Prinzipieller Ablauf des Spiels – erster Entwurf

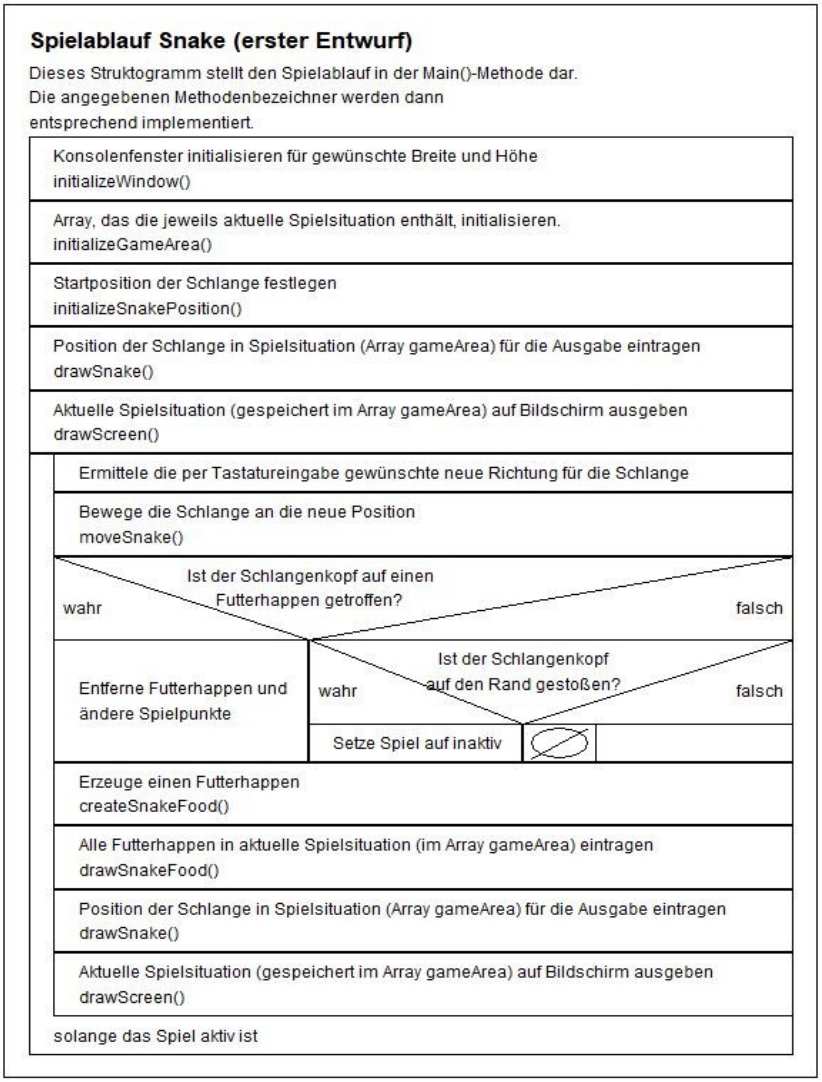
Im Folgenden sollte man sich über den prinzipiellen Ablauf des Spieles klar werden. Alle bereits erstellten Methoden und auch diejenigen, die noch zu erstellen sind, werden im Rahmen des Spielablaufes, der in der **`Main()`**-Methode implementiert wird, aufgerufen.

Man muss sich also grob darüber klar werden in welchen Einzelschritten solch ein Spiel abläuft.

Da gibt es sicherlich mehrere Möglichkeiten hinsichtlich der einzelnen Schritte und deren Reihenfolge. In dem dargestellten Struktogramm ist eine dieser Möglichkeiten realisiert.

Auch werden die bereits implementierten Methoden noch zu erweitern sein.

Aber für einen ersten groben Überblick mag das Struktogramm ganz nützlich sein.



Realisierung der Schlange

Die Schlange besteht aus einem Schlangenkopf und einer Vielzahl von Körperteilen. Zu Beginn des Spiels hat die Schlange bereits eine vorgegebene Länge.

Jedes Teil der Schlange hat eine Position im Spielfeld, die durch die jeweilige Kombination aus Zeile und Spalte bestimmt ist.

Das bedeutet also, dass für den Schlangenkopf und für jedes Körperteil dessen Position zu speichern ist.

