht haben wir bis jetzt nichts zum eigentlichen Spiel beigetragen. Es sind bis hierher nur Klassen und Methoden erarbeitet worden, die sich auf die Bildschirmgestaltung beziehen.

Diese Trennung von "Benutzerschnittstelle" und "Programm" ist auch für den Unterricht eine sinnvolle Sache, weil vermieden wird, dass sich Probleme in den Vordergrund schieben, die an der eigentliche Aufgabe ( und damit auch an dem eigentlichen inhaltlichen Anspruch ) vorbeigehen.

Bevor wir uns mit Lösungsansätzen für das eigentliche Spiel befassen, solle es nun aber noch um eine Verbesserung des bisher erarbeiteten Programmcodes gehen.

## Arrays

Bei 9 Variablen ist es noch leicht möglich, den Überblick zu behalten. Was aber, wenn es noch mehr von diesen Variablen werden ?

Dann muss eine numerierte Liste der Variablen her, die leichter zu handhaben ist. Solch eine Liste gleicher Variablen, Objekte oder Klasseninstanzen heisst **ARRAY**.

In unserm Fall beginnt das mit der Definition der Buttons.  
Die soll nun wie folgt aussehen:

```
Button[] b = new Button[9];           // eine Liste mit 9 Buttons
```

Die eckigen Klammern sagen dem Javacompiler, dass nun viele Buttons kommen, die alle den Namen **b** tragen, aber durchnummeriert werden beginnend mit der Nummer 0 ( sowas doofes ! Wer käme schon auf die Idee, dem ersten Getränk am Abend die Nummer 0 zu geben ?- Niemand ! Aber das erste Element eines Arrays, das hat die Nummer 0.)

Die neun Knöpfe werden also die Namen **b[0]**, **b[1]**, ....., **b[8]** tragen. Das muss man aber nicht vorher festlegen, das passiert beim Erzeugen der Buttons durch eine Wiederholschleife, in der eine Zählvariable **i** hochgezählt wird:

```
for (i=1;i<9;i++)
{
    b[i]= new Button(" ");
    b[i].setFont(new Font("SanSerif",1,28));
    b[i].addActionListener(meinLauscher);
}
```

Damit sind alle Buttons erzeugt.

Nun kommen die auch so in das Layout:

In der Mitte des Layouts existiert ein Panel namens *Mitte*. Dort wird das *GridLayout* eingebaut und in dieses werden die Buttons nun plaziert:

```
mitte.setLayout(new GridLayout(3,3,3,3));
for (i=1;i<9;i++)
{
    mitte.add(b[i]);
}
```

Auch in der Methode *actionPerformed* kann man auf die Elemente des Arrays zugreifen:

```
public void actionPerformed(ActionEvent meinEvent)
{
    Object welcherButton;                //welcher Knopf war gedrückt
    String wasstehtdrauf = meinEvent.getActionCommand();//Beschriftung holen
    welcherButton=meinEvent.getSource(); // welcher Button war gedrückt ?
    nachricht.setText(" Mitteilungen :      ");
    for (i=0;i<9;i++)                    // Schleifenbeginn .. gehe alle Buttons durch
    {
        if (welcherButton==b[i])        // für Button b[i]
        {
            if (wasstehtdrauf==" ")     // Feld( oder Button ) noch frei ?
                {if (weristdran==1)
                    {
                        b[i].setLabel("X"); // weristdran = 1 : Spieler setzt "X" und
                        weristdran=0;      // Rechner ist dran
                    }
                    else
                    {
                        b[i].setLabel("O"); // weristdran =0 = Rechner = O und
                        weristdran=1;      // nun kommt der Spieler wieder dran
                    }
                }
            else                          // Feld schon belegt .....
                nachricht.setText("Feld ist schon belegt");
        }
    }
} // Ende von for....
```

..... U.S.W .....

Das ist natürlich ein wenig dumm, denn der Rechner drückt ja keine Maustaste, die dann hier im ActionListener abgefragt werden müsste. Der Event wird allein vom Spieler ausgeführt. Der obige Programmtext wäre also z.B. für zwei Spieler geeignet. Denen allerdings kann man nur empfehlen sich ein Blatt Papier zu nehmen und dann zu spielen, das macht mehr Spaß.

