Stefan Feldmann

Projekt „Mühle“ (Gesellschaftsspiel)

Eine Facharbeit im 4. Kurshalbjahr des Informatikunterrichts.

Name:	Feldmann
Vorname:	Stefan
Schule:	Gymnasium Francisceum Zerbst
Klasse:	Informatik 12_1, 4. Kurshalbjahr – „Projektarbeit“
Kurslehrer:	Hr. Günter Ritzmann
Abgabedatum:	12. März 2007

Vorwort

Diese Facharbeit soll eine umfassende Erklärung zu dem Projekt „Gesellschaftsspiel Mühle“ und damit eine finale Abschlussarbeit des 4. Kurshalbjahres im Kursunterricht Informatik 2007 darstellen.

Dieser Facharbeit liegt ein Datenträger bei, der folgende Ordnerstrukturen enthält:

/Facharbeit/Programme/Mühle_02_19_2007

/Facharbeit/Programme/Mühle_02_27_2007

/Facharbeit/Programme/Mühle_02_28_2007

/Facharbeit/Programme/Mühle_03_06_2007

/Facharbeit/Programme/Mühle_03_11_2007

/Facharbeit/Programme/Mühle_Endfassung

/Facharbeit/Dokumente

/Facharbeit/Veröffentlichung

Die in dieser Facharbeit enthaltenen Quelltextabschnitte beziehen sich auf die Programmversion, die sich in der Ordnerstruktur „/Facharbeit/Programme/Mühle_03_07_2007“ befindet.

Die endgültige Programmfassung, welche allerdings besser zum Spielen geeignet ist, und daher einfacher zu bedienen ist (weniger Nachrichtendialoge etc.), befindet sich in der Ordnerstruktur „/Facharbeit/Programme/Mühle_Endfassung“.

Die Ordnerstruktur „/Facharbeit/Dokumente“ beinhaltet die für die allgemeinen Texte notwendigen Informationen, die zur Bewertung gedacht sind (beinhaltet diese Facharbeit als Worddatei).

Sollte das Vorhaben bestehen, diese Facharbeit zu veröffentlichen, so bitte ich darum, nur die unter der Ordnerstruktur „/Facharbeit/Veröffentlichung“ bereitgestellten Dateien zur Verfügung zu stellen. Diese sind entweder schreibgeschützt oder liegen im Adobe Reader-Format vor.

Sofern Interesse an der Programmentstehung oder Probleme bei Programmteilen oder Teilen der Facharbeit bestehen, bitte ich darum, mich unter stefanfeldmann@hotmail.de zu kontaktieren.

Zerbst, den 11. März 2007

Stefan Feldmann.

Inhalt

Das Gesellschaftsspiel „Mühle“

Das Mühlespiel ist ein altes Brettspiel für zwei Personen (meist Schwarz und Weiß), bei dem der Zufall keine Rolle spielt. Ziel dieses Spieles ist, sog. Mühlen zu schließen und dem Gegner einen nach dem anderen Stein zu nehmen. Derjenige, der zum Schluss nur noch zwei Steine besitzt, hat verloren. Mühle wird auf einem Spielbrett gespielt, welches aus drei ineinander liegenden Quadraten mit Verbindungslinien in den Seitenmitten besteht.

Das Mühlespiel gehört neben dem Mensch-ärgere-dich-nicht-Spiel zu den bekanntesten Brettspielen in Deutschland. Als Spielfiguren werden gewöhnlich neun schwarze und neun weiße runde, flache Spielsteine verwendet, die meist aus Holz oder Kunststoff gemacht sind. Andere Farben sind allerdings auch möglich.

Allgemeine Informationen

Das Mühlespiel ist ein Spiel mit vollständiger Information. Es ist gerecht, d.h. es konnte nachgewiesen werden, dass weder der An- noch der Nachziehende zwingend gewinnen kann. Bei gleichstarken erfahrenen Gegnern endet das Spiel deshalb oft unentschieden, wobei der nachziehende Spieler im Gegensatz zum Schach im Vorteil ist, weil er den letzten Stein auf das Brett setzen darf. Der nachziehende Spieler kann beim Setzen des letzten Steins einen möglichen Zugzwang berechnen. Bereits 1993 wurde das Spiel von Ralph Gasser an der ETH Zürich erstmals komplett gelöst. Der Informatiker Peter Stahlhacke berechnete das Spiel auf seinem Heimrechner neu. Die 17 GB große Datenbank mit allen Stellungen wurde auf der Spielseite Inetplay als das perfekt Mühle spielende Programm Mr. Data veröffentlicht. Das spielstärkste Spiel heißt Mühle24 und spielt offensiver als Mr. Data. Es ist nicht fehlbar (d.h. es kann verlieren) und leidet unter veraltetem Grafikdesign, verblüfft aber mit unerwarteten Spielzügen und bringt auch erfahrene Spieler in Schwierigkeiten.

Geschichte

In Europa ist das Mühlespiel seit der Bronzezeit bekannt. In Deutschland wurden verschiedene Varianten der Mühle, wie z.B. die Neunermühle, die Rad- oder Rundmühle, sowie die Dreiermühle bei Ausgrabungen römischer Grenzbefestigungen entdeckt. Aber auch in China ist das Spiel seit ca. 2000 Jahren bekannt. Vermutlich gehört dieses Taktikspiel zu den ältesten Brettspielen überhaupt.

Abbildung 1 - Radmühle

Vom 12. Bis zum 18. Jahrhundert gehörte das Mühlespiel zu den beliebtesten Brettspielen in Europa. Erst ab Anfang des 19. Jahrhunderts wurde die Mühle vom Schachspiel nach und nach verdrängt.

Spielverlauf und Standardregeln

Beim Mühlespiel spielen generell zwei Personen. Das Standard-Spielbrett besteht aus drei konzentrisch liegenden Quadraten, die in den Seitenmitten verbunden sind. Die Eckpunkte und die Mittelpunkte bilden 24 Felder, auf die die 18 Spielsteine gestellt werden. 12 Felder haben 2 Nachbarfelder, 8 haben 3 und 4 haben 4 Nachbarfelder. Drei Spielfelder bilden 16mal eine Strecke mit einem Mittelpunkt. Stehen auf diesen 3 Feldern Steine, so bilden sie eine „Mühle“.

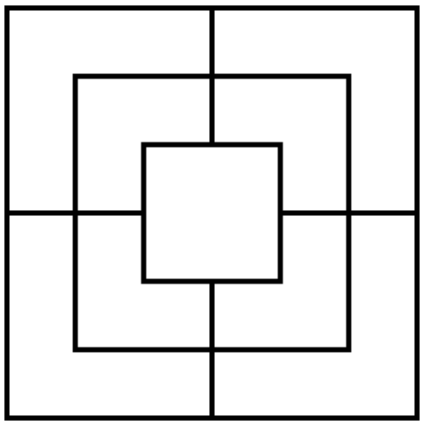


Abbildung 2 - Spielfeld des Spieles

Die beiden Spieler bekommen je neun weiße und neun schwarze Spielsteine. Zu Beginn ist das Brett leer. Weiß beginnt den ersten Spielzug. Wer Weiß wird, bestimmt normalerweise das Los.

Im Mühlespiel unterscheidet man für gewöhnlich drei Phasen:

Phase 1 – Anfangs-/ Setzphase

In der Setzphase setzen die Spieler abwechselnd je einen Stein auf ein freies Feld.

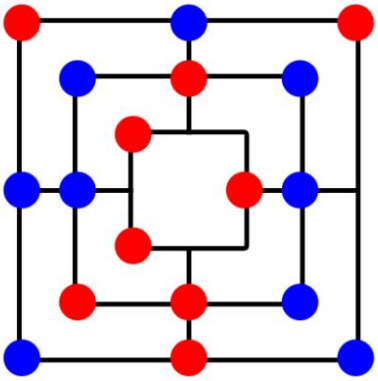


Abbildung 3 - Spiel nach Setzphase

Insbesondere in diesem Abschnitt des Spiels ist es weniger wichtig, frühzeitig Mühlen zu bilden, als vielmehr, eine große Beweglichkeit seiner Steine sicherzustellen. So sind die vier Kreuzungspunkte des Mühlebrettes bevorzugt zu besetzen, während die Eckpunkte zu vermeiden sind.

Eine Mühle besteht aus drei Steinen, die in einer Reihe liegen. Drei diagonal liegende Steine bilden keine Mühle. Gelingt einem Spieler eine Mühle, so darf er einen Stein des Gegners vom Brett und damit aus dem Spiel nehmen. Steine aus einer Mühle darf er nicht nehmen, sie sind geschützt.

Sind alle Steine gesetzt, so könnte das Spiel wie dargestellt aussehen.

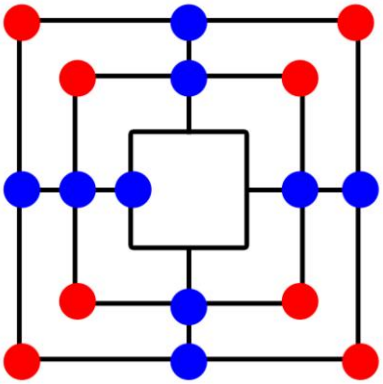


Abbildung 4 - Blockade

Phase 2 – Mittel-/ Zugphase

Befinden sich nun alle Steine auf dem Spielbrett, so werden sie abwechselnd längs einer Linie um ein Feld verschoben. Dafür stehen sechs freie Feder zur Verfügung. Wieder achtet jeder darauf, dass der gegnerische Spieler keine Mühle erstellt und dass er selbst nicht blockiert wird.

Was man unter blockieren versteht, zeigt die Abbildung 4. Das Spiel ist somit schon zu Ende.

Weiterhin gibt es eine Mühlenstellung, die bereits sprichwörtlich geworden ist. Das ist die sog. Zwickmühle. Hier schließt ein Stein abwechselnd zwei Mühlen fortwährend. Der Besitzer der Mühle darf während jedem Zug einen gegnerischen Stein nehmen. Zwei Arten von Zwickmühlen sind möglich.

Kommt es nun zu keiner Blockade des anderen und gelingt es, Mühlen zu bauen, so wird die Zahl der Steine immer kleiner. Schließlich hat ein Spieler nur noch drei Steine. Dann beginnt die dritte Phase.

Phase 3 – End-/ Sprungphase

In der letzten Phase darf der Spieler mit drei Steinen jedes freie Feld besetzten. Deshalb ist es günstig, wenn der gegnerische Spieler möglichst zwei Mühlen offenhält, um von den drei Steinen einen entfernen zu können. Dann ist das Spiel zu Ende. Gelingt es nicht, den dritten Stein noch zu nehmen, so kann der Spieler mit den drei Steinen noch gewinnen.

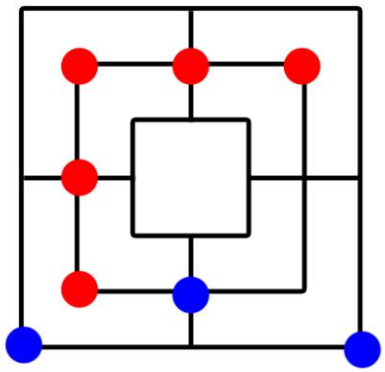


Abbildung 5 - Sonderfall 1 - alles Mühlen

Sonderfälle

Außerdem gibt es Zwei Spielsituationen, die durch die o.g. Spielregeln nicht erfasst werden können.