Da sich die Schlange im Spielfeld bewegt, ändern sich entsprechend die Positionen dieser Schlangenelemente. Abgesehen davon, ändert sich auch die Anzahl der Körperteile der Schlange. Jedes Mal, wenn sie einen Futterhappen frisst, wird ein Körperteil am Ende der Schlange hinzugefügt.

Zeile	Spalte
30	40
30	41
30	42
30	43
0	0
0	0
0	0
0	0
0	0
0	0

Wie speichert man nun die Positionen (Zeile/Spalte) der einzelnen Schlangenelemente (Kopf und Körper)? Sicherlich kann man sich verschiedene Möglichkeiten der Speicherung überlegen. Die hier verwendete Möglichkeit ist eine Tabelle mit zwei Spalten für die Position des betreffenden Schlangenelementes (Zeile und Spalte) und einer ausreichenden Anzahl an Zeilen, um die wachsende Schlange zu realisieren. Da die Positionswerte immer größer Null sind, bedeutet dies, dass alle Zeilen mit den Wertepaaren 0/0 kein Schlangenelement sind. Zu Beginn des Spiels wird die Schlangenposition vorgegeben. Somit wird die Tabelle mit Startwerten versehen. Im Beispiel rechts befindet sich der Schlangenkopf zu Beginn des Spiels an der Position Zeile = 30 und Spal-

te = 40. Das erste Körperteil befindet sich an der Position Zeile = 30 und Spalte = 41. Das zweite Körperteil bei 30/42 und das dritte Körperteil bei 30/43. Die Gesamtlänge der Schlange ist also 4 Körperelemente. Die verbleibenden sechs Tabelleneinträge haben die Werte 0/0. Somit kann die Schlange maximal aus 10 Körperelementen bestehen. Der erste Eintrag bezeichnet immer den Schlangenkopf.

Wir werden noch sehen, dass sich mit dieser zweispaltigen Tabelle dann auch die Bewegung der Schlange relativ leicht realisieren lässt!

Eine Tabelle lässt sich in der Programmierung durch ein zweidimensionales Array abbilden. Hierbei gibt man die gewünschte Anzahl der Zeilen und Spalten an. Die Information, die in solch einer Tabellenzelle gespeichert werden soll, ist eine Ganzzahl, der Datentyp hierzu ist ein Integer.

Die Anzahl der Zeilen, die ja die maximale Länge der Schlange bestimmt, wird durch eine Konstante namens **`maxLengthOfSnake`** festgelegt, so dass diese an einer zentralen Stelle im Programmcode verändert werden kann.

```
// Bei den bereits existierenden Konstanten außerhalb
// der Main()-Methode zu definieren
// Maximale Länge der Schlange
const int maxLengthOfSnake = 250;
// Zeichen für Schlangenkörper
const char snakeBodyCharacter = 'S';
// ============================================
// Variablen zu Beginn und innerhalb der Main()
// Aktuelle Gesamtlänge der Schlange
int lengthOfSnake = 4;
// Enthält die Positionen des Schlangenkopfes
// und Schlangenkörpers als Zeilen/Spalten Wertepaare
// Die Werte müssen größer 0 sein
int[,] positionsOfSnake = new int[maxLengthOfSnake, 2];
```

Das Zeichen für die Schlange, soll wie auch schon beim Spielfeldrand, durch eine Character-Konstante definiert werden. Gewählt wurde der Bezeichner **`snakeBodyCharacter`** mit dem Wert **`'S'`**. Dieses Zeichen wird dann verwendet, wenn die Schlange in die aktuelle Spielsituation im Array **`gameArea`** eingetragen wird.

Beide Konstanten sind außerhalb von **`Main()`**, dort wo bereits die schon existierenden Konstanten erzeugt wurden, anzulegen.

Veränderliche Informationen zur Schlange werden in Variablen innerhalb von **`Main()`** gespeichert. Dazu gehören die aktuelle Position der Schlange und die momentane Länge der Schlange.

Die momentane Länge der Schlange darf natürlich nie den Wert für die maximale Länge überschreiten. Das wird später wichtig, wenn die Schlange durch das Fressen von Futterhappen länger wird. Die momentane Länge der Schlange wird in der Variablen **`lengthOfSnake`** gespeichert. Die Länge sind immer ganzzahlige Werte, daher wird der Datentyp **`int`** verwendet.

Die Datenstruktur zur Speicherung der Schlangenposition wurde ja schon besprochen. Diese ist das zweidimensonale Array mit dem Bezeichner **`positionsOfSnake`**.