Wir ändern also noch einmal den Programmtext ab:

```
if (welcherButton==b[i])                // für Button b[i]
{
    if (wasstehtdrauf==" ")             // Feld( oder Button ) noch frei ?
        {
            b[i].setLabel("X");         // weristdran = 1: Spieler = "X " und
            weristdran=0;               // Rechner ist dran
        }
    else
```

```
nachricht.setText("Feld ist schon belegt");
}
```

// Ende von welcherButton = b1

Wenn man die Programmplanung soweit gebracht hat, sehen Schülerinnen und Schüler notwendige Änderungen der Programmstruktur. So ist es nach jedem Spielerzug sicher notwendig, zu prüfen, ob der Spieler / die Spielerin schon gewonnen hat, denn dann muss kein Computerzug mehr erfolgen. Ein *kompletter* Spielzug muss dann möglicherweise wieder etwas anders aussehen: *Spielerzug -- auf Sieg prüfen -- ggf Computerzug -- auf Sieg prüfen.*

### Aufgabe:

Bringe das Programm so mit Arrays zum Laufen, dass die Buttons bedienbar sind und nur auf freie Felder gesetzt werden kann.

### Spielen und spielen lassen...

Nun geht es an das *eigentliche* Spielen, dass man nun auf sehr unterschiedliche Art realisieren kann. Es gibt für TIC-TAC-TOE eine ganze Reihe von Lösungen, die in Büchern zu finden sind oder auch im WWW.

Alle haben eines gemeinsam: Sie sind so schlecht erklärt, dass sie wohl nur der Autor versteht. Das soll uns hoffentlich besser gelingen. Wir werden *einen* möglichen Lösungsweg beschreiten und dabei hoffentlich unsere Lösungsstrategie immer weiter verbessern.

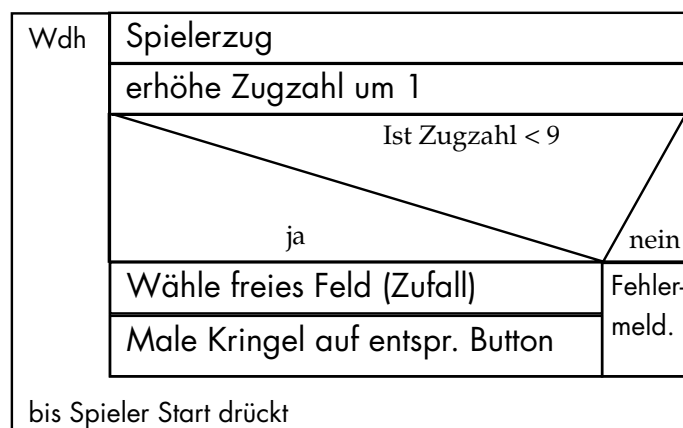
Es kann auch sein, dass wir einige Details unserer "Benutzerschnittstelle" ( was für ein respekt-einflößendes Wort) noch verändern müssen.

*Eine sehr einfache Lösung:*

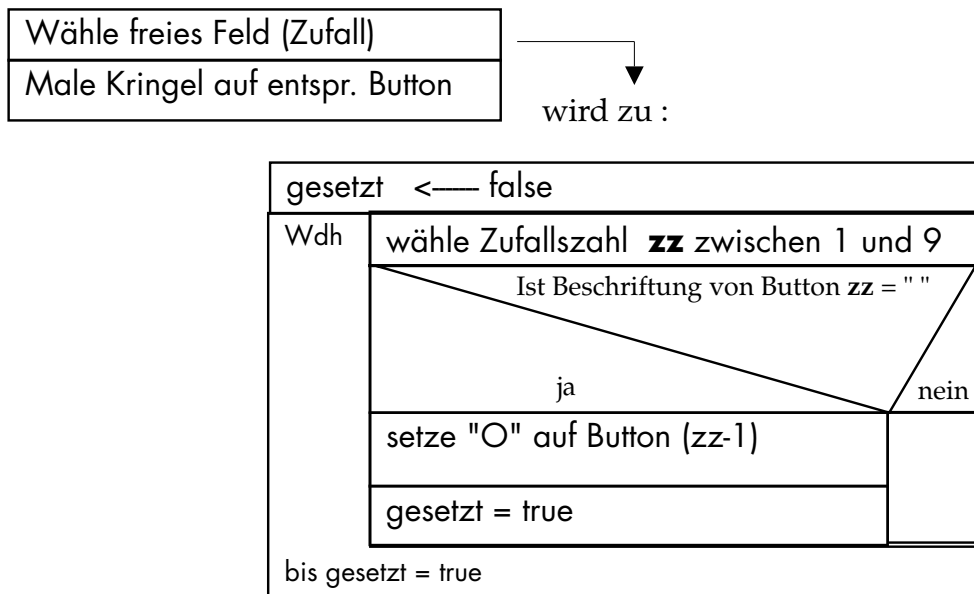
Der Spieler/ die Spielerin beginnt. Dann sucht der Rechner per Zufall ein freies Feld und setzt dort hin.