Angenommen, ein Spieler schließt gerade eine Mühle. Die Steine des anderen Spielers befinden sich alle in verschiedenen Mühlen. Dann müsste dieser Spieler einen Stein abgeben, was allerdings der Regel widerspricht, dass Steine in einer Mühle geschützt sind. Dieser Fall ist so gelöst, dass der Spieler, dessen Steine sich alle in einer Mühle befinden, dennoch einen Stein abgeben muss.

Ein weiterer Sonderfall wäre, dass ein Spieler in der ersten Phase mit einem Stein gleichzeitig zwei Mühlen schließt. Dieser Fall ist so geregelt, dass trotzdem nur ein Stein genommen werden darf.

Programmumsetzung des Projekts „Mühle“

Vorüberlegungen/ Vereinfachungen

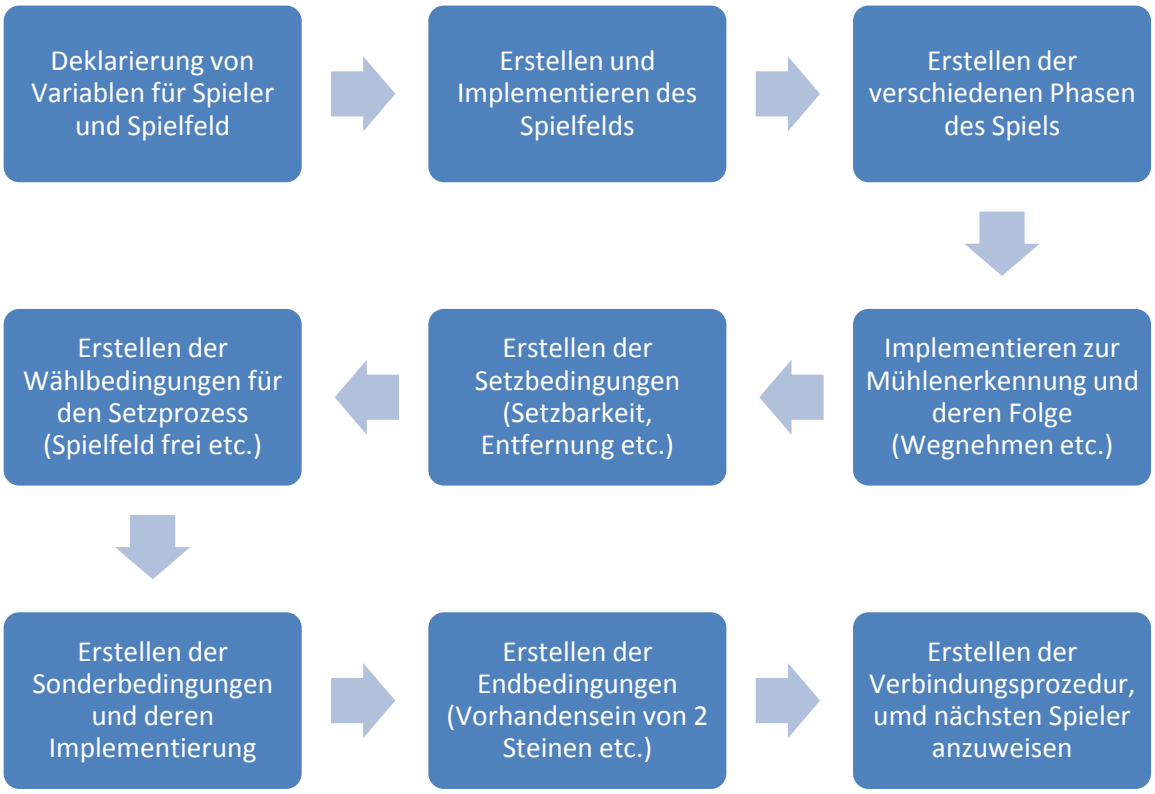
Ziel der Programmumsetzung des Spiels „Mühle“ soll sein, eine sinnvolle Reproduktion unter Beachtung aller gängigen Spielregeln des Spiels zu erstellen.

Für diese Reproduktion soll zunächst der Computergegner unter Verwendung von zwei realen Spielern ausgeschlossen werden.

Außerdem soll die Reproduktion des Spiels in einer ansprechenden zweidimensionalen Ansicht erfolgen, die unter der Programmiersprache „Turbo Pascal“ bei Verwendung des Programmes „Borland Delphi 7.0“ erstellt wird.

Ablauf der Umsetzung

Um eine Umsetzung des Gesellschaftsspiels zu garantieren, wurde folgender Ablauf der Umsetzung erstellt:



Dieser Ablaufplan stellt die allgemeine Grundstruktur für die Arbeit an dem Programm dar. Im Folgenden sollen die wesentlichen Bedingungen für die einzelnen Prozesse sowie den allgemeinen Quelltext erläutert werden.

Quelltexterläuterung

Festlegung der Prozeduren

Zusätzlich zu den bereits benötigten Prozeduren kommen noch einige selbsterstellte Prozeduren, die verschiedene, sich wiederholende Schritte, einbinden sollen.

FormCreate: „procedure FormCreate“ beinhaltet die Anweisungen, die bei Erstellen der Form ausgeführt werden sollen.

FormPaint: „procedure FormPaint“ ordnet an, in der Form einzelne Pixel erstellen zu können.

FormMouseMove:	„procedure FormMouseMove“ kann die aktuelle Mauszeigerposition ermitteln und erleichter die Feststellung der gewählten Figuren.
FormClick:	„procedure FormClick“ soll die Anweisungen beinhalten, die ausgeführt werden sollen, wenn auf die Form geklickt wird.
Suche_Muehle:	„procedure Suche_Muehle“ ermittelt, ob sich auf dem Spielfeld Mühlen befinden.
NaechsterSpieler:	„procedure NaechsterSpieler“ führt an bestimmten Stellen den Wechsel der Spieler aus.
Setzbar:	„procedure Setzbar“ prüft, ob der aktuell gewählte Stein setzbar ist.
Wegnehmen_Moeglich:	„procedure Wegnehmen_Moeglich“ prüft, ob der Stein, der aufgrund einer neu erstellten Mühle entnommen werden soll, sich auch nicht in einer Mühle befindet.
Setzen:	„procedure Setzen“ prüft, ob die neue Zielposition des gewählten Steins auch eine den Spielregeln entsprechende Position ist.

Der Quelltext, der diese Prozeduren in die Form einbinden soll, sieht wie folgt aus:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  Tform1 = class(TForm)
    Informationen: TMemo;
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
```

```
    Y: Integer);
  procedure FormClick(Sender: TObject);
  procedure suche_muehle;
  procedure NaechsterSpieler;
  procedure setzbar (i:integer);
  procedure wegnehmen_moeglich;
  procedure setzen (i,j:integer);
 private
  { Private-Deklarationen }
 public
  { Public-Deklarationen }
 end;
```

Deklaration der globalen Variablen

Um eine einfache Erstellung des Gesellschaftsspiels zu garantieren, müssen verschiedene Variablen deklariert werden. Dazu gehören folgende Deklarationen:

TPunkt: TPunkt wurde als Recordstruktur gewählt und umfasst die Bytevariablen „x“ und „y“. TPunkt weist verschiedenen Variablen ihre x- und y-Position zu.

TFigur: TFigur wurde als Recordstruktur gewählt, um den verschiedenen, einzelnen Figuren verschiedene Eigenschaften zuzuweisen. TFigur weist einer Figur die Eigenschaften „Pos“ (als Positionsausgabe der Figur über TPunkt), „gesetzt“ (als Zuweisung, ob die Figur bereits gesetzt ist) und „setzbar“ (als Zuweisung, ob die Figur gesetzt ist) zu.

TSpieler: TSpieler umfasst eine Recordstruktur, um den Spielern die jeweiligen Eigenschaften zuzuordnen. Dazu gehören „Anzahl_Figuren“ (Anzahl der Figuren außerhalb des Spielfelds zu Beginn), „Figur“ als Feldstruktur von „TFigur“ (um die neun Figuren den Spielern zuzuweisen), „Farbe“ (um die jeweilige Farbe des Spielers zuzuweisen) und „Anzahl_Figuren_Innen“ (um die Figuren innerhalb des Spielfelds zu zählen).

TMuehle: Die Recordstruktur TMuehle weist den verschiedenen Mühlenpositionen die Werte „vorhanden“ (ob die Mühle vorhanden ist), „Spieler“ (welcher Spieler diese Mühle besetzt) und „genommen“ (ob bereits aufgrund dieser Mühle ein Stein genommen wurde) zu.

TSpielfeld: TSpielfeld ist eine Recordstruktur, welche die Werte „Spieler“ (welcher Spieler das jeweilige Feld besetzt), „Bewertung“ (um zu bestimmen, ob das Feld besetzt ist oder nicht) und „Muehle“ (ob sich dort eine Mühle befindet).

Spielfeld: Die Feldstruktur „Spielfeld“ vereint die Recordstruktur „TSpielfeld“ in einem zweidimensionalen Feld, um die aktuellen Positionen der Felder zu bestimmen.

MouseX: MouseX ist der Integerwert, der später die x-Position des Mauszeigers bestimmt.

MouseY: MouseY ist der Integerwert, der später die y-Position des Mauszeigers bestimmt.

Akt_Spieler: Die Integervariable „Akt_Spieler“ merkt sich den aktuellen Spieler.

Spieler: Die Feldstruktur Spieler vereint die Recordstruktur „TSpieler“, um diese Struktur zwei verschiedenen Spielern zuzuordnen.

Muehle: Die Feldstruktur „Muehle“ soll für die 16 verschiedenen Mühlepositionen die Eigenschaften, die durch „TMuehle“ gegeben werden, zuordnen.

Akt_Muehle: „Akt_Muehle“ gibt die soeben erzielte Mühle an, falls eine Mühle erzielt wurde.

Gewaehlt: „Gewaehlt“ gibt den aktuell zum Setzen gewählten Spielstein an.

Anzahl_Setzbar: „Anzahl_Setzbar“ ermittelt die aktuelle Anzahl der setzbaren Figuren.

Richtig_Gesetzt: „Richtig_Gesetzt“ gibt an, ob die neue in der Setzphase gewählte Position auch den Spielregeln entspricht.

Anzahl_Muehlen: „Anzahl_Muehlen“ ermittelt, wie viele Steine sich gerade in einer Mühle befinden.

Der Quelltext, der die oben angegebenen Variablen deklariert, sieht wie folgt aus:

```
type  TPunkt = record
  x: Byte;
  y: Byte;
  end;
 TFigur = record
  Pos: TPunkt;
  gesetzt:byte;
  setzbar: boolean;
  end;
 TSpieler = record
```