Startposition für die Schlange festlegen

Vor Beginn des Spiels muss die Startposition der Schlange festgelegt werden. Hierzu müssen die Werte in das Array **`positionsOfSnake`** eingetragen werden. Diese Aufgabe soll eine Methode mit dem Namen **`initializeSnakePosition()`** übernehmen.

Wie muss der Methodenkopf gestaltet werden? Wie schon bei den anderen Initialisierungsmethoden, ist nicht wirklich ein Rückgabewert notwendig. Daher kann der Rückgabetyp **`void`** lauten.

Aber welche Informationen benötigt diese Methode, um ihre Arbeit zu erledigen?

Sie benötigt zumindest das Array **`positionsOfSnake`**, in dem ja die Position der Schlange gespeichert ist. Und sie benötigt die aktuelle Länge der Schlange, mittels **`lengthOfSnake`**.

Neben im Kasten ist die Methode (unvollständig) implementiert. Natürlich kann man den Kopf und den Körper der Schlange mit festen Werten in dem Array **`positionsOfSnake`** eintragen.

```
static void initializeSnakePosition(int[,] positionsOfSnake,
                                    int lengthOfSnake)
{
    // Kopf in die Mitte des Spielfeldes
    int row =
    int column =

    // Position für Schlange speichern
    for (int i = 0; i < lengthOfSnake; i++)
    {
        positionsOfSnake[i, 0] = row;
        positionsOfSnake[i, 1] = column + i;
    }
}
```

Bei dieser Variante soll der Kopf in der Mitte des Bildschirms erscheinen und der Rest der Schlange soll waagerecht nach rechts gezeichnet werden.

Das heißt, dass die Zeile gleich bleibt, aber die Spalte um 1 erhöht wird. Noch wird für Kopf und Körper das gleiche Zeichen verwendet. Das wird sich aber noch ändern!

Eintragen der Schlange in die aktuelle Spielsituation

Um die aktuelle Position der Schlange im Spielgeschehen darzustellen, wird diese Tabelle zeilenweise von oben nach unten durchlaufen und dann das Zeichen für die Schlange in das Array **`gameArea`** eingetragen. Wie viele Zeilen zu durchlaufen sind, wird durch die Länge der Schlange bestimmt. Diese Aufgabe soll eine Methode mit dem Bezeichner **`drawSnake()`** übernehmen!

Auch hier wieder die Überlegungen, wie der Methodenkopf auszusehen hat. In jedem Fall benötigt diese Methode zusätzlich Informationen: Sie muss die aktuellen Positionen der Schlange kennen, die in dem Array **`positionsOfSnake`** gespeichert sind. Sie muss die Länge der Schlange kennen, damit nur die betreffenden Zeilen des Arrays verwendet werden. Und Sie muss das Array kennen, in dem die Spielsituation einzutragen ist.

Um es kurz zu machen: Ein Rückgabewert ist wieder nicht unbedingt erforderlich, daher wird auch hier erneut der Datentyp **`void`** verwendet.

```
static void drawSnake(int[,] positionsOfSnake,
                      int lengthOfSnake,
                      char[,] gameArea)
{
    // Schlangenkörper in die Spielsituation eintragen
    // To Do

}
```

Noch ein Hinweis!

Bei den Parametern von Methoden muss ja deren Datentyp und ein Bezeichner angegeben werden. Der Datentyp ist immer der Datentyp derjenigen Information, die als Parameter übergeben wird. Für den Bezeichner kann man sich immer etwas ausdenken! In diesem Projekt wird allerdings immer der Bezeichner der zu übergebenden Information verwendet! Dass muss aber nicht so sein!

Aufgaben

1. Ergänzen Sie in der Methode **`initializeSnakePosition`** die beiden Berechnungen für **`row`** und **`column`**, um den Kopf der Schlange in der Mitte zu positionieren.
   Hinweis: Verwenden Sie die Konstanten, durch die auch die Größe des Spielfeldes festgelegt wird.
2. Überlege Sie sich, wie der Schleifenrumpf der **`for`**-Schleife abgeändert werden muss, wenn der Schlangenkörper in einer anderen Richtung eingezeichnet werden soll. Testen Sie Ihre Überlegungen!
3. Implementieren Sie die Methode **`drawSnake`**.