Das geht solange, bis alle Felder belegt sind oder ein Sieg zu verzeichnen ist. Diese Entscheidung bleibt dem Spieler überlassen, der dann den Startknopf drückt.

Dieser sehr simple Ansatz als Struktogramm.



Das muss man noch etwas genauer machen:



Dabei taucht eine neue Variable " *gesetzt* " auf, die wir noch verwenden wollen. In ihr wird festgehalten, ob das Programm ( der Rechner ) sein Symbol plaziert hat oder nicht. Wenn der Computerzug beginnt, steht diese Variable auf *false*.

Wie der Zufallsgenerator zur Erzeugung der Zahl *zz* funktioniert ,kann man in unseren bekannten Ballprogrammen nachlesen und daher nun die

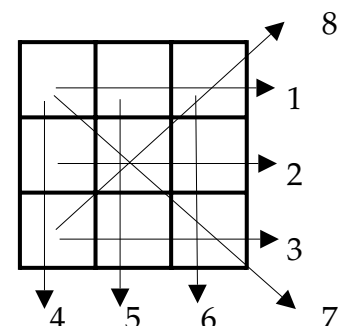
### Aufgabe:

Schreibe das Programm so um, dass Spieler und Rechner miteinander spielen. Dabei soll der Rechner noch nicht prüfen, ob ein Sieg vorliegt, wohl aber meckern, wenn es neun Spielzüge waren. Es soll weder für den Rechner noch für die Spielerin / den Spieler möglich sein, auf ein schon besetztes Feld zu setzen.

Nun geht es um einen Schritt weiter. Nach jedem Spielerzug und nach jedem Computerzug soll der Rechner prüfen, ob ein Sieg vorliegt. Diese Prüfung sollte in einer gesonderten Methode ablaufen, da sonst die Gefahr besteht, dass unser Programm unübersichtlich wird.

Um dieses Problem zu lösen, müssen wir uns einige Tricks mit den Feldern einfallen lassen.

Es sind acht Reihen, Spalten, Diagonalen zu prüfen: Das könnte man nun auf sehr unterschiedliche Art prüfen. z.B. wäre es möglich, eine Variable mit der Bezeichnung *Spieler-sieg* zu vereinbaren, die den Wert true oder false annehmen kann und dann etwa folgendermaßen vorzugehen:



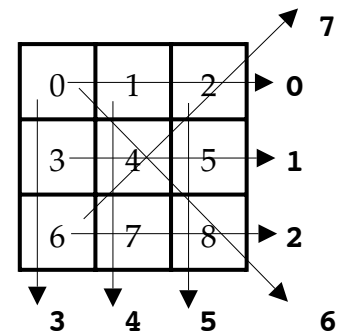


dann mit irgendwelchen Wiederholschleifen ein kurzes Programm erzeugen ?