```
  Anzahl_Figuren: integer;
  Figur: Array[1..9] of TFigur;
  Farbe: TColor;
  Anzahl_Figuren_Innen: Integer;
  end;
 TMuehle = record
  vorhanden: boolean;
  Spieler: Integer;
  genommen: boolean;
  end;
 TSpielfeld = record
  Spieler: Integer;
  Bewertung: Integer;
  Muehle: boolean;
  end;
var
 form1: Tform1;
 spielfeld: array[0..6,0..6] of TSpielfeld;
 mousex, mousey: integer;
 akt_Spieler: Integer;
 Spieler: array[1..2] of Tspieler;
 muehle: array [1..16] of TMuehle;
 akt_muehle: integer;
 gewaehlt: integer;
 anzahl_setzbar: integer;
 richtig_gesetzt: boolean;
 anzahl_muehlen: integer;
```

„Procedure FormCreate“

In der Prozedure FormCreate sollen verschiedene Anfangsbedingungen erzeugt werden. Dazu gehören die Bedingung, dass der beginnende Spieler der Spieler Weiß (also „1“) sein wird, die aktuell setzbaren Figuren „0“ betragen und verschiedene Größen festgelegt werden (z.B. Höhe und Breite des Spielfeld).

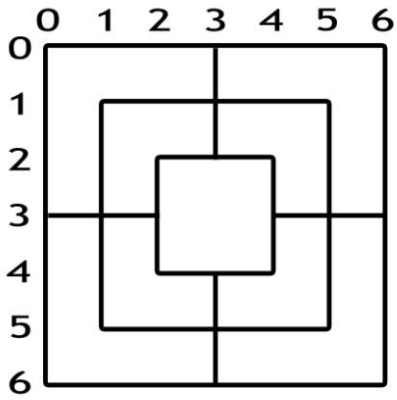


Abbildung 6 - Nummerierung der Achsen

Außerdem wird zunächst festgelegt, dass alle Positionen auf dem Spielfeld am Anfang des Spiels unbesetzt sind, also kein Spieler bis jetzt einen Stein gesetzt hat und keine Mühle besteht.

Weiterhin erfolgt in dieser Prozedur die Festlegung des Spielfelds. Das Spielfeld besteht aus 7*7=49 verschiedenen Kontaktpunkten. Diese Kontaktpunkte werden zunächst erstellt, sie sollen, wenn sie für das Spiel relevant sind (also besetzbar sind) die Bewertung „1“ erhalten. Dazu werden zwei Laufvariablen deklariert, die allen Positionen auf dem Spielfeld (von „spielfeld[0,0]“ bis „spielfeld[6,6]) zunächst die Bewertung „1“ zuweisen (d.h. alle Spielfeldpositionen sind nun besetzbar). Als nächstes werden die für das Spiel irrelevanten Positionen „gelöscht“, d.h. ihnen wird die Bewertung „0“ zugeordnet (d.h. diese Positionen sind nicht besetzbar).

Zuletzt werden den jeweiligen Spielern die Anzahl der Startfiguren und die Farbe zugeordnet.

Der Quelltext, der diese Anweisungen beinhaltet, ist folgender:

```
procedure Tform1.FormCreate(Sender: TObject);
var i,j:integer;
begin
for i:=1 to 2 do
for j:=1 to 9 do
begin
 spieler[i].Figur[j].gesetzt:=0;
 spieler[i].Figur[j].setzbar:=false;
 end;
for i:=0 to 6 do
for j:=0 to 6 do
```

```
begin
 spielfeld[i,j].Spieler:=0;
 spielfeld[i,j].Muehle:=false;
end;
akt_Spieler:=1;
anzahl_setzbar:=0;
Informationen.Lines.Add('Spieler "Weiß" beginnt!');
form1.width:=609;
form1.height:=750;
//Initialisierung des Spielfelds
 for i:=0 to 6 do
 for j:=0 to 6 do
  spielfeld[i,j].Bewertung:=1;



 for i:= 1 to 9 do
 begin
 spieler[1].Figur[i].gesetzt:=0;
 spieler[2].Figur[i].gesetzt:=0;
 end;


 for i:=1 to 2 do
 begin
 spielfeld[0,i].Bewertung:=0;
 spielfeld[0,i+3].Bewertung:=0;
 spielfeld[i,0].Bewertung:=0;
 spielfeld[i+3,0].Bewertung:=0;
 spielfeld[6,i].Bewertung:=0;
```

```
 spielfeld[6,i+3].Bewertung:=0;

 spielfeld[i,6].Bewertung:=0;

 spielfeld[i+3,6].Bewertung:=0;

 end;

spielfeld[1,4].Bewertung:=0;

spielfeld[1,2].Bewertung:=0;

spielfeld[5,4].Bewertung:=0;

spielfeld[5,2].Bewertung:=0;

spielfeld[4,1].Bewertung:=0;

spielfeld[2,1].Bewertung:=0;

spielfeld[4,5].Bewertung:=0;

spielfeld[2,5].Bewertung:=0;

spielfeld[3,3].Bewertung:=0;


spieler[1].anzahl_figuren:=9;

spieler[1].farbe:=clwhite;


spieler[2].anzahl_figuren:=9;

spieler[2].farbe:=clblack;


end;
```

„Procedure FormPaint“

“FormPaint” enthält die Anweisungen, die nötig sind, um das Spielfeld zu zeichnen. Dazu werden zur Zierde 3 verschiedene Rahmen um das Spielfeld gelegt. Dann wird die typische Struktur des Spielfelds erstellt (drei konzentrisch liegende Quadrate sowie die Verbindungslinien).

Die Kontaktpunkte werden durch kleine Kreise dargestellt, die sich an allen Positionen befinden, an welchen die Bewertung „1“ vorliegt.

Außerdem beinhaltet die Prozedur „Form Create“ die nötigen Werte, welche die entsprechenden Spielfiguren zeichnen. Sie fragt ab, ob die Figur gesetzt ist und zeichnet an der entsprechenden Stelle einen Kreis in der Farbe der Spielfigur, der die Spielfigur darstellt.

Der entsprechende Quelltext lautet:

```
procedure Tform1.FormPaint(Sender: TObject);
var i,j: integer;
begin
with form1.canvas do
begin
 // Zeichnen des Spielfelds
 pen.color:=clblack;
 pen.width:=6;
 brush.style:=bsclear;
 rectangle(3,3,600,600);
 pen.width:=3;
 rectangle(8,8,600-7,600-7);
 pen.width:=1;
 rectangle(12,12,600-11,600-11);
 pen.width:=3;
 rectangle(75,75,525,525);
 rectangle(150,150,450,450);
 rectangle(225,225,375,375);
 moveto(300,75);
 lineto(300,225);
 moveto(300,375);
 lineto(300,524);
 moveto(75,300);
```

```
lineto(225,300);
moveto(375,300);
lineto(524,300);
 brush.color:=clwhite;
//Zeichnen der Eckpunkte
for i:=0 to 6 do
for j:=0 to 6 do
 begin
 if spielfeld[i,j].Bewertung>=1 then
 ellipse(65+75*i,65+75*j,85+75*i,85+75*j);
 end;

for i:=1 to 9 do
begin
if spieler[1].Figur[i].gesetzt=1 then begin
 canvas.Brush.color:=clwhite;
ellipse((spieler[1].Figur[i].pos.x)*75+55,(spieler[1].Figur[i].pos.y)*75+55,(spieler[1].Figur[i].pos.x)*75
+95,(spieler[1].Figur[i].pos.y)*75+95);
 end;
if spieler[2].Figur[i].gesetzt=1 then begin
 canvas.Brush.color:=clblack;
ellipse((spieler[2].Figur[i].pos.x)*75+55,(spieler[2].Figur[i].pos.y)*75+55,(spieler[2].Figur[i].pos.x)*75
+95,(spieler[2].Figur[i].pos.y)*75+95);
 end;
 end;
end;
end;
```

„Procedure FormMouseMove“

Mit dieser Prozedur soll die aktuelle Mauszeigerposition globalisiert werden, damit sie für die folgenden Prozeduren verwendet werden kann.

Dieser Quelltext lautet:

```
procedure Tform1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
mousex:=x;
mousey:=y;
form1.caption:='x-Wert: '+inttostr(mousex)+', y-Achse: '+inttostr(mousey);
end;
```

„Procedure FormClick“

Diese Prozedur ist die wichtigste des gesamten Projekts. Sie vereint alle Phasen des Spiels und beinhaltet alle Problematiken des Gesellschaftsspiels „Mühle“.

Zu diesen Problematiken gehören folgende Themen:

- Entnehmen eines Steines bei erfolgter Mühle
- Wählen eines Steines zum Ziehen
- Ziehen des gewählten Steines
- Wählen eines Steines, wenn die Anzahl der inneren Figuren unter vier liegt (dieser Stein wäre dann jedenfalls setzbar)
- Ziehen eines Steines, wenn die Anzahl der inneren Figuren unter vier liegt (dieser Stein könnte überall gezogen werden)
- Setzen der Steine, wenn die Anzahl der inneren Steine noch nicht 9 beträgt (d.h. wenn sich noch nicht alle Steine im Spiel befinden)
- Endbedingung des Spiels (muss nach jedem Zug überprüft werden)

Diese Problematiken müssen in der Reihenfolge abgefragt werden, wie sie oben formuliert wurden, Da die jeweiligen Abfragen z.B. für das Entnehmen eines Steins erst erfolgen können, wenn die Prozedur „FormClick“ ein neues Mal geöffnet werden.

Um das Verständnis des Programms aber zu vereinfachen, sollen die Bausteine, die erklärt werden in der Reihenfolge abgehandelt werden, wie sie auch im Spiel erfolgen.

Setzen der Steine auf das Spielfeld

Das Setzen der Steine erfolgt zum Beginn des Gesellschaftsspiels „Mühle“.

Zunächst ermittelt das Programm, ob die mit der Maus geklickte Position mit einem der Kontaktpunkte bzw. Figurpositionsmöglichkeiten übereinstimmt.

Die unmittelbare Bedingung für das Setzen eines Steins muss lauten, dass sich noch nicht alle Steine des Spielers im Spiel befinden. Es muss also abgefragt werden, ob die Anzahl der Figuren des aktuellen Spielers noch nicht 9 beträgt bzw. ob die Anzahl der Figuren, die sich noch außerhalb vom Spielfeld befinden, noch nicht null beträgt, andernfalls soll der Setzvorgang beginnen.

Ist diese Bedingung erfüllt, wird abgefragt, ob der gewählte Kontaktpunkt auch unbesetzt ist und überhaupt existiert. Wenn dies nicht der Fall ist, soll das Programm bestätigen, dass diese Position bereits besetzt ist.

Ist nun der gewählte Kontaktpunkt besetzbar, so soll eine Figur gesetzt werden. Dazu wird die Anzahl der Außenfiguren dekrementiert, die Position der neuen Figur übernommen (entsprechend der Laufvariablen), die Eigenschaft „gesetzt“ bestätigt, die Anzahl der Innenfiguren inkrementiert, der Spielfeldposition die aktuelle Bewertung sowie der aktuelle Spieler zugeordnet. Dann sollen die Prozeduren „Repaint“ (welche die Form neu zeichnet) sowie „Suche_Muehle“ (ob eine neue Mühle erreicht wurde) durchgeführt werden. Wenn keine neue Mühle erkannt wurde, soll der Nächste Spieler an der Reihe sein, sonst nicht, da sonst der aktuelle Spieler noch einen Stein entnehmen muss.

Zuletzt erfolgt die Überprüfung, ob der Setzvorgang nun gestartet werden kann (falls die Anzahl der Innenfiguren nun 9 beträgt und die der Außenfiguren 0).

Der Quelltext, der das Setzen der Figuren auf das Spielfeld beinhaltet, sieht wie folgt aus:

```
for i:=0 to 6 do
for j:=0 to 6 do
 begin
 if (mousex > 55+75*i) and (mousex < 95+75*i) and (mousey > 55+75*j) and (mousey < 95+75*j)
 then
 begin
  if (spieler[akt_spieler].Anzahl_Figuren>=1) then
   begin
    if spielfeld[i,j].Bewertung=1 then
     begin
```

