# Aufgabe 4: Würfelglück

Team-ID: 00195

Team-Name: WLR

Bearbeiter/-innen dieser Aufgabe:
Baran Peters

21. November 2021

## Inhaltsverzeichnis

**1 Vorwissen** **1**

**2 Lösungsidee** **1**

**3 Umsetzung** **2**

**4 Beispiele** **3**
4.1 Beispieldaten Nr. 1 . . . . . 3
4.2 Beispieldaten Nr. 2 . . . . . 3
4.3 Beispieldaten Nr. 3 . . . . . 4
4.4 Beispieldaten Nr. 4 . . . . . 4

**5 Quellcode** **4**

## 1 Vorwissen

Bevor man die Dokumentation durchliest, sollte man sich die auf der BwInf-Seite verlinkten Regeln von "Mensch-Ärgere-Dich-Nicht" sowie die durch die Aufgabenstellung abgeänderten Regeln vertraut machen.

## 2 Lösungsidee

Das zu lösende Problem erfordert grundsätzlich eine Simulation von mehreren "Mensch-Ärgere-Dich-Nicht"-Spielen. Die gegebenen Würfel treten dabei einzeln immer gegeneinander an, sodass man am Ende jeden Würfel mit jedem anderen Würfel spielen lassen hat und dadurch eine Rangliste anhand der Siege (bzw. Siegeswahrscheinlichkeit) ausgeben kann. Um dadurch zwischen jedem Würfel einen Sieger bestimmen zu können, muss man in jeder Würfelkombination genügend Spiele ausführen, sodass die empirischen Gewinnwahrscheinlichkeiten stellvertretend für die stochastischen Wahrscheinlichkeiten als eine gute Näherung verwendet werden können.

Das sogenannte "Gesetz der großen Zahlen" besagt, dass je öfter man ein Zufallsexperiment (was die einzelnen Spiele zwischen den Würfeln grundsätzlich auch sind; sie bestehen aus einer Folge von mehreren, unabhängigen Zufallsexperimenten, dem Werfen der Würfel) durchführt, desto mehr nähern sich die empirischen Vorhersagen denen der Stochastik. Solange man keine allzu große Abweichung zwischen Gewinnwahrscheinlichkeiten folgender Runden hat, kann man davon ausgehen, dass die Wahrscheinlichkeiten sehr gut angenähert worden sind und man damit den besseren Würfel bestimmen kann. Aus den dabei entstehenden Wahrscheinlichkeiten werden jeweils für jeden Würfel am Ende Gewinnwahrscheinlichkeiten angegeben, aus denen sich eine Bestenliste ergibt. Alternativ zu den durchschnittlichen

Gewinnwahrscheinlichkeiten kann man auch einfach nur die einzelnen Siege der Würfel zählen. Allerdings kann es sein, dass ein Würfel, der sonst in der Bestenliste über einem anderen Würfel stehen würde, dabei unter diesem landen kann, weil der nun bessere Würfel nur leicht besser war als der andere Würfel und dadurch den Sieg bekommt. Das ist meistens einfach auf Gegebenheiten zurückzuführen, die nichts mit der eigentlichen Nützlichkeit des Würfels zu tun haben, sondern mit Glück. Die Nutzung der Gewinnwahrscheinlichkeiten ermöglicht daher eine genauere Bestimmung der Gewinnwahrscheinlichkeiten.

Die Simulation des Spiels an sich erfolgt nach den vorgegebenen Regeln des Herstellers und der Aufgabenstellung. Die einzelnen Spieler mit den jeweiligen Würfeln ziehen immer mit der vordersten Figur, die ziehbar ist. Eine Figur ist nicht ziehbar, falls sie entweder schon eine Runde vollendet hat, also wieder zum Anfangs-Feld gelangt ist, oder mit der jeweiligen Augenzahl auf das Feld einer eigenen Figur ziehen würde. Falls keine Figur auf den regulären Spielfeldern ziehbar sein sollte, wird versucht eine Figur auf den Zielfeldern zu ziehen, sodass dadurch möglicherweise eine Figur des jeweiligen Spielers in der Lage sein wird, in das frei gewordene Zielfeld zu rücken. Dies wäre der Fall, wenn beispielsweise bei dem Würfel [1,1,1,1,6] eine Figur, die eine Runde vollendet hat, nicht auf das Zielfeld rücken kann, da nur das zweite Feld des Zielfeldes frei ist. Hier ist ein Ziehen der Figuren auf den Zielfeldern sinnvoll. Falls ein Spieler weder auf den Zielfeldern noch auf dem Spielfeld eine Figur ziehen kann, ist er spielunfähig, kann also theoretisch nur noch durch eine Interaktion mit dem Gegner wieder ziehen.

Falls nun beide Spieler in einen solchen Zustand geraten sollten, wenn die letzten Figuren eine Runde vollendet haben, es aber nicht möglich ist mit den jeweiligen Würfeln in die Zielfelder zu ziehen, dann wird das Spiel abgebrochen und keiner der Spieler bekommt einen Sieg, da es sich um ein Unentschieden handelt.

# 3 Umsetzung

Die Lösungsidee wird in einem Programm der Sprache JavaScript umgesetzt. Voraussetzung ist hierbei die Nutzung der neusten Node.JS Laufzeitumgebung. Nachdem die Daten über die Kommandozeile eingelesen werden, beginnt der hauptsächliche Teil des Programms.

