J2 LAMA

Bei dieser Aufgabe geht es nicht darum, sich einen komplizierten Algorithmus auszudenken, sondern die richtigen Werkzeuge zu wählen. Daher soll hier ein kurzer und einfacher Weg vorgestellt werden, ein funktionierendes und leicht erweiterbares Programm zu erhalten.

Zielsetzung und Plan

Warum das Rad neu erfinden, wenn alle benötigten Werkzeuge schon zur Verfügung stehen?

Diese Lösung soll insbesondere Folgendes demonstrieren:

- Die in eigentlich allen üblichen Programmiersprachen vorhandenen Standard-Elemente für grafische Benutzungsschnittstellen (engl.: graphical user interfaces, kurz GUI) können schon alles, was man für diese Aufgabe braucht.
- Ein gut strukturiertes Programm macht es einfach, Regeln für weitere Spielvarianten hinzuzufügen.

Die Lösung in dieser Aufgabe wird in C# 6.0 mit Windows Forms umgesetzt. Das Ganze funktioniert auch in jeder anderen Sprache mit entsprechenden GUI-Elementen. Natürlich lässt sich ein LAMA auch als Konsolenprogramm realisieren, bei denen etwa die Tastenzustände mit Textzeichen dargestellt werden und die zu drückende Taste per Angabe der Tastenposition im LED-Raster ausgewählt wird. Aber netter ist eine grafische Schnittstelle schon.

Um die Tasten bzw. Buttons eines LAMA darzustellen, nehmen wir - Überraschung! - Buttons. Diese bieten alles, was wir brauchen:

- Sie können ihre Farbe ändern (Licht an/aus).
- Es lässt sich im Programm leicht darauf reagieren, wenn ein Button gedrückt wird (LEDs umschalten).
- Sie können zur Laufzeit des Programms erzeugt werden (verschiedene Feldgrößen).

Umsetzung

Abbildung 1 zeigt ein Programmfenster für ein LAMA mit 16×16 Tasten. Oben links sieht man die Eingabemöglichkeiten für Feldgröße und Spielmodus. Mit diesen ist keine weitere Programm-Logik verbunden, und ihre Realisierung hängt nur von der gewählten UI-Bibliothek ab.

Für die Knöpfe brauchen wir zwei Farben für An und Aus:

```
Color OnColor = Color.Yellow;
Color OffColor = Color.Black;
```

Um später auf die Buttons einfach zugreifen zu können, speichern wir sie in einem 2D-Array:

```
Button[,] grid;
```

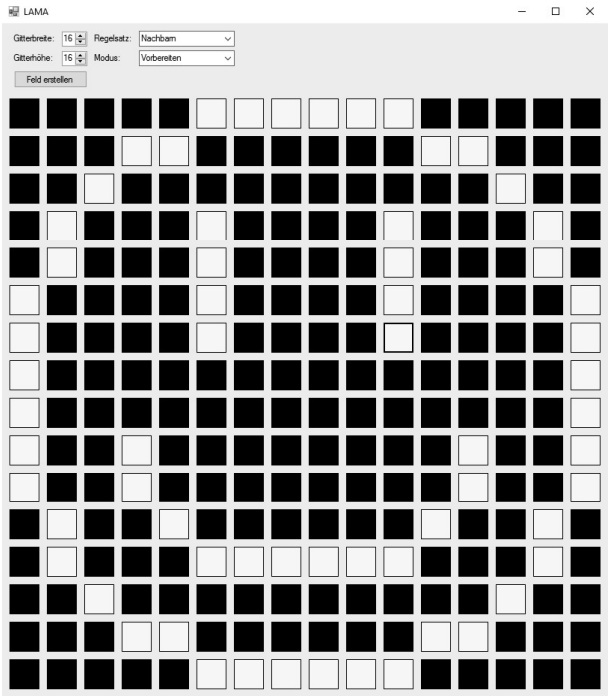

Abbildung 1: Fertiges Fenster mit einem 16×16-LAMA

Des weiteren müssen wir noch unterscheiden, ob das Spielfeld gerade vorbereitet oder ob schon gespielt wird. Wenn gespielt wird, welche Regeln werden dann genutzt? Die Auswahlmöglichkeiten realisieren wir als Aufzählungstypen (Enums), die wir als Datenquelle für die Auswahl des Regelsatzes im UI und auch später im Programm verwenden.

```
enum RuleSet { Nachbarn, ObenLinks }
enum Mode { Vorbereiten, Spielen }
```

Damit haben wir jetzt alles, um die Buttons zu erzeugen:

```
private void createGrid_Click(object sender, EventArgs e) {
  //Alte Buttons löschen
  Controls.OfType<Button>().Where(a => a !=
     sender).ToList().ForEach(Controls.Remove);
  sizeX = (int)sizeXSelect.Value; //Numeric UpDown für Größe in x-Richtung
  sizeY = (int)sizeYSelect.Value; //Numeric UpDown für Größe in y-Richtung
  grid = new Button[sizeX, sizeY];
  //Für jede Position im Gitter
  for(int x = 0; x < sizeX; x++) {
    for(int y = 0; y < sizeY; y++) {
      //Neuen Button erzeugen und konfigurieren
      Button b = new Button();
      b.Tag = new int[] { x, y }; //Tag = Position im Gitter
      b.Location = new Point(10 + x * 50, 100 + y * 50);
      b.Size = new Size(40, 40);
      b.Click += ButtonClicked;
      b.BackColor = OffColor;
      b.FlatStyle = FlatStyle.Flat;
      Controls.Add(b);
      grid[x, y] = b;
    }
  }
  //Fenstergröße anpassen
```

```
  Size = new Size(Math.Max(sizeX * 50 + 25, 339), sizeY * 50 + 138);
}
```