Momentaner Inhalt der Datei `Programm.cs`

Bei den Methoden unterhalb von **`Main`** ist aus Platzgründen nur ein leerer Anweisungsblock angegeben! Die Definition der Methoden ist aus den bisherigen Unterlagen zum Projekt ersichtlich.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Snake_Test
{
    class Program
    {
        // Voreinstellungen (Konstanten)
        // =============================
        // Fenstergröße festlegen
        const int windowWidth = 80;
        const int windowHeight = 40;

        // Unterer Rand für Spielinfos
        const int bottomBorder = 10;

        // Hintergrundfarbe Spielfeld
        const ConsoleColor gameBackgroundColor = ConsoleColor.White;

        // Zeichen für Spielfeldrand
        const char gameBorderCharacter = '#';

        // Maximale Länge der Schlange
        const int maxLengthOfSnake = 250;

        // Zeichen für Schlangenkörper
        const char snakeBodyCharacter = 'S';

        static void Main(string[] args)
        {
            // ================================================
            // Informationen zur Anzeige
            // ================================================
            // Der Inhalt dieses Arrays wird zur Darstellung
            // der aktuellen Spielsituation ausgegeben
            // gameArea[rows, columns]
            char[,] gameArea = new char[windowHeight - bottomBorder, win-
dowWidth];

            // ================================================
            // Informationen zur Schlange
            // ================================================
            // Enthält die Positionen des Schlangenkopfes
            // und Schlangenkörpers als Zeilen/Spalten Wertepaare
            // Die Werte müssen größer 0 sein
            int[,] positionsOfSnake = new int[maxLengthOfSnake, 2];

            // Aktuelle Gesamtlänge der Schlange
            int lengthOfSnake = 4;

            // Fenster initialisieren
            initializeWindow();

            // Spielfeld initialisiere
            initializeGameArea(gameArea);

            // Startpositionen der Schlange
            initializeSnakePosition(positionsOfSnake, lengthOfSnake);

            // Schlange in Array eintragen
            drawSnake(positionsOfSnake, lengthOfSnake, gameArea);

            // Inhalt des Spielfeldes auf
            // Bildschirm ausgeben
            drawScreen(gameArea);
        }

        // Methoden zu Initialisierung
        // Definition aus Platzgründen weggelassen
        // ========================================================
        // Konsolenfenster initialisieren
        static void initializeWindow()
        {
        }

        // Spielfeld initialisieren
        static void initializeGameArea(char[,] gameArea)
        {
        }

        // Anfangsposition der Schlange initialisieren
        static void initializeSnakePosition(int[,] positionsOfSnake,
                                            int lengthOfSnake)
        {
        }

        // Methoden zur Spiellogik
        // ========================================================
        // Gibt den Inhalt des Spielfeldes im Fenster aus
        static void drawScreen(char[,] gameArea)
        {
        }

        // Überträgt die Position der Schlange in das Spielfeld
        static void drawSnake(int[,] positionsOfSnake,
                              int lengthOfSnake,
                              char[,] gameArea)
        {
        }
    }
}
```

Schlangenfutter

Nun soll an zufälligen Stellen innerhalb des Spielfeldes ein Futterhappen für die Schlange erscheinen. Aus meiner Sicht ist es sinnvoll, wie schon beim Zeichen für den Bildschirmrand oder für die Schlange, auch dieses Zeichen mittels einer globalen Konstante zu definieren. Dann muss man nur an dieser einen Stelle ein anderes Zeichen angeben, damit sich das auf das gesamte Spiel auswirkt. In meinem Projekt habe ich den Bezeichner **`snakeFoodCharacter`** mit dem Wert `'o'` verwendet.

Das Erzeugen eines Futterhappens für die Schlange soll ebenfalls in einer Methode erfolgen, die dann bei Bedarf aufgerufen wird.

Bevor nun diese Methode entwickelt wird, muss man sich aber wieder überlegen, wie man die Position der Futterhappen speichert, um eine geeignete Datenstruktur zu verwenden.

Es liegt aber sicherlich nahe, dass man wie bereits bei der Schlange, eine Zeilen- und Spaltennummer speichern muss, bei der Futterhappen dann zu zeichnen ist. Wenn jeder Futterhappen den gleichen Wert haben soll, ist dies ausreichend. Um aber etwas mehr Abwechslung in das Spiel zu bringen, soll jeder Futterhappen einen anderen zufälligen Punktwert (innerhalb vorgegebener Grenzen) haben, um den die erreichten Spielpunkte des Spielers erhöht werden.

Die Position (Zeile/Spalte) der Futterhappen soll ebenfalls zufällig sein.

Somit sind pro Futterhappen drei Werte zu speichern: Zeilennummer, Spaltennummer und Wert.

Es wird also wieder eine Tabelle benötigt, die pro Tabellenzeile drei Werte enthält. Jede Zeile entspricht einem Futterhappen.

Zeile	Spalte	Wert
5	35	8
56	12	2
13	42	4
30	17	9
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Es wird also erneut ein zweidimensionales Array benötigt, wobei dieses Mal drei Spalten vorhanden sein sollen.

Wie auch schon bei der Schlange, ist es auch hier sinnvoll, die maximale Anzahl der Zeilen (maximale Anzahl der Futterhappen) mittels einer Konstanten festzulegen und dann nur noch mit dieser Konstanten zu arbeiten.

Des Weiteren soll in einer Variablen die Anzahl der bereits erzeugten Futterhappen gespeichert werden. Dieser Wert ändert sich ja im Verlauf des Spieles; es werden neue Futterhappen (bis zur Maximalanzahl) erzeugt und es werden auch Futterhappen von der Schlange gefressen, die dann aus dem Array gelöscht werden müssen.

Für die maximale Anzahl der Futterhappen soll eine Konstante mit der folgenden Bezeichnung verwendet werden: **`maxAmountOfSnakeFood`**. Und die Variable für vorhandene Anzahl an Futterhappen soll den Bezeichner **`amountOfSnakeFood`** haben.

Die Konstante **`maxAmountOfSnakeFood`** wird wiederum vor der **`Main`**-Methode bei den anderen Konstanten definiert. Die Variable **`amountOfSnakeFood`** wird innerhalb der Main-Methode (zu Beginn) definiert.

Die Datenstruktur zur Speicherung der Futterhappen hat den Bezeichner **`positionsOfSnakeFood`** und wird wie dargestellt erzeugt.