```
      spieler[akt_spieler].Anzahl_Figuren:=spieler[akt_spieler].Anzahl_Figuren-1;
      spieler[akt_spieler].Figur[9-(spieler[akt_spieler].Anzahl_Figuren)].Pos.x:=i;
      spieler[akt_spieler].Figur[9-(spieler[akt_spieler].Anzahl_Figuren)].Pos.y:=j;
      spieler[akt_spieler].Figur[9-(spieler[akt_spieler].Anzahl_Figuren)].gesetzt:=1;
      spieler[akt_spieler].Anzahl_Figuren_Innen:=spieler[akt_spieler].Anzahl_Figuren_Innen+1;
      spielfeld[i,j].Bewertung:=2;
      spielfeld[i,j].Spieler:=akt_spieler;
      repaint();
      suche_muehle;
      if akt_muehle=0 then
      NaechsterSpieler;
      if spieler[akt_spieler].Anzahl_Figuren=0 then
       Informationen.Lines.Add('Ab nun beginnt der Setzvorgang!');
      end else
     begin
      if spielfeld[i,j].Bewertung=2 then
      begin
      Informationen.Lines.Add('Hier befindet sich bereits ein Stein!');
      showmessage('Hier befindet sich bereits ein Stein!');
      end;
 end;
end;
end;
```

Entnehmen eines Steines aufgrund einer neuen Mühle

Aufgrund der Anweisungen bezüglich des Setzens eines Steines auf das Spielfeld wurde die Prozedur „FormClick“ im Falle einer neuen Mühle neu gestartet. Damit wurde der Spieler nicht gewechselt, denn der Spieler, der die Mühle erzeugt hat, soll auch einen Stein entfernen.

Dazu wird mit der Prozedur „Wegnehmen_moeglich“ zunächst überprüft, inwiefern die Steine des gegnerischen Spielers wegnehmbar sind. Diese Prozedur gibt auch den Wert „anzahl_muehlen“ aus, welcher zur folgenden Überprüfung verwendet wird.

Es folgt eine Überprüfung, ob die Anzahl der gegnerischen Figuren auch der Anzahl der Figuren des Gegners entspricht, die sich in einer Mühle befinden. Das bedeutet, dass überprüft wird, ob sich alle Steine des Gegners in Mühlen befinden. Sollte dies der Fall sein, so beginnt die Entnahme eines Spielsteins wenn ein entsprechender gewählt wurde, ohne dass überprüft wird, ob sich dieser Stein in einer Mühle befindet (da ja in jedem Falle ein Stein aus einer Mühle entfernt werden muss).

Zur Entnahme eines Steins werden verschiedene Abläufe implementiert: Die Mühle wird als „genommen“ bezeichnet, entsprechend dem Spieler wird die Spielfeldbewertung auf „1“ gesetzt und der Spieler auf „0“ (es befindet sich kein Stein mehr an dieser Stelle), die Anzahl der Innenfiguren wird dekrementiert. Weiterhin wird aus der Variable „test“ abgelesen, ob soeben noch eine zweite Mühle „also „Doppelmühle“) gleichzeitig erzeugt wird. Ist dies der Fall, so wird auch diese als „genommen“ betrachtet, damit kein zweimaliges Entnehmen eines Steins erfolgt.

Zuletzt werden der entfernten Spielfigur noch die Werte zugewiesen, die nötig sind, um zu sagen, dass sich diese nicht mehr im Spiel befindet.

Anschließend erfolgt das Ende der Einweisungen für den Fall, dass sich alle Steine des Gegners in Mühlen befinden. Es folgt die Implementierung für den Fall, dass sich nicht alle gegnerischen Spielsteine in einer Mühle befinden. Diese Implementierung ist die gleiche, es erfolgt nur noch eine kurze Abfrage, ob sich der zu entnehmende Stein in einer Mühle befindet. Ist dem so, so darf er nicht entnommen werden (er ist „geschützt“), sonst ja.

Der Quelltext zum Entnehmen einer Mühle mit allen Bedingungen ist der folgende:

```
if (akt_muehle<>0) and (anzahl_muehlen=spieler[n+1].Anzahl_Figuren_Innen) then
begin
for i:=0 to 6 do
for j:=0 to 6 do
 begin
  if (mousex > 55+75*i) and (mousex < 95+75*i) and (mousey > 55+75*j) and (mousey < 95+75*j)
  then
  begin
  Muehle[akt_muehle].genommen:=true;
  if test>akt_muehle then
  muehle[test-akt_muehle].genommen:=true;
```

```
wegnehmen_moeglich;
if (spielfeld[i,j].Spieler=2) and (akt_spieler=1) then
begin
 spielfeld[i,j].Bewertung:=1;
 spielfeld[i,j].Spieler:=0;
 Spieler[2].Anzahl_Figuren_innen:=Spieler[2].Anzahl_Figuren_innen-1;
 if (Spieler[2].Anzahl_Figuren_innen<=3) and (spieler[2].Anzahl_Figuren=0) then
 informationen.Lines.add('Für Schwarz beginnt nun der Sprungvorgang!');
 for k:=1 to 9 do
  if (Spieler[2].Figur[k].Pos.x=i) and (Spieler[2].Figur[k].Pos.y=j) then
   Spieler[2].Figur[k].gesetzt:=0;
 Repaint();
 akt_muehle:=0;
end;
 if (spielfeld[i,j].Spieler=1) and (akt_spieler=2) then
begin
 spielfeld[i,j].Bewertung:=1;
 spielfeld[i,j].Spieler:=0;
 Spieler[1].Anzahl_Figuren_innen:=Spieler[1].Anzahl_Figuren_innen-1;
 if (Spieler[1].Anzahl_Figuren_innen<=3) and (spieler[1].Anzahl_Figuren=0) then
 informationen.Lines.add('Für Weiß beginnt nun der Sprungvorgang!');
 for k:=1 to 9 do
  if (Spieler[1].Figur[k].Pos.x=i) and (Spieler[1].Figur[k].Pos.y=j) then
   Spieler[1].Figur[k].gesetzt:=0;
 Repaint();
 akt_muehle:=0;
end;
end;
```

```
  end;
 end else


if (akt_muehle<>0) then
begin
 for i:=0 to 6 do
 for j:=0 to 6 do
 begin
  if (mousex > 55+75*i) and (mousex < 95+75*i) and (mousey > 55+75*j) and (mousey < 95+75*j)
  then
  begin
  Muehle[akt_muehle].genommen:=true;
  if test>akt_muehle then
  muehle[test-akt_muehle].genommen:=true;
  wegnehmen_moeglich;
  if spielfeld[i,j].muehle=false then
  begin
  if (spielfeld[i,j].Spieler=2) and (akt_spieler=1) then
  begin
    spielfeld[i,j].Bewertung:=1;
    spielfeld[i,j].Spieler:=0;
    Spieler[2].Anzahl_Figuren_innen:=Spieler[2].Anzahl_Figuren_innen-1;
    if (Spieler[2].Anzahl_Figuren_innen<=3) and (spieler[2].Anzahl_Figuren=0) then
    informationen.Lines.add('Für Schwarz beginnt nun der Sprungvorgang!');
    for k:=1 to 9 do
     if (Spieler[2].Figur[k].Pos.x=i) and (Spieler[2].Figur[k].Pos.y=j) then
      Spieler[2].Figur[k].gesetzt:=0;
    Repaint();
```

```
 akt_muehle:=0;
end;
 if (spielfeld[i,j].Spieler=1) and (akt_spieler=2) then
begin
 spielfeld[i,j].Bewertung:=1;
 spielfeld[i,j].Spieler:=0;
 Spieler[1].Anzahl_Figuren_innen:=Spieler[1].Anzahl_Figuren_innen-1;
 if (Spieler[1].Anzahl_Figuren_innen<=3) and (spieler[1].Anzahl_Figuren=0) then
 informationen.Lines.add('Für Weiß beginnt nun der Sprungvorgang!');
 for k:=1 to 9 do
  if (Spieler[1].Figur[k].Pos.x=i) and (Spieler[1].Figur[k].Pos.y=j) then
   Spieler[1].Figur[k].gesetzt:=0;
 Repaint();
 akt_muehle:=0;
end;
end else
 showmessage('Dieser Stein befindet sich in einer Mühle!');
end;
end;
NaechsterSpieler;
end
else
```

Ziehen eines Steins mit Setzbarkeitsabfrage

Unter der Bedingung, dass bereits ein Stein gewählt wurde und dass die Anzahl der InnenFiguren größer als drei ist (Zugphase), soll dieser durch „Ziehen“ eine neue Position erhalten. Dazu soll zunächst abgefragt werden, ob die angeklickte Position existiert und ob das Setzen der Richtigkeit entspricht. Ist dies der Fall, wird die alte Position der Spielfigur gelöscht und durch die neue Position aktualisiert. Außerdem wurde der Quelltext so ergänzt, dass ein Umwählen möglich ist: Sollte der gewählte Stein dem aktuellen Spieler entsprechen, so soll dieser als neu gewählter Stein beim neuen

Aufruf der Prozedur „FormClick“ (also neuem Klick auf die Form) fungieren, er ist neu gewählt, unter der Voraussetzung, dass er auch setzbar ist.

Auch hier muss wieder überprüft werden, ob eine neue Mühle vorliegt, damit erkannt werden kann, ob wieder ein Stein entfernt werden soll.

Der Quelltext dieses Teilabschnittes lautet wie folgt:

```
begin
if (spieler[akt_spieler].Anzahl_Figuren=0) and (spieler[akt_spieler].Anzahl_Figuren_Innen>3) then
begin
 if gewaehlt>0 then
 begin
 for i:=0 to 6 do
 for j:=0 to 6 do
 begin
 if (mousex > 55+75*i) and (mousex < 95+75*i) and (mousey > 55+75*j) and (mousey < 95+75*j) then
  begin
  setzen(i,j);
  if richtig_gesetzt=true then
  begin

spielfeld[spieler[akt_spieler].Figur[gewaehlt].Pos.x,spieler[akt_spieler].Figur[gewaehlt].Pos.y].Bewertung:=1;

spielfeld[spieler[akt_spieler].Figur[gewaehlt].Pos.x,spieler[akt_spieler].Figur[gewaehlt].Pos.y].Spieler:=0;
      spieler[akt_spieler].Figur[gewaehlt].Pos.x:=i;
      spieler[akt_spieler].Figur[gewaehlt].Pos.y:=j;
      spielfeld[i,j].Bewertung:=2;
      spielfeld[i,j].Spieler:=akt_spieler;
```

```
      Repaint();
      gewaehlt:=0;
      suche_muehle;
      NaechsterSpieler;
end else if (richtig_gesetzt=false) and (spielfeld[i,j].spieler=akt_spieler) then begin
  for k:=1 to 9 do
   if (spieler[akt_spieler].Figur[k].Pos.x=i) and (spieler[akt_spieler].Figur[k].pos.y=j) then
    begin
    gewaehlt:=k;
    if spieler[akt_spieler].Figur[k].setzbar=false then
    showmessage('Dieser Stein kann nicht gesetzt werden!') else
    showmessage('Figur '+inttostr(k)+' ist gewählt!');
end;
 end;
 end;
 end;
 end else
```

Sollten allerdings die für den oberen Quelltext nötigen Eingaben nicht vorhanden sein, so ist noch kein Stein zum Ziehen gewählt worden. Sollte dies der Fall sein, so muss noch ein Stein gewählt werden.

Falls nun die Bedingung, dass kein Stein gewählt ist, erfüllt ist, so wird wieder überprüft, ob die geklickte Position zu einem Kontaktpunkt gehört. Falls dies zutrifft, so soll erkannt werden, ob dieser Kontaktpunkt wirklich zu dem aktuellen Spieler gehört und sich dort dessen Figur befindet. Dann soll überprüft werden, wie viele Figuren setzbar sind (mithilfe der Prozedur „setzbar“. Sollte keine Figur setzbar sein, so ist das Spiel beendet und je nach dem aktuellen Spieler erfolgt eine Meldung, dass der Gegner aufgrund der Blockade gewonnen hat.

Ist dies nicht der Fall, so gilt die angeklickte Figur (sofern sie zum Spieler gehört) als gewählt, falls sie auch setzbar ist. Sollte die Figur nicht setzbar sein, so erscheint dementsprechend eine Meldung.

Dementsprechend folgt der anweisende Quelltext:

```
if gewaehlt=0 then
begin
for i:=0 to 6 do
for j:=0 to 6 do
 begin
 if (mousex > 55+75*i) and (mousex < 95+75*i) and (mousey > 55+75*j) and (mousey < 95+75*j) then
  begin
   if (spielfeld[i,j].Bewertung=2) and (spielfeld[i,j].spieler=akt_spieler) then
    for k:=1 to 9 do
     begin
    if (spieler[akt_spieler].Figur[k].Pos.x=i) and (spieler[akt_spieler].Figur[k].Pos.y=j) and (spielfeld[i,j].Spieler=akt_spieler)
     and (spielfeld[i,j].Bewertung=2) then
      begin
      anzahl_setzbar:=0;
      setzbar;
      if anzahl_setzbar=0 then
      begin
      if akt_spieler=1 then
      begin
      showmessage('Kein Stein ist setzbar! Spieler Schwarz hat gewonnen!');
      close;
      end else
      begin
      showmessage('Kein Stein ist setzbar! Spieler Weiß hat gewonnen!');
      close;
      end;
```

```
      end
      else
      begin
      if spieler[akt_spieler].Figur[k].setzbar then
      begin
      gewaehlt:=k;
      showmessage('Figur '+inttostr(k)+' ist gewählt!');
      anzahl_setzbar:=0;
      end else
      if anzahl_setzbar<>0 then
      begin
      showmessage('Dieser Stein kann nicht gesetzt werden!');
      gewaehlt:=0;
      end;
      end;
     end;
  end else if (spielfeld[i,j].spieler<>akt_spieler) and (spielfeld[i,j].Bewertung<>1) then
ShowMessage('Dies ist nicht der aktuelle Spieler!');
   end;
  end;
  end;
end else
```

Ziehen eines Steins ohne Setzbarkeitsabfrage

Für den Fall, dass der aktuelle Spieler weniger als drei Innenfiguren besitzt, soll die Sprungphase beginnen.

Dementsprechend wird zunächst abgefragt, ob die Anzahl der Innenfiguren des aktuellen Spielers auch geringer als 4 ist, damit gezogen werden kann.

Die Anweisung entspricht dann der Anweisung bezüglich des Ziehens mit Setzbarkeitsabfrage, allerdings ohne die jeweilige Setzbarkeitsprüfung.

Außerdem wurde auch hier der Quelltext durch das Umwählen ergänzt (s. oben, „Ziehen eines Steins mit Setzbarkeitsabfrage) allerdings ohne die unnötige Abfrage, ob er setzbar sei.

Sie gliedert sich auch in zwei Arten: Wählen des Steins und Ziehen des gewählten Steins.

Dieser Quelltext lautet wie folgt:

```
if (gewaehlt>0) and (spieler[akt_spieler].Anzahl_Figuren_Innen<=3) then
 begin
  for i:=0 to 6 do
  for j:=0 to 6 do
  begin
  if (mousex > 55+75*i) and (mousex < 95+75*i) and (mousey > 55+75*j) and (mousey < 95+75*j) then
   begin
   if spielfeld[i,j].Bewertung=1 then
    begin

spielfeld[spieler[akt_spieler].Figur[gewaehlt].Pos.x,spieler[akt_spieler].Figur[gewaehlt].Pos.y].Bewertung:=1;

spielfeld[spieler[akt_spieler].Figur[gewaehlt].Pos.x,spieler[akt_spieler].Figur[gewaehlt].Pos.y].Spieler:=0;
      spieler[akt_spieler].Figur[gewaehlt].Pos.x:=i;
      spieler[akt_spieler].Figur[gewaehlt].Pos.y:=j;
      spielfeld[i,j].Bewertung:=2;
      spielfeld[i,j].Spieler:=akt_spieler;
      Repaint();
      gewaehlt:=0;
      suche_muehle;
      NaechsterSpieler;
```

```
 end else if (spielfeld[i,j].spieler=akt_spieler) and (spielfeld[i,j].bewertung=2) then begin
 for k:=1 to 9 do
  if (spieler[akt_spieler].Figur[k].Pos.x=i) and (spieler[akt_spieler].Figur[k].pos.y=j) then
   begin
   gewaehlt:=k;
   showmessage('Figur '+inttostr(k)+' ist gewählt!');
 end;
  end;
 end;
 end;
 end else
```

```
if (gewaehlt=0) and (spieler[akt_spieler].Anzahl_Figuren_Innen<=3) and (spieler[akt_spieler].Anzahl_Figuren=0)then
for i:=0 to 6 do
for j:=0 to 6 do
 begin
 if (mousex > 55+75*i) and (mousex < 95+75*i) and (mousey > 55+75*j) and (mousey < 95+75*j) then
  begin
  if (spielfeld[i,j].Bewertung=2) and (spielfeld[i,j].spieler=akt_spieler) then
  begin
  for k:=1 to 9 do
    if spieler[akt_spieler].Figur[k].Pos.x=i then
    if spieler[akt_spieler].Figur[k].Pos.y=j then
    begin
     gewaehlt:=k;
     showmessage('Figur '+inttostr(k)+' ist gewählt!');
```

```
    end;
   end;
  end;
 end else
```

Endabfrage des Spiels

Auf jeden Zug muss auch die Abfrage geschehen, ob denn bereits ein Spieler gewonnen habe. Dazu endet die Prozedur „FormClick“ mit der Abfrage, ob denn ein Spieler bereits nur noch zwei Steine besitze. Ist dies der Fall, so wird die Form geschlossen, nach dem eine entsprechende, über den Sieg informierende Meldung geöffnet wurde.

Der entsprechende Quelltext:

```
for k:=1 to 2 do
 if spieler[k].Anzahl_Figuren=0 then
  if spieler[k].Anzahl_Figuren_Innen=2 then
   if k=1 then
    begin
    ShowMessage('Ende des Spiels - Spieler Schwarz hat gewonnen!');
    close;
    end
    else
    begin
    Showmessage('Ende des Spiels - Spieler Weiß hat gewonnen!');
    close;
    end;
    end;
```

„Procedure NaechsterSpieler”

Die Prozedur „NaechsterSpieler“ wird dann durchlaufen, wenn der nächste Spieler an der Reihe ist. Dazu erfolgt eine Ausgabe für den Spielenden, sowie die In- bzw. Dekrementierung der globalen Variable „akt_Spieler“.

Deswegen lautet der Quelltext:

```
procedure TForm1.NaechsterSpieler;
begin
if akt_muehle=0 then
begin
if (akt_spieler=2) then
 begin
  akt_spieler:=1;
  Informationen.Lines.Add('Spieler "Weiß" ist am Zug!');
  end else
  begin
  akt_spieler:=2;
  Informationen.Lines.Add('Spieler "Schwarz" ist am Zug!');
  end;
end else
akt_spieler:=akt_spieler;
  end;
```

„Procedure Suche_Muehle“

Die Prozedur „Suche_Muehle“ ist eine der kompliziertesten Strukturen im gesamten Quelltext, da jede einzelne Mühle abgefragt werden muss, da die Positionen der Mühlen tatsächlich willkürlich

erscheinen und nicht direkt berechnet werden können. Daher soll die Position der jeweiligen Mühle bildlich veranschaulicht werden, um eine Erklärung jeder einzelnen Mühlenposition zu vermeiden.

Dennoch ist es wichtig, zu sagen, dass mit jeder Mühleposition zunächst die Abfrage erfolgt, ob die drei betroffenen Felder vom Spieler besetzt sind und überhaupt besetzt sind. Ist dies der Fall, so gilt die Mühle als „vorhanden“ und der Spieler, der die Mühle besetzt, gilt als Spieler der Mühle, sonst nicht.

Diese Abfrage wird für jede einzelne Mühleposition durchlaufen, wenn die Prozedur „Suche_Muehle“ aktiviert wird.

Der Quelltext dafür ist folgender:

```
procedure tform1.suche_muehle;

var i: integer;

begin


if (spielfeld[0,0].Bewertung=2) and (spielfeld[3,0].Bewertung=2)    //Mühle 1

and (spielfeld[6,0].Bewertung=2)and

((spielfeld[0,0].Spieler)=(spielfeld[3,0].Spieler)) and

((spielfeld[0,0].Spieler)=(spielfeld[6,0].Spieler)) then

begin

Muehle[1].vorhanden:=true;

Muehle[1].Spieler:=Spielfeld[0,0].Spieler;

end else

begin

muehle[1].vorhanden:=false;

muehle[1].Spieler:=0;

muehle[1].genommen:=false;

end;


if (spielfeld[6,0].Bewertung=2) and (spielfeld[6,3].Bewertung=2)    //Mühle 2

and (spielfeld[6,6].Bewertung=2)and
```

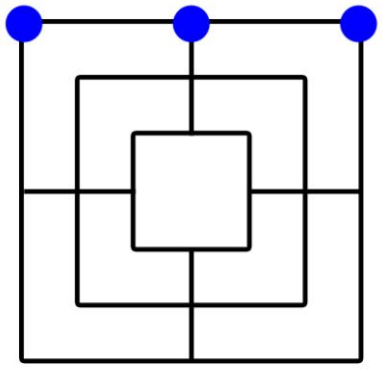


Abbildung 7 - Mühle 1

```
((spielfeld[6,0].Spieler)=(spielfeld[6,3].Spieler)) and
((spielfeld[6,0].Spieler)=(spielfeld[6,6].Spieler)) then
begin
Muehle[2].vorhanden:=true;
Muehle[2].Spieler:=Spielfeld[6,0].Spieler;
end else
begin
muehle[2].vorhanden:=false;
muehle[2].Spieler:=0;
muehle[2].genommen:=false;
end;
```

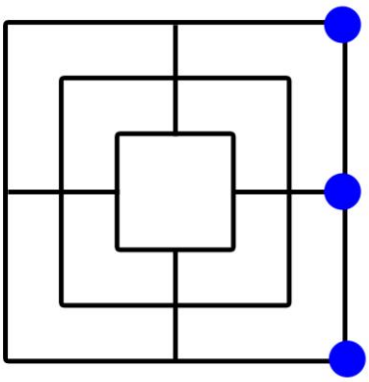


Abbildung 8 - Mühle 2

```
if (spielfeld[6,6].Bewertung=2) and (spielfeld[3,6].Bewertung=2)    //Mühle 3
and (spielfeld[0,6].Bewertung=2)and
((spielfeld[6,6].Spieler)=(spielfeld[3,6].Spieler)) and
((spielfeld[6,6].Spieler)=(spielfeld[0,6].Spieler)) then
begin
 Muehle[3].vorhanden:=true;
 Muehle[3].Spieler:=Spielfeld[6,6].Spieler;
 end else
begin
muehle[3].vorhanden:=false;
muehle[3].Spieler:=0;
muehle[3].genommen:=false;
end;
```

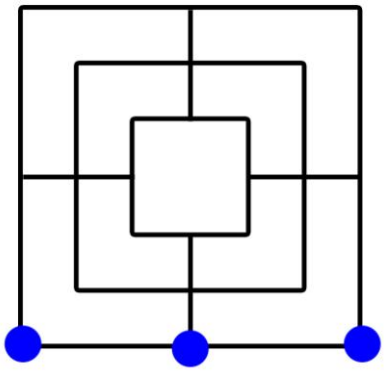


Abbildung 9 - Mühle 3