Jetzt müssen wir noch darauf reagieren können, wenn auf eine Taste (im Programm also auf einen Button) gedrückt wird. Dazu haben alle Buttons zu dem Ereignis Button.Click den Event-Handler ButtonClicked zugewiesen bekommen. Um feststellen zu können, welcher Knopf denn nun gedrückt worden ist, nutzen wir den Tag des Buttons: Dort haben wir beim Erstellen die Position im Gitter gespeichert.

Wenn das Spiel später gespielt wird, muss oft der Zustand einer Taste umgekehrt werden. Dies lagern wir in eine kleine Hilfsfunktion aus, die den richtigen Button ebenfalls über die Koordinaten x und y seiner Gitterposition identifiziert.

```
private void toggle(int x, int y) =>
    grid[x, y].BackColor = grid[x, y].BackColor == OffColor ? OnColor :
       OffColor;
```

Damit lässt sich jetzt der EventHandler implementieren:

```
private void ButtonClicked(object sender, EventArgs e) {
  int buttonX = ((int[])((Button)sender).Tag)[0];
  int buttonY = ((int[])((Button)sender).Tag)[1];
  toggle(btnX, btnY);
  if((Mode)mode.SelectedIndex == Mode.Vorbereiten) return;
  switch((RuleSet)ruleset.SelectedIndex) {
    case RuleSet.Nachbarn:
      //Die Klassenvariable "offsets" ist so definiert:
      //int[,] offsets = { { -1, 0 }, { 0, 1 }, { 1, 0 }, { 0, -1 } };
      for(int direction = 0; direction < 4; direction++) {
        int x = offsets[direction, 0] + buttonX;
        int y = offsets[direction, 1] + buttonY;
        if(0 <= x && x < sizeX && 0 <= y && y < sizeY)
          toggle(x, y);
      }
      break;
    case RuleSet.ObenLinks:
      for(int x = 0; x <= buttonX; x++)
        for(int y = 0; y <= buttonY; y++)
          if(x != buttonX || y != buttonY)
            toggle(x, y);
      break;
  }
  //Spiel gewonnen, wenn keine Lampe an.
  if(grid.Cast<Button>().Where(b => b.BackColor == OnColor).Count() == 0)
    MessageBox.Show("Gewonnen!");
}
```

Hier sieht man, wie einfach es ist, eine Spielvariante hinzuzufügen: Im Switch-Statement implementiert man die gewünschte Reaktion auf das Drücken einer Taste als weiteren Fall.

Am Ende der Funktion wird geprüft, ob das Spiel gewonnen wurde, nämlich genau dann, wenn kein Licht mehr leuchtet.

Bewertungskriterien

- Die Realisierung des LAMA-Spiels, egal ob grafisch oder in der Konsole, sollte die Anforderungen der Aufgabenstellung erfüllen:
  - Ein Anfangszustand soll festgelegt werden. Das kann sehr unterschiedlich realisiert sein. Entscheidend ist, dass die Zahl der möglichen Anfangszustände zumindest potenziell nicht zu klein ist. Eine zufällige Wahl des Anfangszustands ist in Ordnung, solange der (zugegeben unwahrscheinliche) Fall vermieden wird, dass alle Leuchten aus sind — eine solche Spielrunde wäre doch allzu schnell zu Ende. Ein einziger, im Programm fest codierter Zustand ist zu wenig. Das Einlesen des Anfangszustandes aus einer Datei ist in Ordnung, auch wenn nur sehr wenige Dateien als Beispiele beiliegen. Auch eine Eingabe durch den Benutzer ist eine akzeptable Variante.
  - Umschalten von Feldern durch Klicken ist gut, durch Eingabe von Koordinaten in der Konsole akzeptabel. Einlesen der Züge aus einer Text-Datei reicht nicht.
  - Der Zustand der LEDs muss erkennbar angezeigt werden: Eine grafische Darstellung ist nett, eine gute ASCII-Darstellung in der Konsole genügt aber auch.
- Das Spiel sollte, einschließlich der Regeln der jeweiligen Spielvariante, korrekt realisiert sein. Das Umschalten der Leuchten muss regelgerecht funktionieren, und das Ende des Spiels muss erkannt werden.
- Für Teilaufgabe 2 muss die Spielfeldgröße (in Grenzen) flexibel wählbar sein.
- Die in Teilaufgabe 3 beschriebene Spielvariante muss realisiert sein.
- Anhand von Beispielen sollte erkennbar werden, dass die Wahl der gedrückten Taste und insbesondere das Umschalten funktioniert. Es reicht aus, pro Spielvariante einige wenige Spielschritte darzustellen. Eine rein textuelle Darstellung der Spielschritte genügt.