```
// ============================================
// Informationen zu den Futterhappen
// ============================================
// Enthält die Position und Wert aller Futterhappen
// Maximal 100 Futterhappen sind möglich
// positionsOfSnakeFood[row, column, value]
int[,] positionsOfSnakeFood = new int[maxAmountOfSnakeFood, 3];
```

Erzeugen der Futterhappen

Auch dieser Vorgang wird wieder in einer Methode definiert, die bei Bedarf aufgerufen wird. Diese Methode soll den Bezeichner **`createSnakeFood`** erhalten. Auch jetzt muss man sich wieder überlegen, wie der Methodenkopf aussehen soll, d.h. welche Informationen benötigt die Methode und soll sie einen Ergebniswert zurückliefern.

Okay...da ja diese Methode die soeben besprochene Datenstruktur verändern soll, muss zumindest dieses zweidimensionale Array **`positionsOfSnakeFood`** übermittelt werden. Dann kann die Methode dort einen neuen Futterhappen dort eintragen.

Diese Methode darf aber nur dann einen neuen Futterhappen erzeugen, wenn die maximale Anzahl der Futterhappen noch nicht erreicht wurde. Also soll die Methode keinen neuen Futterhappen erzeugen, wenn diese maximale Anzahl bereits erreicht wurde. Man kann dies auch anders realisieren!

Zum Beispiel, dass man die Methode nur dann aufruft, wenn folgende Überprüfung wahr ist:

```
if (amountOfSnakeFood < maxAmountOfSnakeFood)
    createSnakeFood(…);
```

Aber so überprüft die Erzeugungsmethode selbst, ob dies zutrifft und somit muss die Methode **`createSnakeFood`** die momentane Anzahl der Futterhappen kennen!

Was muss die Methode noch wissen? Futterhappen dürfen nicht an Stellen erzeugt werden, die gerade durch den Schlangenkörper belegt sind! Also muss diese Methode wissen, wo sich momentan die Schlange befindet (steht in **`positionsOfSnake`**) und wie lange die Schlange momentan ist (steht in **`lengthOfSnake`**). Das Gleiche gilt auch für bereits existierendes Futter.

Die Position und der Wert eines Futterhappens soll ja „zufällig“ sein. Hierzu gibt es in Programmiersprachen fertige kleine „Maschinen“, die Zufallswerte erzeugen können. Man erstellt eine solche Maschine folgendermaßen: `Random randomNumbersGenerator = new Random();`

Der selbstgewählte Name der Maschine lautet **`randomNumbersGenerator`**. Da in diesem Projekt noch an anderen Stellen eine solche „Maschine“ benötigt wird, erstellen wir einmal diese Maschine in der Main-Methode und nutzen diese an allen Stellen, wo man sie benötigt.

Somit benötigt unsere Methode **`createSnakeFood`** insgesamt fünf Informationen:

1. den Zufallszahlengenerator
2. die Position der Schlange
3. die Länge der Schlange
4. die Position der bereits vorhandenen Futterhappen
5. die Anzahl der bereits vorhandenen Futterhappen

Uff...das war jetzt eine ganze Menge an Information! Einmal sacken lassen :-)

Jetz soll noch überlegt werden, ob ein Rückgabewert erforderlich ist. Ja, in diesem Fall soll es einen Rückgabewert geben, nämlich die neue Anzahl von Futterhappen. Diese Information ist ja eine Ganzzahl, also ein Integer-Wert. Somit sieht der Methodenkopf folgendermaßen aus.

