Die Lösungsidee:

Um herausfinden zu können, wie oft Frederick eine Zahl hintereinander werfen müsste, um das Ziel zu erreichen, beziehungsweise ob es überhaupt möglich ist mit nur einer Würfelzahl das Ziel zu erreichen, muss man den Spielverlauf simulieren.
Dies kann man durch das Hochzählen einer Variable erreichen, die mit dem Wert 0 startet, welcher für das Startfeld steht. Nach jedem Hochzählen der Variable muss man lediglich kontrollieren, ob der Wert der Variable, dem eines Feldes, welches mit einer Leiter verbunden ist entspricht. Ist dies der Fall, setzt man den Wert der Variable auf das andere Feld, welches mit der selben Leiter verbunden ist.
Dieser Prozess wiederholt sich solange, bis die Variable einen Wert >= 100 hat. Ist der Wert genau 100 ist die Simulation beendet. Ist der Wert hingegen größer als 100, so muss man lediglich den Teil der über 100 liegt von 100 abziehen, um die zu vielen Schritte wieder zurückzulaufen. Hier kommt es allerdings zu folgendem Problem:
Wenn zum Beispiel das Spiel mit der Würfelzahl 6 simuliert wird und man steht zufällig auf der 97, so wird die Simulation niemals enden, da [Feld] 97 + 6 [Schritte] in diesem Fall zu [Feld] 97 + 3 [Schritte] - 3 [Schritte] führt, wodurch man nach jedem Zug an der selben Position stehen würde wie vorher. Um dies zu verhindern kann man jedoch einfach mitzählen, wie oft der Spieler bereits auf jedem Feld war und die Simulation abbrechen, wenn ein Feld mehrfach betreten wird.
Während die Schritte auf dem Brett simuliert werden muss lediglich durch eine weitere Variable mitgezählt werden, wie viele Schritte gemacht wurden. Diese Variable kann dann am Ende ausgegeben werden, falls es möglich ist das Ziel des Spiels zu erreichen.

Die Umsetzung mit Quelltext:

Die Umsetzung der Lösungsidee erfolgt in meiner Bearbeitung in der Programmiersprache Ruby. Das Programm beziehungsweise Script („auf_und_ab.rb“) liegt im Ordner „Junioraufgabe 1“ und stellt zeitgleich Quelltext und ausführbare Datei dar. Ich habe das Script mit der Ruby Version 2.5.3-1-x64 in der CMD von Windows 10 getestet. Neben dieser grundsätzlichen Lösung liegt im Unterordner „Lösung mit Oberfläche“ eine Projektmappe für Visual Studio. Das darin enthaltene Programm ist in Visual Basic geschrieben und bietet die Möglichkeit eigene Werte einzutragen, während das Ruby Script ausschließlich für das Beispiel Spielbrett anwendbar ist.
Da das Prinzip beider Lösungen gleich ist, erfolgt die Erklärung des Quelltextes ausschließlich am Ruby Script:

```ruby
def start(z)
    i=0
    s=0
    $benutzteFelder = Array.new(100, 0)
    while i < 100 && zielErreichbar(i) do
        i+=z
        i = leiterBenutzen(i)
        i = zuruecklaufen(i)
        s=s+1
    end
    if i==100
        puts "Mit der Würfelzahl #{z} muss man #{s}-mal würfeln, um das Ziel zu erreichen."
    else
        puts "Mit der Würfelzahl #{z} scheint es nicht möglich zu sein, in das Ziel zu gelangen."
    end
end
```

Wie der Name dieser Methode es vermuten lässt, startet hier die Simulation eines Spiels. Der Methode übergibt man den Wert, der mit dem Würfel immer wieder geworfen werden soll, in die Variable *z* als Integer. Die Variable *i* ist ein Integer, welcher das derzeitige Spielfeld, auf dem man steht, enthält. Die Variable *s* hingegen zählt die Spielzüge mit.
Die while-Schleife in der Funktion läuft solange, bis das Ziel erreicht ist oder die Methode *zielErreichbar(i)* „entscheidet“, dass das Ziel nicht erreichbar ist.
Nachdem ein Schritt gemacht wurde durch das addieren der Würfelzahl auf das derzeitige Feld, wird durch die Methode *leiterBenutzen(i)* kontrolliert, ob das Feld auf dem man gerade wäre mit einer Leiter verbunden ist und setzt das Feld auf das andere Feld, welches mit dieser Leiter verbunden ist, sollte man auf einem Feld mit Leiter stehen. Anschließend wird durch die Methode *zuruecklaufen(i)* kontrolliert, ob man über das 100. Feld hinüberlaufen würde und setzt in einem solchen Fall das Feld so viele Schritte vor dem 100. Feld zurück, wie man eigentlich über dieses hinübergelaufen wäre.
Ist die while-Schleife beendet folgt die Ausgabe der Ergebnisse der Simulation.

```
def leiterBenutzen(i)
   if i==6
      i=27
      elsif i==27
         i=6
      elsif i==14
         i=19
      elsif i==19
         i=14
      elsif i==21
         i=53
      elsif i==53
         i=21
      elsif i==31
         i=42
      elsif i==42
         i=31
      elsif i==33
         i=38
      elsif i==38
         i=33
      elsif i==46
         i=62
      elsif i==62
         i=46
      elsif i==51
         i=59
      elsif i==59
         i=51
      elsif i==57
         i=96
      elsif i==96
         i=57
      elsif i==65
         i=85
      elsif i==85
         i=65
      elsif i==68
         i=80
      elsif i==80
         i=68
      elsif i==70
         i=76
      elsif i==76
         i=70
      elsif i==92
         i=98
      elsif i==98
         i=92
      end
      return i
end
```