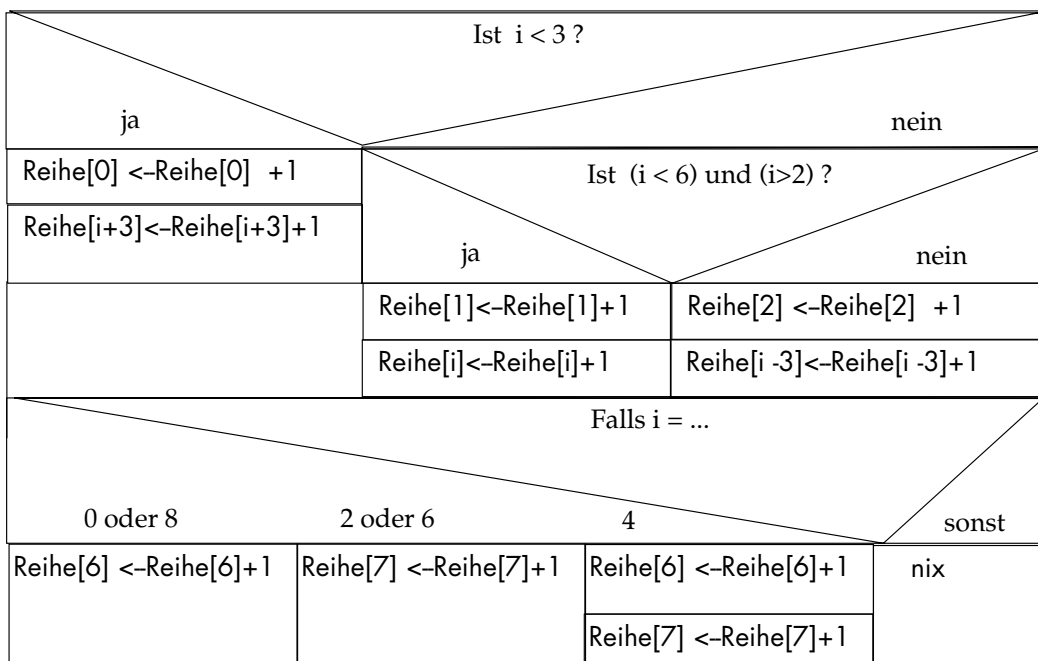
*Schaun' wir mal:*

Feld - und Reihennummern , wie man sie in Arrays verwalten kann .

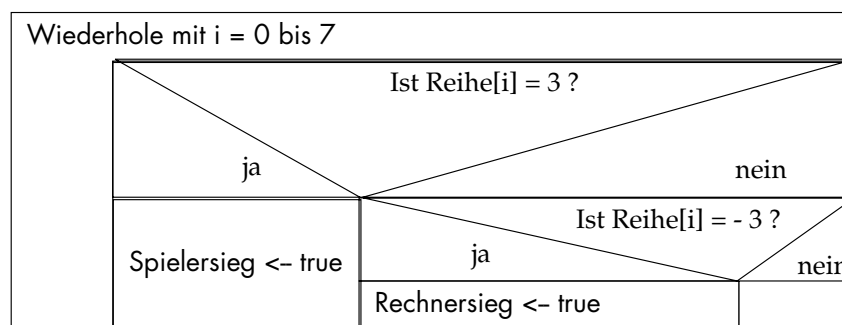
Innehalb der Setzroutine, die in *actionPerformed* untergebracht ist, kann man dann Summen setzen, nämlich wenn



```
b[i].setLabel("X");
```



für den Spielerzug gilt, dann kommt nun die Abfrage:



Man prüfe selbst nach, dass diese Festlegung stimmt.  
Bleibt nun noch die Prüfung, ob *Sieg oder nicht*, was in einer weiteren Schleife gemacht wird.  
Liest sich alles recht gut.

Der zugehörige Programmtext muss jetzt zeigen, ob ein funktionstüchtiges übersichtliches Programm dabei herauskommt.

Es sind hier die veränderten Teile des schon bekannten Programms kommentiert:

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class TICTACTOE3 extends Applet implements ActionListener
{
  /***/
  // Zuerst die Objekte und Variablen
  //
  /***/
  Button[]   b = new Button[9];
  Label      ueberschrift,nachricht;
  Button     start;
  Panel      nord,mitte,sued;
  int        weristdran,zugzahl;           // weristdran= 1 für Spieler und 2 für Computer
  int        i;                           // Zählvariable
  boolean    spielerSieg,computerSieg,unentschieden; // wer siegt ?
  int[]      reihen = new int[8];         // 3 Zeilen, 3 Spalten, 2 Diagonalen

  public void init()
  {
    ueberschrift= new Label(" TIC- TAC - TOE");
    ueberschrift.setFont(new Font("SanSerif",1,20));
    ueberschrift.setAlignment(Label.CENTER);
    nord=new Panel();
    nord.add(ueberschrift);
    for (i=0;i<9;i++)
      {
        b[i]=new Button(" ");
        b[i].setFont(new Font("SanSerif",1,28));
        b[i].addActionListener(this);
      }
    mitte=new Panel();
    mitte.setLayout(new GridLayout(3,3,3,3));
    for(i=0;i<9;i++)
```