```
static int createSnakeFood(Random randomNumbersGenerator,
                           int[,] positionsOfSnake,
                           int lengthOfSnake,
                           int[,] positionsOfSnakeFood,
                           int amountOfSnakeFood)
```

Wir haben also in unserer Methode Zugriff auf all diese Informationen und teilen dem Aufrufer am Ende der Methode die (neue) aktuelle Anzahl von Futterhappen mit.

Jetzt geht es an die Definition dieser Methode! Wie sieht deren Ablauf wohl aus?

1. Direkt zu Beginn der Methode wird überprüft, ob bereits die maximale Anzahl von Futterhappen erreicht wurde. Falls ja, wird die Methode beendet. Hierzu wird die aktuelle Anzahl von Futterhappen zurück geliefert.
2. Es werden die für einen neuen Futterhappen benötigten Zahlen per Zufallszahlengenerator gezogen (Zeile, Spalte, Wert).
3. Nun wird überprüft, ob an dieser Stelle bereits ein Futterhappen existiert. Falls ja, soll die Methode ohne Erzeugung eines neuen Futterhappens wie zuvor enden.
4. Anschließend wird überprüft, ob sich an dieser Stelle ein Körperteil der Schlange befindet. Falls ja, wird erneut die Methode ohne einen neuen Futterhappen beendet.
5. Jetzt kann der neue Futterhappen in dem Array **`positionsOfSnakeFood`** an der ersten freien Stelle eingetragen werden. Dann wird die aktuelle Anzahl von Futterhappen um Eins erhöht. Dann endet die Methode und liefert die neue Anzahl von Futterhappen zurück.

Okay...hier folgt nun eine Beispiel-Implementation dieser Methode.

```
static int createSnakeFood(Random randomNumbersGenerator,
                           int[,] positionsOfSnake,
                           int lengthOfSnake,
                           int[,] positionsOfSnakeFood,
                           int amountOfSnakeFood)
{
    // Keine neuen Futterhappen, wenn bereits Maximum erreicht
    if (amountOfSnakeFood >= maxAmountOfSnakeFood)
        return amountOfSnakeFood;

    // Drei Zufallszahlen für die Position
    // und den Wert des Futterhappens ziehen
    // Die Position soll innerhalb des
    // Spielfelds mit einem Abstand von 2
    // zum Rand sein
    // Somit liegt der Wert für x zwischen
    // 3 und 76 und für y zwischen 3 und 26
    // für die Größe 80x40

    int column = randomNumbersGenerator.Next(3, windowWidth - 3);
    int row = randomNumbersGenerator.Next(3, windowHeight -
                                          bottomBorder - 3);
    int value = randomNumbersGenerator.Next(1,11);

    // Ein Futterhappen darf nur auf
    // einem leeren Feld erzeugt werden!

    // Zuerst prüfen, ob die Position bereits
    // durch einen anderen Futterhappen belegt ist

    // Annahme: Position ist frei
    bool isPositionEmpty = true;

    // Alle Futterhappen durchlaufen
    for (int i = 0; i < amountOfSnakeFood; i++)
    {
        // Hat der Futterhappen die gleiche Position?
        if (positionsOfSnakeFood[i, 0] == row &&
            positionsOfSnakeFood[i, 1] == column)
        {
            // Position ist belegt
            isPositionEmpty = false;
            // Schleife vorzeitig beenden
            break;
        }
    }