Diese Methode erhält die derzeitige Position auf dem Spielbrett und ändert diese, falls man auf einem Feld mit Leiter steht zu dem anderen Feld, was mit derselben Leiter verbunden ist. Anschließend gibt die Methode die eventuell neue Position zurück.

```
def zuruecklaufen(i)
   if i>100
      i=i-100
      i=100-i
   end
   return i
end
```

Diese Methode erhält ebenfalls die derzeitige Position. Sollte diese über 100 liegen, was bei nur 100 Spielfeldern nicht möglich und nach Regeln des Spiels verboten ist, wird von der Position auf dem Brett 100 abgezogen, wodurch man weiß, wie viel Schritte man über die 100 hinausgelaufen ist. Anschließend wird dieser Wert wieder von 100 abgezogen, wodurch man die zu vielen Schritte zurückläuft. Auch hier wird die eventuell neue Position zurückgegeben.

```
def zielErreichbar(i)
   if $benutzteFelder[i] > 2
            return false
      else
            $benutzteFelder[i] = $benutzteFelder[i] + 1
            return true
      end
end
```

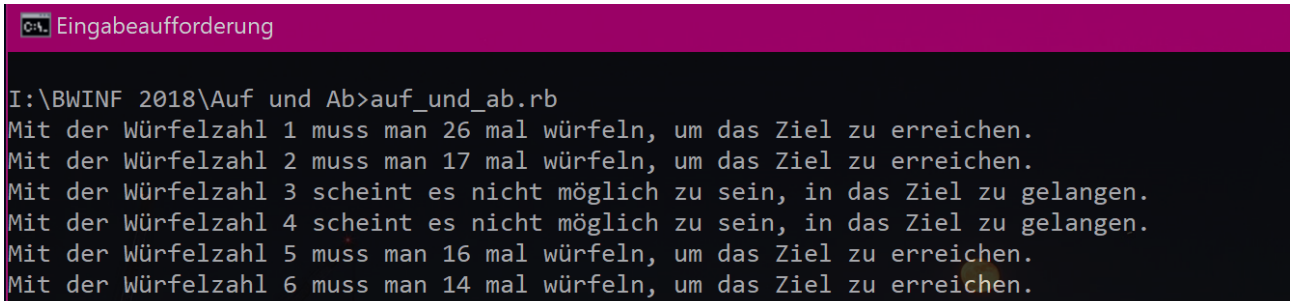
Die letzte Methode des Programms: In dieser Methode wird nach jedem Schritt auf dem Spielbrett, beziehungsweise jedem Durchlauf der while-Schleife aus der Methode *start(z)*, kontrolliert, ob das derzeitige Feld bereits zwei Mal betreten wurde. Ist dies der Fall so gibt die Methode false zurück. Ist dies nicht der Fall hingegen true. Welche Felder bereits benutzt wurden, wird in dem Array $benutzteFelder gespeichert, wobei dieser Array für jedes Feld auf dem Spielbrett einen Eintrag mit dem Startwert 0 hat. Dieser Wert wird nach jedem Schritt für das derzeitige Feld um eins erhöht.

```
start(1)
start(2)
start(3)
start(4)
start(5)
start(6)
```

Durch die letzten Zeilen des Programmes wird die Simulation für jede der möglichen Würfelzahlen gestartet.

Beispiel (Demonstration des Ruby Scripts):

Da es nur 6 Würfelzahlen gibt und für jede dieser das Programm angewendet werden soll, gibt es in dem Programm keine Möglichkeit zur Eingabe, sondern nur eine Ausgabe.
Die Ausgabe sieht wie folgt aus:



Beispiel (Demonstration der VB.net Anwendung):

In dieser Lösung ist es möglich ein beliebig großes Feld, mit anderen Leitern zu verwenden. Auch die Würfelzahl kann beliebig eingestellt werden.