```

    {
        mitte.add(b[i]);
    }
start=new Button(" Start");
start.addActionListener(this);
nachricht= new Label(" Neues Spiel kann beginnen ");
sued= new Panel();
sued.add(start);
sued.add(nachricht);
setLayout(new BorderLayout());
add("North",nord);
add("Center",mitte);
add("South",sued);
myReset(); // das Spiel beginnt
}

//*****
// Hier wird nun die Methode actionPerformed
//*****
public void actionPerformed(ActionEvent meinEvent)
{
    int i;
    Object welcherButton;
    String wasstehtdrauf = meinEvent.getActionCommand();
    welcherButton=meinEvent.getSource();
    nachricht.setText(" Mitteilungen : ");
    if((zugzahl<9)&&!spielerSieg)&&!computerSieg) // es geht bis zum Zug 9 oder Sieg
    {
        for(i=0;i<9;i++)
        {
            if (welcherButton==b[i])
            {
                if (wasstehtdrauf==" ") // wenn Button frei
                {
                    b[i].setLabel("X");
                    zugzahl++; // zugzahl um 1 erhöhen
                    verwalteReihen(i); // Summen aktualisieren
                    gibtesSieg(); // hat Spieler gewonnen ?
                    if(!spielerSieg) // wenn Spieler nicht gewonnen hat...
                    {
                        Computerzug(); // Computer ist dran und setzt
                        gibtesSieg(); // hat Computer gewonnen ?
                        if(computerSieg)
                        {nachricht.setText("Computer hat gewonnen");}
                    } // Ende von nicht Spielersieg
                }
            }
        }
    }
}

```

```

        else
            Nachricht.setText("Spieler hat gewonnen !"); // wenn doch Spielersieg
        } // Ende von if (! spielerSieg )
    else
        Nachricht.setText("Feld ist schon belegt"); // wenn Feld nicht frei
    } // Ende von if (wasstehtdrauf == " ")
} // Ende von if( welcherButton==b[i])
} // Ende von for i= ....

// Nun noch der Startbutton

if (welcherButton==start)
{
    myReset();
} // Ende von actionPerformed

/*****/
// Setzt am Anfang alles zurück
//
/*****/
public void myReset()
{
    int j;
    for(j=0;j<9;j++)
    {
        b[j].setLabel(" ");
    }
    for(j=0;j<8;j++)
    {
        reihen[j]=0;
    }
    zugzahl=0;
    spielerSieg=false;
    computerSieg=false;
    unentschieden=false;
    weristdran=1; // Spieler beginnt
    Nachricht.setText("Neues Spiel kann beginnen.....");
}

```

```

/*****/
// Nun eine Methode zur Wahl einer Zufallszahl zwischen 0 und 8
// bzw. zwischen 1 und 9; gibt ein int zurück
/*****/
public int zufall()
{
return (int)Math.round(Math.random()*8); // erzeugt Zufallszahl zwischen 0 und 8
}

/*****/
// Computerzug
// sucht solange nach einem Feld, bis er ein freies gefunden hat
/*****/
public void Computerzug()
{
boolean gesetzt=false; // zu Beginn kein Erfolg beim Setzen
int i; // wohin soll er setzen ?
String was_steht_drauf; // Buttonbeschriftung
while ((!gesetzt)&&(zugzahl<9)) // solange noch nicht erfolgreich gesetzt tue....&& ist UND
{
i=zufall(); // eine Buttonnummer per Zufall wählen
was_steht_drauf=b[i].getLabel();
if(was_steht_drauf==" ")
{
b[i].setLabel("O"); // setze dahin
zugzahl++; // Zugzahl um 1 erhöht
weristdran=0; // Rechner ist momentan dran
verwalteReihen(i); // Summen aktualisieren
gesetzt=true;
weristdran=1; // Spieler ist dran
}
} // Ende von while
} // Ende von Computerzug

/*****/
// Nach jedem Zug müssen die Reihen aktualisiert werden
// um eventuell einen Sieger finden zu können
/*****/
public void verwalteReihen(int bn) // übergeben wird die Nummer des Buttons
{
if(weristdran==1) // Spieler ist dran
{
if ( bn<3)
{reihen[0]++;reihen[bn+3]++;} // für die Felder 0, 1 und 2
else

```