    // Falls keine Futterhappenposition
    if (isPositionEmpty == true)
    {
        // Prüfen, ob die Position bereits
        // duch den Schlangenkörper belegt ist
        for (int i = 0; i < lengthOfSnake; i++)
        {
            if (positionsOfSnake[i, 0] == row &&
                positionsOfSnake[i, 1] == column)
            {
                isPositionEmpty = false;
                break;
            }
        }
    }

    // Falls nicht, Futterhappen
    // am Ende hinzufügen und Anzahl inkrementieren
    if (isPositionEmpty == true)
    {
        positionsOfSnakeFood[amountOfSnakeFood, 0] = row;
        positionsOfSnakeFood[amountOfSnakeFood, 1] = column;
        positionsOfSnakeFood[amountOfSnakeFood, 2] = value;
        amountOfSnakeFood++;
    }

    return amountOfSnakeFood;
}
```

So...die Futterhappen stehen jetzt aber nur in dem Array **`positionsOfSnakeFood`**. Damit sie dann auch sichtbar werden, müssen sie noch in das Array **`gameArea`** übertragen werden.

Auch für diese Aufgabe wird eine eigene Methode erstellt.

Eintragen der Futterhappen in das Array `gameArea`

Die Positionen und Werte aller Futterhappen befinden sich in dem zweidimensionalen Array mit dem Bezeichner **`positionsOfSnakeFood`**. Damit die Futterhappen dann auch tatsächlich angezeigt werden, muss anhand der Positionsangaben in dem Array **`gameArea`** nun das festgelegte Zeichen für einen Futterhappen gespeichert werden, damit diese bei Ausführung der Methode **`drawScreen()`** auch auf dem Bildschirm erscheinen.

Diese Methode soll den Bezeichner **`drawSnakeFood`** erhalten. Wie schon zuvor ist zu überlegen, welche Informationen diese Methode in Form von Parametern benötigt und welcher Rückgabetyp nötig ist.

Im Wesentlichen benötigt diese Methode natürlich Informationen zu den Futterhappen. Konkret ist dies das Array mit den Positionsangaben **`positionsOfSnakeFood`** und den aktuellen Wert der Variable **`amountOfSnakeFood`**, also die aktuelle Anzahl an Futterhappen. Ebenso muss diese Methode auf das Array **`gameArea`** Zugriff haben, um das Zeichen für einen Futterhapppen dort eintragen zu können.

Da hier kein Ergebniswert zustande kommt, soll wieder der Rückgabetyp **`void`** zum Zuge kommen. Somit sieht der Methodenkopf wie folgt aus.

```
static void drawSnakeFood(int[,] positionsOfSnakeFood,
                          int amountOfSnakeFood,
                          char[,] gameArea)
```

Was muss nun innerhalb der Methode passieren? Anhand der Anzahl der Futterhappen wird für jeden Futterhappen dessen Zeilen- und Spaltennummer ermittelt und dann genau an dieser Position im Array **`gameArea`** das per Konstante **`snakeFoodCharacter`** festgelegte Zeichen eingetragen. Dieser Vorgang lässt sich per **`for`**-Schleife (da ja die Anzahl der Futterhappen bekannt ist!) erledigen. Dies wird gleich eine Aufgabe für Sie sein :-)

Anpassen der Main-Methode

Für den momentanen Stand des Projekts muss jetzt noch die Main-Methode des Programms soweit ergänzt werden, dass das Spielfeld inklusive der Startposition der Schlange gezeichnet wird und zumindest in einer offenen Schleife permanent neue Futterhappen erzeugt werden, solange das Spiel aktiv ist.

Ob das Spiel aktiv oder nicht mehr aktiv ist, soll durch eine boolsche Variable mitgeteilt werden. Diese Variable hat einen Bezeichner, der als Frage formuliert ist, nämlich **`isGameActive`**.

Diese Variable ist innerhalb von Main deklariert und erhält den Wert **`true`**. Die zuvor erwähnte und im Struktogramm auf Seite 1 dargestellte Schleife (eine **`do...while`**-Schleife) überprüft den Wert und solange dieser Wert **`true`** ist, wird die Schleife für den Spielablauf erneut durchlaufen.

Diese Variable sollte irgendwo zu Beginn der Main-Methode, dort wo auch alle anderen Variablen bereits definiert sind, mit dem Wert **`false`** erzeugt werden.

Nach den Aufrufen der verschiedenen Initialisierungsmethoden kommt nun die **`do...while`**-Schleife, in deren Schleifenrumpf nun der eigentlich Spielablauf stattfindet.

Und direkt vor der **`do...while`**-Schleife wird die Variable **`isGameActive`** auf true gesetzt.

Auf der nächsten Seite ist die momentane Version der **`Main`**-Methode dargestellt.