```
if (spielfeld[0,0].Bewertung=2) and (spielfeld[0,3].Bewertung=2)    //Mühle 4
```

```
and (spielfeld[0,6].Bewertung=2)and
((spielfeld[0,0].Spieler)=(spielfeld[0,3].Spieler)) and
((spielfeld[0,0].Spieler)=(spielfeld[0,6].Spieler)) then
begin
 Muehle[4].vorhanden:=true;
 Muehle[4].Spieler:=Spielfeld[0,0].Spieler;
end else
begin
muehle[4].vorhanden:=false;
muehle[4].Spieler:=0;
muehle[4].genommen:=false;
end;
```

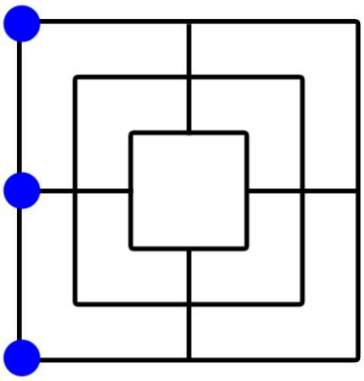


Abbildung 10 - Mühle 4

```
if (spielfeld[1,1].Bewertung=2) and (spielfeld[3,1].Bewertung=2)    //Mühle 5
and (spielfeld[5,1].Bewertung=2)and
((spielfeld[1,1].Spieler)=(spielfeld[3,1].Spieler)) and
((spielfeld[1,1].Spieler)=(spielfeld[5,1].Spieler)) then
begin
 Muehle[5].vorhanden:=true;
 Muehle[5].Spieler:=Spielfeld[1,1].Spieler;
end else
begin
muehle[5].vorhanden:=false;
muehle[5].Spieler:=0;
muehle[5].genommen:=false;
end;
```

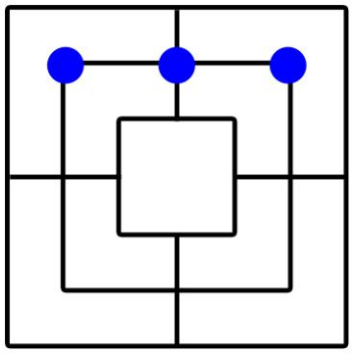


Abbildung 11 - Mühle 5

```
if (spielfeld[5,1].Bewertung=2) and (spielfeld[5,3].Bewertung=2)    //Mühle 6
```

```
and (spielfeld[5,5].Bewertung=2)and
((spielfeld[5,1].Spieler)=(spielfeld[5,3].Spieler)) and
((spielfeld[5,1].Spieler)=(spielfeld[5,5].Spieler)) then
begin
 Muehle[6].vorhanden:=true;
 Muehle[6].Spieler:=Spielfeld[5,1].Spieler;
end else
begin
muehle[6].vorhanden:=false;
muehle[6].Spieler:=0;
muehle[6].genommen:=false;
end;
```

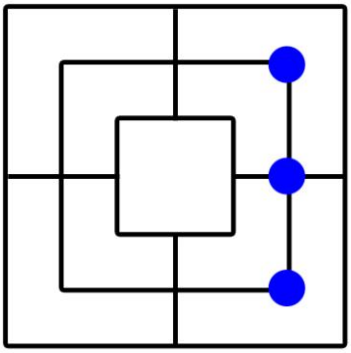


Abbildung 12 - Mühle 6

```
if (spielfeld[1,5].Bewertung=2) and (spielfeld[3,5].Bewertung=2)    //Mühle 7
and (spielfeld[5,5].Bewertung=2)and
((spielfeld[1,5].Spieler)=(spielfeld[3,5].Spieler)) and
((spielfeld[1,5].Spieler)=(spielfeld[5,5].Spieler)) then
begin
 Muehle[7].vorhanden:=true;
 Muehle[7].Spieler:=Spielfeld[1,5].Spieler;
end else
begin
muehle[7].vorhanden:=false;
muehle[7].Spieler:=0;
muehle[7].genommen:=false;
end;
```

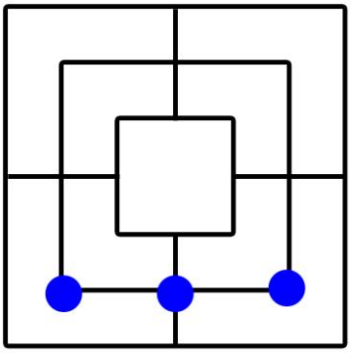


Abbildung 13 - Mühle 7

```
if (spielfeld[1,1].Bewertung=2) and (spielfeld[1,3].Bewertung=2)    //Mühle 8
```

```
and (spielfeld[1,5].Bewertung=2)and
((spielfeld[1,1].Spieler)=(spielfeld[1,3].Spieler)) and
((spielfeld[1,1].Spieler)=(spielfeld[1,5].Spieler)) then
begin
 Muehle[8].vorhanden:=true;
 Muehle[8].Spieler:=Spielfeld[1,1].Spieler;
end else
begin
muehle[8].vorhanden:=false;
muehle[8].Spieler:=0;
muehle[8].genommen:=false;
end;
```

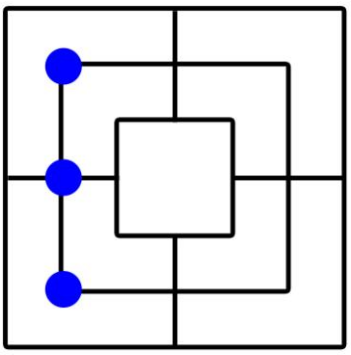


Abbildung 14 - Mühle 8

```
if (spielfeld[2,2].Bewertung=2) and (spielfeld[3,2].Bewertung=2)    //Mühle 9
and (spielfeld[4,2].Bewertung=2)and
((spielfeld[2,2].Spieler)=(spielfeld[3,2].Spieler)) and
((spielfeld[2,2].Spieler)=(spielfeld[4,2].Spieler)) then
begin
 Muehle[9].vorhanden:=true;
 Muehle[9].Spieler:=Spielfeld[2,2].Spieler;
end else
begin
muehle[9].vorhanden:=false;
muehle[9].Spieler:=0;
muehle[9].genommen:=false;
end;
```

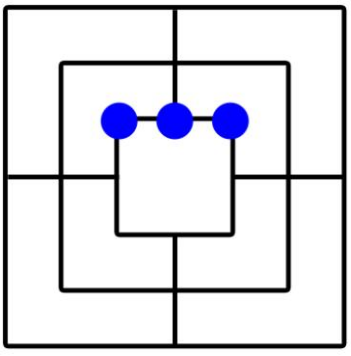


Abbildung 15 - Mühle 9

```
if (spielfeld[4,2].Bewertung=2) and (spielfeld[4,3].Bewertung=2)    //Mühle 10
and (spielfeld[4,4].Bewertung=2)and
((spielfeld[4,2].Spieler)=(spielfeld[4,3].Spieler)) and
((spielfeld[4,2].Spieler)=(spielfeld[4,4].Spieler)) then
begin
 Muehle[10].vorhanden:=true;
 Muehle[10].Spieler:=Spielfeld[4,2].Spieler;
end else
begin
muehle[10].vorhanden:=false;
muehle[10].Spieler:=0;
muehle[10].genommen:=false;
end;
```

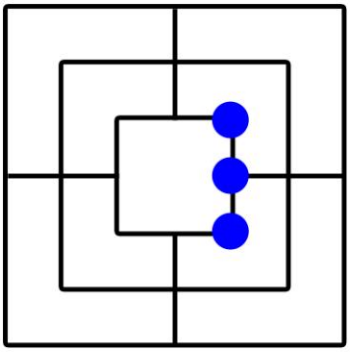


Abbildung 16 - Mühle 10

```
if (spielfeld[2,4].Bewertung=2) and (spielfeld[3,4].Bewertung=2)    //Mühle 11
and (spielfeld[4,4].Bewertung=2)and
((spielfeld[2,4].Spieler)=(spielfeld[3,4].Spieler)) and
((spielfeld[2,4].Spieler)=(spielfeld[4,4].Spieler)) then
begin
 Muehle[11].vorhanden:=true;
 Muehle[11].Spieler:=Spielfeld[2,4].Spieler;
end else
begin
muehle[11].vorhanden:=false;
muehle[11].Spieler:=0;
```

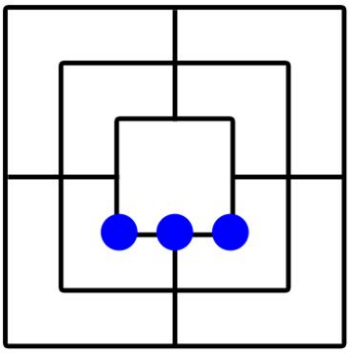


Abbildung 17 - Mühle 11

```
muehle[11].genommen:=false;
end;

if (spielfeld[2,2].Bewertung=2) and (spielfeld[2,3].Bewertung=2)    //Mühle 12
and (spielfeld[2,4].Bewertung=2)and
((spielfeld[2,2].Spieler)=(spielfeld[2,3].Spieler)) and
((spielfeld[2,2].Spieler)=(spielfeld[2,4].Spieler)) then
begin
 Muehle[12].vorhanden:=true;
 Muehle[12].Spieler:=Spielfeld[2,2].Spieler;
end else
begin
muehle[12].vorhanden:=false;
muehle[12].Spieler:=0;
muehle[12].genommen:=false;
end;
```

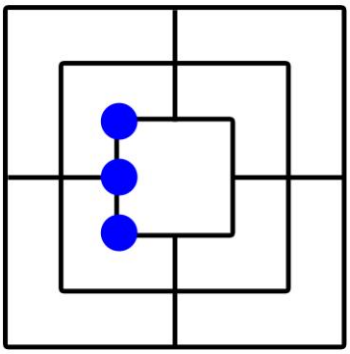


Abbildung 18 - Mühle 12

```
if (spielfeld[3,0].Bewertung=2) and (spielfeld[3,1].Bewertung=2)    //Mühle 13
and (spielfeld[3,2].Bewertung=2)and
((spielfeld[3,0].Spieler)=(spielfeld[3,1].Spieler)) and
((spielfeld[3,0].Spieler)=(spielfeld[3,2].Spieler)) then
begin
 Muehle[13].vorhanden:=true;
 Muehle[13].Spieler:=Spielfeld[3,0].Spieler;
end else
begin
muehle[13].vorhanden:=false;
```

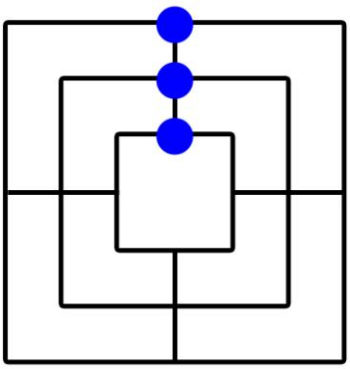


Abbildung 19 - Mühle 13