```

    {
        if ((bn<6)&&(bn>2))
            {reihen[1]++;reihen[bn]++;} // für die Felder 3,4 und 5
        else
            {reihen[2]++;reihen[bn-3]++;} // für Felder 6 bis 8 und Ende von if ((bn<6)&&(bn>2))
    } // Ende von if ( bn<3)

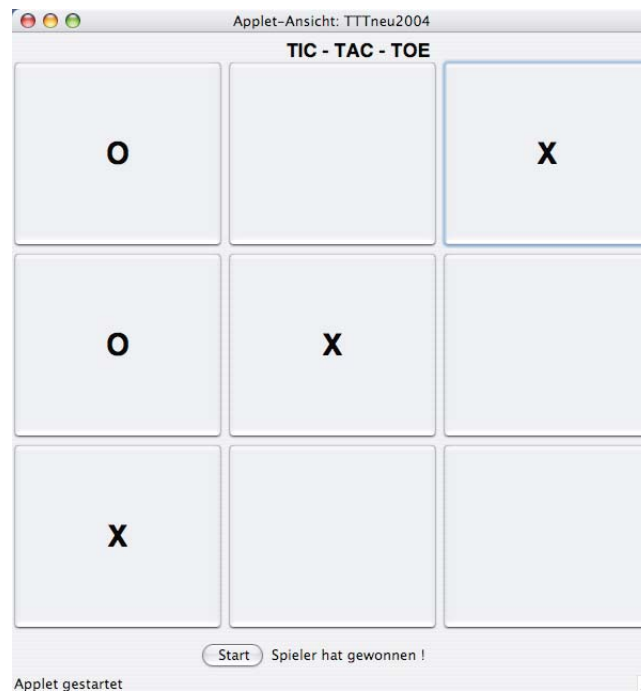
    if((bn==0) || (bn==8)) {reihen[6]++;} // Spezialfälle für die beiden Diagonalen
    if((bn==2) || (bn==6)) {reihen[7]++;}
    if(bn==4) {reihen[6]++;reihen[7]++;}
} // ende von weristdran = 1
else
{
    if ( bn<3)
        {reihen[0]- ;reihen[bn+3]- ;}
    else
        {if ((bn<6)&&(bn>2))
            {reihen[1]- ;reihen[bn]- ;}
        else
            {reihen[2]- ;reihen[bn-3]- ;}
        }

    if((bn==0) || (bn==8)) {reihen[6]- ;}
    if((bn==2) || (bn==6)) {reihen[7]- ;}
    if(bn==4) {reihen[6]- ;reihen[7]- ;}
}
}
//*****
// Nun wird geprüft, ob einer gewonnen hat
//
//*****
public void gibtesSieg()
{
    int k;
    for(k=0;k<8;k++)
    {
        if(reihen[k]==3)
            {spielerSieg=true;}
        if(reihen[k]==-3)
            {computerSieg=true;}
        if((zugzahl==9)&&!computerSieg)&&!spielerSieg)
            {unentschieden=true;}
    } // Ende von for
} // Ende von gibtesSieg

} // Ende vom Applet

```

Wenn alles klappt, dann kann man mit diesem Programm schon mal spielen. Allerdings ist die Zufallsmethode, mit der der Rechner setzt wenig geeignet, eine Sieg zu erringen. Anders gesagt: *"Der Rechner spielt ganz schön blöd."*



Aber mit den Reihensummen muss sich doch was machen lassen !

### Aufgabe:

Entwickle eine Methode, die den Rechner dazu bringt, nachzuprüfen, ob für eine Reihe schon die Summe +2 erreicht ist. Wenn ja, dann soll auf das freie Feld in dieser Reihe gesetzt werden, um den Sieg des Spielers zu verhindern.

### Aufgabe:

Entwickle eine Methode, die den Rechner dazu bringt, nachzuprüfen, ob für eine Reihe schon die Summe -2 erreicht ist. Wenn ja, dann soll auf das freie Feld in dieser Reihe gesetzt werden um zu siegen !

### Aufgabe:

Entwickle mit den oben geschilderten Möglichkeiten - oder ganz anderen Verfahren - ein funktionierendes TIC-TAC-TOE-Programm, das nicht mehr verliert.