```
static void Main(string[] args)
{
    // ============================================
    // Variablen
    // ============================================
    // Zum Ziehen von Zufallszahlen für Futterhappen
    Random randomNumbersGenerator = new Random();

    // ============================================
    // Informationen zum Spiel
    // ============================================
    // Ist das Spiel noch aktiv?
    bool isGameActive = false;
    // ============================================

    // ============================================
    // Informationen zur Anzeige
    // ============================================
    // Der Inhalt dieses Arrays wird zur Darstellung
    // der aktuellen Spielsituation ausgegeben
    // gameArea[rows, columns]
    char[,] gameArea = new char[windowHeight - bottomBorder, windowWidth];
    // ============================================

    // ============================================
    // Informationen zur Schlange
    // ============================================
    // Enthält die Positionen des Schlangenkopfes
    // und Schlangenkörpers als Zeilen/Spalten Wertepaare
    // Die Werte müssen größer 0 sein
    int[,] positionsOfSnake = new int[maxLengthOfSnake, 2];

    // Aktuelle Gesamtlänge der Schlange
    int lengthOfSnake = 4;
    // ============================================

    // ============================================
    // Informationen zu den Futterhappen
    // ============================================
    // Enthält die Position und Wert aller Futterhappen
    // Maximal 100 Futterhappen sind möglich
    // positionsOfSnakeFood[row, column, value]
    int[,] positionsOfSnakeFood = new int[maxAmountOfSnakeFood, 3];

    // Anzahl der Futterhappen
    int amountOfSnakeFood = 0;
    // ============================================

    // ============================================
    // Initialisierung des Spiels
    // Fenster initialisieren
    initializeWindow();

    // Spielfeld initialisiere
    initializeGameArea(gameArea);

    // Startpositionen der Schlange festlegen
    initializeSnakePosition(positionsOfSnake, lengthOfSnake);

    // Schlange in Array eintragen
    drawSnake(positionsOfSnake, lengthOfSnake, gameArea);
    // ============================================

    // ============================================
    // Beginn der Spiellogik
    // ============================================
    // Spiel starten
    isGameActive = true;
    // Inhalt des Spielfeldes und Statusinformationen
    // auf Bildschirm ausgeben
    drawScreen(gameArea);

    // Ablauf des Spieles
    do
    {
        // Wartezeit
        Thread.Sleep(100);

        // Futterhappen erzeugen
        amountOfSnakeFood = createSnakeFood(randomNumbersGenerator,
                                            positionsOfSnake,
                                            lengthOfSnake,
                                            positionsOfSnakeFood,
                                            amountOfSnakeFood);

        // Eintragen der Position der Futterhappen im Spielfeld
        drawSnakeFood(positionsOfSnakeFood, amountOfSnakeFood, gameArea);

        // Inhalt des Spielfeldes auf dem Bildschirm ausgeben
        drawScreen(gameArea);
    }
    while (isGameActive == true);
}
```

Eine Anweisung zu Beginn der **`do...while`**-Schleife ist noch neu!

Die Anweisung **`Thread.Sleep(100)`** bewirkt, dass das Programm für 100 ms nichts macht, sich quasi „schlafen legt“. Dies wird gemacht, damit der Spielablauf etwas ausgebremst wird. Der Parameter in den runden Klammern gibt also die Zeit in Millisekunden an, die das Programm pausieren soll. Natürlich kann man den Wert vergrößern, dann läuft das Spielgeschehen langsamer oder den Wert verringern, dann läuft es eben schneller, oder ganz weglassen.

Damit diese „Maschine“, die das Pausieren macht, auch gefunden wird, muss zu Beginn des Programms der Namensbereich **`System.Threading`** eingebunden werden.

Diese Anweisung ganz am Anfang der Datei lautet also: `using System.Threading;`

Aufgaben

1. Ergänzen Sie die Main-Methode in ihrer Datei **`Program.cs`** anhand des Beispiels.
2. Fügen Sie anhand der Beispiel-Implementation die Methode **`createSnakeFood()`** hinzu.
3. Definieren Sie die Methode **`drawSnakeFood()`**.

Wenn Sie alles gemacht haben, sollten die Spielfeldumrandung und die Schlange erscheinen. Anschließend sollten Futterhappen an zufälligen Positionen angezeigt werden.