```
muehle[13].Spieler:=0;
muehle[13].genommen:=false;
end;

if (spielfeld[4,3].Bewertung=2) and (spielfeld[5,3].Bewertung=2)    //Mühle 14
and (spielfeld[6,3].Bewertung=2)and
((spielfeld[4,3].Spieler)=(spielfeld[5,3].Spieler)) and
((spielfeld[4,3].Spieler)=(spielfeld[6,3].Spieler)) then
begin
 Muehle[14].vorhanden:=true;
 Muehle[14].Spieler:=Spielfeld[4,3].Spieler;
end else
begin
muehle[14].vorhanden:=false;
muehle[14].Spieler:=0;
muehle[14].genommen:=false;
end;
```

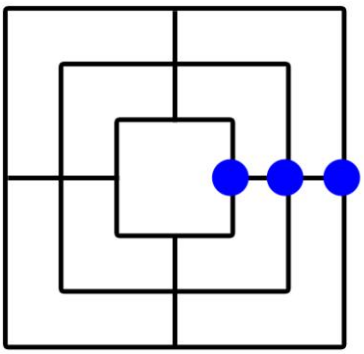


Abbildung 20 - Mühle 14

```
if (spielfeld[3,4].Bewertung=2) and (spielfeld[3,5].Bewertung=2)    //Mühle 15
and (spielfeld[3,6].Bewertung=2)and
((spielfeld[3,4].Spieler)=(spielfeld[3,5].Spieler)) and
((spielfeld[3,4].Spieler)=(spielfeld[3,6].Spieler)) then
begin
 Muehle[15].vorhanden:=true;
 Muehle[15].Spieler:=Spielfeld[3,4].Spieler;
end else
begin
```

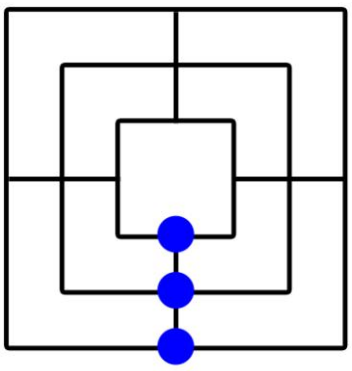


Abbildung 5 - Mühle 15

```
muehle[15].vorhanden:=false;
muehle[15].Spieler:=0;
muehle[15].genommen:=false;
end;
```

```
if (spielfeld[0,3].Bewertung=2) and (spielfeld[1,3].Bewertung=2)    //Mühle 16
and (spielfeld[2,3].Bewertung=2)and
((spielfeld[0,3].Spieler)=(spielfeld[1,3].Spieler)) and
((spielfeld[0,3].Spieler)=(spielfeld[2,3].Spieler)) then
begin
 Muehle[16].vorhanden:=true;
 Muehle[16].Spieler:=Spielfeld[0,3].Spieler;
end else
begin
muehle[16].vorhanden:=false;
muehle[16].Spieler:=0;
muehle[16].genommen:=false;
end;
```

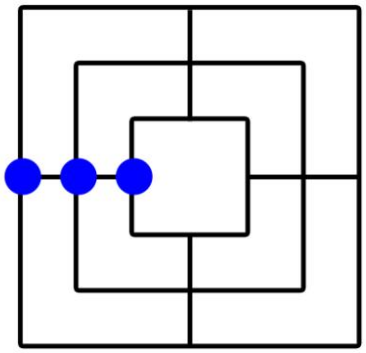


Abbildung 22 - Mühle 16

Auf diese Abfragen, welche die Mühlen überprüfen, folgt die Funktion, die feststellen soll, ob die Mühle eine aktuell gewählte ist (und damit ein Stein entfernt werden darf).

Dazu wird genau 16-mal abgefragt, ob eine vorhandene Mühle auch noch nicht als „genommen“ gilt. Ist dies der Fall, dann soll die aktuelle Mühle diejenige sein, die dem entspricht.

Sollten sich allerdings alle Steine des Gegners in Mühlen befinden, soll dementsprechend benachrichtigt werden, dass ein Stein aus einer Mühle entfernt werden darf, sonst soll dem aktuellen Spieler entsprechend benachrichtigt werden, dass ein Stein entfernt werden darf.

Ergänzend zu der aktuellen Mühle wird, falls eine „Doppelmühle“ entsteht, eine zweite Variable „test“ deklariert, welche den Wert der aktuellen Mühle und der zweiten aktuellen Mühle addiert, um diesen dann wieder in FormClick abfragen zu können.

Darum lautet der Quelltext:

```
for i:=1 to 16 do
 if (muehle[i].vorhanden) and (muehle[i].genommen=false) then
 begin
  akt_muehle:=i;
  test:=test+akt_muehle;
  if muehle[i].Spieler=1 then
   begin
    if (anzahl_muehlen=spieler[2].Anzahl_Figuren_Innen) and (anzahl_muehlen<>0) then
    showmessage('Alle Steine des Spielers Schwarz befinden sich in einer Mühle. Bitte einen Stein aus
einer Mühle entfernen!') else
    begin
    ShowMessage('Mühle '+Inttostr(i)+' wurde durch Spieler Weiß erzielt!');
    ShowMessage('Bitte einen Stein von Spieler Schwarz entfernen!');
    informationen.lines.add('Mühle '+Inttostr(i)+' wurde durch Spieler Weiß erzielt!');
    informationen.lines.add('Bitte einen Stein von Spieler Schwarz entfernen!');
    end;
   end
  else
begin
 if muehle[i].Spieler=2 then
  begin
  wegnehmen_moeglich;
   if (anzahl_muehlen=spieler[1].Anzahl_Figuren_Innen) and (anzahl_muehlen<>0) then
showmessage('Alle Steine des Spielers Weiß befinden sich in einer Mühle. Bitte einen Stein aus einer
Mühle entfernen!') else
    begin
```

```
  ShowMessage('Mühle '+Inttostr(i)+' wurde durch Spieler Schwarz erzielt!');
  ShowMessage('Bitte einen Stein von Spieler Weiß entfernen!');
  informationen.lines.add('Mühle '+Inttostr(i)+' wurde durch Spieler Schwarz erzielt!');
  informationen.lines.add('Bitte einen Stein von Spieler Weiß entfernen!');
  end;
  end;
  end;
 end;
end;
```

„Procedure Wegnehmen_Moeglich“

Zusätzlich musste noch eine Prozedur erstellt werden, die nah an die Prozedur „Suche_Muehle“ gelegt ist. Diese Prozedur lautet „Wegnehmen_moeglich“ und überprüft, welche Steine sich in welcher Mühle befinden, damit ein Stein, der sich in einer Mühle befindet, „geschützt“ ist und nicht entnommen werden darf.

Dazu wird abgefragt, welche Spielfeldpositionen sich in Mühlen befinden. Diese Positionen sollen dann in der Prozedur „FormClick“ abgefragt werden, um festzustellen ob der jeweilige Stein entfernt werden darf.

Gleichzeitig zählt die Prozedur „Wegnehmen_Moeglich“ die Steine, die sich in einer Mühle befinden, um später zu erkennen, ob sich alle Steine in einer Mühle befinden oder nicht (Wegnehmen aus Mühle).

Der entsprechende Quelltext lautet:

```
procedure TForm1.wegnehmen_moeglich;
var i,m,n: integer;
begin
anzahl_muehlen:=0;
for m:=0 to 6 do
for n:=0 to 6 do
spielfeld[m,n].Muehle:=false;
```

```
for i:=1 to 16 do
 if muehle[i].vorhanden=true then
  begin
   case i of
    1: begin
       spielfeld[0,0].muehle:=true;
       spielfeld[3,0].muehle:=true;
       spielfeld[6,0].muehle:=true;
      end;
    2: begin
       spielfeld[6,0].muehle:=true;
       spielfeld[6,3].muehle:=true;
       spielfeld[6,6].Muehle:=true;
      end;
    3: begin
       spielfeld[6,6].muehle:=true;
       spielfeld[3,6].muehle:=true;
       spielfeld[0,6].muehle:=true;
      end;
    4: begin
       spielfeld[0,0].muehle:=true;
       spielfeld[0,3].muehle:=true;
       spielfeld[6,6].Muehle:=true;
      end;
    5: begin
       spielfeld[1,1].muehle:=true;
       spielfeld[3,1].muehle:=true;
       spielfeld[5,1].muehle:=true;
```

```
  end;
6: begin
   spielfeld[5,1].muehle:=true;
   spielfeld[5,3].muehle:=true;
   spielfeld[5,5].Muehle:=true;
  end;
7: begin
   spielfeld[5,5].muehle:=true;
   spielfeld[3,5].muehle:=true;
   spielfeld[1,5].muehle:=true;
  end;
8: begin
   spielfeld[1,1].muehle:=true;
   spielfeld[1,3].muehle:=true;
   spielfeld[1,5].Muehle:=true;
  end;
9: begin
   spielfeld[2,2].muehle:=true;
   spielfeld[3,2].muehle:=true;
   spielfeld[4,2].muehle:=true;
  end;
10: begin
   spielfeld[4,2].muehle:=true;
   spielfeld[4,3].muehle:=true;
   spielfeld[4,4].Muehle:=true;
  end;
11: begin
   spielfeld[4,4].muehle:=true;
```

```
    spielfeld[3,4].muehle:=true;
    spielfeld[2,4].muehle:=true;
   end;
12: begin
    spielfeld[2,4].muehle:=true;
    spielfeld[2,3].muehle:=true;
    spielfeld[2,2].Muehle:=true;
    end;
13: begin
    spielfeld[3,0].muehle:=true;
    spielfeld[3,1].muehle:=true;
    spielfeld[3,2].muehle:=true;
   end;
14: begin
    spielfeld[6,3].muehle:=true;
    spielfeld[5,3].muehle:=true;
    spielfeld[4,3].Muehle:=true;
   end;
15: begin
    spielfeld[3,6].muehle:=true;
    spielfeld[3,5].muehle:=true;
    spielfeld[3,4].Muehle:=true;
   end;
16: begin
    spielfeld[0,3].muehle:=true;
    spielfeld[1,3].muehle:=true;
    spielfeld[2,3].Muehle:=true;
   end;
```

```
      end;
 end;


for m:=0 to 6 do
for n:=0 to 6 do
begin
if akt_spieler=1 then
 if (spielfeld[m,n].Spieler=2) and (spielfeld[m,n].Muehle=true) then
  anzahl_muehlen:=anzahl_muehlen+1;


if akt_spieler=2 then
 if (spielfeld[m,n].Spieler=1) and (spielfeld[m,n].muehle=true) then
  anzahl_muehlen:=anzahl_muehlen+1;
end;
end;
end.
```

„Procedure Setzbar“

Das Problem an der Prozedur „Setzbar“ stellte ein gravierendes Problem in der Umsetzung dar. Dies ist mitunter die wichtigste Funktion, da sie den Spielern Richtlinien zum Setzen vorgibt.

Die Aufteilung des Spielfelds in 49 theoretisch mögliche Kontaktpunkte erwies sich deshalb auch als große Schwierigkeit, da sich nicht jeder praktisch anwendbare Kontaktpunkt unmittelbar neben einem anderen befindet (z.B. im äußeren Quadrat liegen zwischen den nutzbaren Kontaktpunkten noch zwei weitere). Dennoch war es möglich, zu überprüfen, ob ein Stein setzbar ist, indem die Sache gut strukturiert wurde.

Es gibt drei Bewegungsrichtungen beim Gesellschaftsspiel Mühle: oben (y+1), unten (y-1), rechts (x+1) und links (x-1). Für jede Bewegungsrichtung lassen sich die Setzbarkeitsabfragen parallel dazu durchführen (nur einzelne Werte müssen verändert werden).

So muss zunächst geprüft werden, ob der Kontaktpunkt unmittelbar neben der jeweiligen Figur besetzt oder nicht vorhanden ist. Ist er besetzt, so ist die Figur hinsichtlich dieses Kontaktpunktes

nicht setzbar. Ist er nicht vorhanden, so muss der Kontaktpunkt unmittelbar neben diesem nicht vorhandenen Kontaktpunkt auf die gleiche Art überprüft werden (usw.).

Eine Ausnahme bilden allerdings die Mittellinien des Spielfelds. Hier sind die Steine nur setzbar, wenn unmittelbar neben einem Stein ein freier Kontaktpunkt ist (die Abfrage für „x+3“ o.ä. bleibt aus, da sonst das innere Quadrat übersprungen würde).

Erfolgt diese Art von Abfrage für jede Bewegungsrichtung, so kann erkannt werden, ob dieser Stein bewegt werden darf oder nicht.

Der Quelltext lautet:

```
procedure TForm1.setzbar;
var i,j,m:integer;
begin
for i:=1 to 2 do
for j:=1 to 9 do
begin
spieler[i].Figur[j].setzbar:=false;
if spielfeld[spieler[i].Figur[j].Pos.x+1,spieler[i].Figur[j].Pos.y].Bewertung=1 then
begin
spieler[i].Figur[j].setzbar:=true;
end;
if spieler[i].Figur[j].Pos.y<>3 then
if spielfeld[spieler[i].Figur[j].Pos.x+1,spieler[i].Figur[j].Pos.y].Bewertung=0 then
 if spielfeld[spieler[i].Figur[j].Pos.x+2,spieler[i].Figur[j].Pos.y].Bewertung=1 then
 begin
spieler[i].Figur[j].setzbar:=true;
end
 else if spielfeld[spieler[i].Figur[j].Pos.x+2,spieler[i].Figur[j].Pos.y].Bewertung=0 then
  if spielfeld[spieler[i].Figur[j].Pos.x+3,spieler[i].Figur[j].Pos.y].Bewertung=1 then
  begin
```

```
spieler[i].Figur[j].setzbar:=true;
end;
 if spielfeld[spieler[i].Figur[j].Pos.x-1,spieler[i].Figur[j].Pos.y].Bewertung=1 then
begin
spieler[i].Figur[j].setzbar:=true;
end;
if spieler[i].Figur[j].Pos.y<>3 then
if spielfeld[spieler[i].Figur[j].Pos.x-1,spieler[i].Figur[j].Pos.y].Bewertung=0 then
 if spielfeld[spieler[i].Figur[j].Pos.x-2,spieler[i].Figur[j].Pos.y].Bewertung=1 then
begin
spieler[i].Figur[j].setzbar:=true;
end
 else if spielfeld[spieler[i].Figur[j].Pos.x-2,spieler[i].Figur[j].Pos.y].Bewertung=0 then
if spielfeld[spieler[i].Figur[j].Pos.x-3,spieler[i].Figur[j].Pos.y].Bewertung=1 then
beginspieler[i].Figur[j].setzbar:=true;
end;
 if spielfeld[spieler[i].Figur[j].Pos.x,spieler[i].Figur[j].Pos.y-1].Bewertung=1 then
begin
spieler[i].Figur[j].setzbar:=true;
end;
if spieler[i].Figur[j].Pos.x<>3 then
if spielfeld[spieler[i].Figur[j].Pos.x,spieler[i].Figur[j].Pos.y-1].Bewertung=0 then
 if spielfeld[spieler[i].Figur[j].Pos.x,spieler[i].Figur[j].Pos.y-2].Bewertung=1 then
begin
spieler[i].Figur[j].setzbar:=true;
end
 else if spielfeld[spieler[i].Figur[j].Pos.x,spieler[i].Figur[j].Pos.y-2].Bewertung=0 then
 if spielfeld[spieler[i].Figur[j].Pos.x,spieler[i].Figur[j].Pos.y-3].Bewertung=1 then
```

```
begin
spieler[i].Figur[j].setzbar:=true;
end;

 if spielfeld[spieler[i].Figur[j].Pos.x,spieler[i].Figur[j].Pos.y+1].Bewertung=1 then
begin
spieler[i].Figur[j].setzbar:=true;
end;
if spieler[i].Figur[j].Pos.x<>3 then
if spielfeld[spieler[i].Figur[j].Pos.x,spieler[i].Figur[j].Pos.y+1].Bewertung=0 then
 if spielfeld[spieler[i].Figur[j].Pos.x,spieler[i].Figur[j].Pos.y+2].Bewertung=1 then
begin
spieler[i].Figur[j].setzbar:=true;
end
 else if spielfeld[spieler[i].Figur[j].Pos.x,spieler[i].Figur[j].Pos.y+2].Bewertung=0 then
  if spielfeld[spieler[i].Figur[j].Pos.x,spieler[i].Figur[j].Pos.y+3].Bewertung=1 then
begin
spieler[i].Figur[j].setzbar:=true;
end;

end;
for m:=1 to 9 do
if spieler[akt_spieler].Figur[m].setzbar=true and (spieler[akt_spieler].Figur[m].gesetzt>0) then
anzahl_setzbar:=anzahl_setzbar+1;
end;
```

„Procedure Setzen“

Die Prozedur „Setzen“ liegt nah an der Prozedur „Setzbar“. Sie muss aber unabhängig von der Prozedur „Setzbar“ erfolgen, damit die neue Position überprüft werden kann.

Bei der Prozedur „Setzen“ erfolgt das gleiche Schema wie bei der Prozedur „Setzbar“. Allerdings wird nun nicht die gewählte Position des jeweiligen Spielsteines, sondern die neue Position überprüft. Hierbei wird ermittelt, ob die neue Position auch wirklich nur einen Kontaktpunkt von der alten entfernt ist.

Auch hier werden wieder alle möglichen Richtungen überprüft, um der Variable „richtig_gesetzt“ einen Wert zuzuweisen.

Darum lautet der Quelltext:

```
procedure TForm1.setzen (i,j:integer);

begin

richtig_gesetzt:=false;


if (spieler[akt_spieler].Figur[gewaehlt].Pos.x+1=i) and (spieler[akt_spieler].Figur[gewaehlt].Pos.y=j)

and (spielfeld[i,j].Bewertung=1) then

richtig_gesetzt:=true;


if
(spielfeld[spieler[akt_spieler].Figur[gewaehlt].Pos.x+1,spieler[akt_spieler].Figur[gewaehlt].Pos.y].Be
wertung=0) then

begin

if  (spieler[akt_spieler].Figur[gewaehlt].Pos.x+2=i) and  (spieler[akt_spieler].Figur[gewaehlt].Pos.y=j)

and (spielfeld[i,j].Bewertung=1) then

richtig_gesetzt:=true;

if  (spieler[akt_spieler].Figur[gewaehlt].Pos.x+3=i) and  (spieler[akt_spieler].Figur[gewaehlt].Pos.y=j)

and (spielfeld[i,j].Bewertung=1) then

richtig_gesetzt:=true;

end;


if (spieler[akt_spieler].Figur[gewaehlt].Pos.x-1=i) and (spieler[akt_spieler].Figur[gewaehlt].Pos.y=j)

and (spielfeld[i,j].Bewertung=1) then
```

```
richtig_gesetzt:=true;

if (spielfeld[spieler[akt_spieler].Figur[gewaehlt].Pos.x-1,spieler[akt_spieler].Figur[gewaehlt].Pos.y].Bewertung=0) then
begin
if  (spieler[akt_spieler].Figur[gewaehlt].Pos.x-2=i) and  (spieler[akt_spieler].Figur[gewaehlt].Pos.y=j)
and (spielfeld[i,j].Bewertung=1) then
richtig_gesetzt:=true;
if  (spieler[akt_spieler].Figur[gewaehlt].Pos.x-3=i) and  (spieler[akt_spieler].Figur[gewaehlt].Pos.y=j)
and (spielfeld[i,j].Bewertung=1) then
richtig_gesetzt:=true;
end;

if (spieler[akt_spieler].Figur[gewaehlt].Pos.y-1=j) and (spieler[akt_spieler].Figur[gewaehlt].Pos.x=i)
and (spielfeld[i,j].Bewertung=1) then
richtig_gesetzt:=true;

if (spielfeld[spieler[akt_spieler].Figur[gewaehlt].Pos.y-1,spieler[akt_spieler].Figur[gewaehlt].Pos.x].Bewertung=0) then
begin
if  (spieler[akt_spieler].Figur[gewaehlt].Pos.y-2=j) and  (spieler[akt_spieler].Figur[gewaehlt].Pos.x=i)
and (spielfeld[i,j].Bewertung=1) then
richtig_gesetzt:=true;
if  (spieler[akt_spieler].Figur[gewaehlt].Pos.y-3=j) and  (spieler[akt_spieler].Figur[gewaehlt].Pos.x=i)
and (spielfeld[i,j].Bewertung=1) then
richtig_gesetzt:=true;
end;
```

```
if (spieler[akt_spieler].Figur[gewaehlt].Pos.y+1=j) and (spieler[akt_spieler].Figur[gewaehlt].Pos.x=i)
and (spielfeld[i,j].Bewertung=1) then
richtig_gesetzt:=true;

if
(spielfeld[spieler[akt_spieler].Figur[gewaehlt].Pos.y+1,spieler[akt_spieler].Figur[gewaehlt].Pos.x].Bewertung=0) then
begin
if  (spieler[akt_spieler].Figur[gewaehlt].Pos.y+2=j) and  (spieler[akt_spieler].Figur[gewaehlt].Pos.x=i)
and (spielfeld[i,j].Bewertung=1) then
richtig_gesetzt:=true;
if  (spieler[akt_spieler].Figur[gewaehlt].Pos.y+3=j) and  (spieler[akt_spieler].Figur[gewaehlt].Pos.x=i)
and (spielfeld[i,j].Bewertung=1) then
richtig_gesetzt:=true;
end;
end;
```

Zusammenfassung

Mithilfe von „Borland Delphi 7“ konnte eine annehmbare Implementierung des Gesellschaftsspiels „Mühle erzeugt werden. Dieses Spiel soll keine kommerzielle Aufarbeitung eines Brettspiel darstellen, sondern lediglich die Kenntnisse, welche durch den Informatikunterricht vermittelt wurden, zusammenfassend darstellen.

Im Ausblick auf die Verbesserung der Stabilität und Funktionalität des Programmes werde ich weiterhin versuchen, einen Computergegner zu erschaffen, der die verschiedenen Spielsituationen erlernen und beherrschen kann. Zudem wird eine Spielanalyse erfolgen, die verschiedene Spielstellungen als „unentschieden“ bezeichnen kann, um die Spieldauer nicht weiter hinauszuzögern.

Außerdem könnte die Grundidee dieses Quelltextes durch eine Umsetzung mit OpenGl vervollständigt werden, damit das Gesellschaftsspiel einen moderneren und (vor allem für die kleinen Spieler) interessanteren Charakter erhält.

Literaturverzeichnis

Gasser, Ralph: *Applying Retrograde Analysis to Nine Men's Morris*, in: Artificial Intelligence, 1990, S. 69 ff.

Wollenik, Franz: *Überlegungen zum Mühlespiel*, in: Almogaren, 1990, S. 89 ff

http://de.wikipedia.org/wiki/M%C3%BChle_%28Spiel%29, 11. März 2007

http://www.mathematische-basteleien.de/muehle.htm, 11. März 2007

http://www.tinohempel.de/info/info/facharbeit/fa_inhalt.htm, 11. März 2007

Erklärung des Verfassers

Hiermit bestätige ich, dass diese Facharbeit lediglich unter der Nutzung der im Literaturverzeichnis angegebenen Hilfsmittel angefertigt wurde und alle Formulierungen, die wörtlich oder dem Sinn nach aus anderen Quellen entnommen wurden, kenntlich gemacht wurden.

Stefan Feldmann, 11. März 2007