Das Programm liest die Eingaben ein, iteriert über die verschiedenen Kombinationen der Würfel und startet pro Kombination daraufhin mit `simulateGames()` die Koordination der Simulation der Spiele. `simulateGames()` dahingegen ist zuständig die Spiele über `playGame()` zu starten und gleichzeitig die Reihenfolge und Sinnhaftigkeit zu beachten. Das bedeutet zum einen die Spiele immer mit wechselndem Beginner zu starten, sowie darauf zu achten, ob die Wahrscheinlichkeiten sich zwischen den Spielen noch groß ändern. Das wird bewerkstelligt, indem geschaut wird, ob die Differenz der Wahrscheinlichkeiten von dem gewinnenden Spieler kleiner als 0,5% beträgt. Ein Problem, welches damit aufkommt, ist, dass das an sich erst funktioniert, wenn beide Spieler mindestens einen Sieg haben, da die Wahrscheinlichkeit über $P(G) = \frac{gewonneneSpieler}{gespielteSpiele}$ berechnet wird. Die Wahrscheinlichkeit eines Spielers, der nur gewonnen hat wäre 100%. Damit lässt sich nicht überprüfen, ob genug Spiele gespielt wurden und man, wie in der Lösungsidee genannt, die empirischen Wahrscheinlichkeiten gut angenähert hat. Daher kann man über die kumulierte Binomialverteilung eine Mindestspielanzahl bestimmen, bei der es sehr wahrscheinlich ist, dass bei zwei Würfeln, die eine ähnliche Gewinnwahrscheinlichkeit besitzen, beispielsweise 49% sowie 51%,(also, wo der eine Würfel nur aufgrund einer stochastisch unwahrscheinlichen Niederlagensträhne keinen Sieg erringen konnte), die Wahrscheinlichkeit mindestens einen Sieg zu erlangen 99,9% beträgt:

$$1 - (0.49)^x \cdot x - (0.51)^x \cdot x = 0.9999 \leftrightarrow x = 14,34 \approx 15$$

Bei 15 Spielen kann man sich daher sicher sein, dass auch bei zwei Würfeln, die fast gleich große Gewinnwahrscheinlichkeiten haben, beide mindestens einen Sieg davontragen werden. Ab dieser Spielanzahl kann man davon ausgehen, dass, falls nur ein Würfel alle Spiele gewonnen haben sollte, es sich hierbei nicht um "pures" Glück handelt.

Der hauptsächliche Teil des Programs läuft in `playGame()` ab, wo die Spiele an sich stattfinden. Das Spielfeld wird über ein Array mit 40 Elementen dargestellt. Falls ein Feld nicht besetzt sein sollte, beinhaltet es eine 0, sonst ein Array mit einem Kennzeichen für den Spieler und die gewanderte Distanz der Figur. Feld 1 bzw. Feld 20 stellt das Anfangsfeld des Spieler 1 bzw. Spieler 2 dar. Dadurch stehen sich die beiden Anfangsfelder, wie von den Regeln gefordert, gegenüber. Mithilfe einer Endlosschleife, die nur unter der Bedingung aufhört, dass die Zielfelder eines Spielers voll sind oder beide Spieler keinen Zug mehr durchführen können, wird der Spielverlauf simuliert. Jede Iteration kann dabei als ein Zug, oder eher ein Würfelwurf gesehen werden. In der Endlosschleife wird hauptsächlich mit if-Bedingungen überprüft, ob bestimmte Situationen gegeben sind, und dementsprechend der Zug dann gespielt oder ausgesetzt.

Während der Spielschleife wird zuerst überprüft, ob man eine sechs gewürfelt hat. Falls ja, wird versucht eine neue Figur ins Spiel zu bringen. Im weiteren Verlauf werden die auf dem Spielfeld verbleibenden Stücke bestimmt, diese nach ihrer Position sortiert und dann durchiteriert. In der Schleife wird dann überprüft, ob die Figur schon eine Runde vollendet hat, also wieder bei seinem Startfeld angelangt ist. Falls ja, kann die Figur nun in die Zielfelder gezogen werden. Falls nicht, wird überprüft, ob der Zug außerhalb des Arrays des Spielfeldes fallen würde. Dementsprechend wird beachtet, dass man bei einem Zielfeld mit einem Index über 40 die überstehenden Felder von dem Anfang des Spielfeld-Arrays aus abzählt. Vor den jeweiligen Zügen wird auch kontrolliert, ob man bei einem Zug mit der Figur insgesamt mehr als 40 Felder gelaufen wäre, also die Runde überschreiten würde, was verboten wäre. In dem Fall kann mit der ausgehend vom Startfeld verbleibenden Feldanzahl nur direkt auf die Zielfelder gezogen werden, da diese in dem Fall als Verlängerung des Spielfeldes gesehen wird. Falls eine Figur in der Lage ist zu ziehen, wird nach dem Zug natürlich die Schleife beendet.

Im Folgenden wird, falls die Schleife vollständig durchiteriert wurde, die Möglichkeit des Ziehens auf den Zielfeldern überprüft. Das vollständige Iterieren in der vorherigen Schleife bedeutet, dass für die Spielsteine auf den regulären Feldern kein Zug gefunden werden konnte. Falls das auch nicht klappen sollte, wird geprüft, ob der Spieler gar nicht mehr ziehen kann. Das ist der Fall, wenn der Spieler mit seiner vordersten Figur auf dem regulären Spielfeld nicht mehr weiter ziehen kann, kein Zielfeld mehr mit den möglichen Augenzahlen erreicht werden kann und auch keine Figur auf den Zielfeldern bewegt werden kann.

Zuletzt wird überprüft, ob der Spieler gewonnen hat oder beide Spieler "feststecken", falls ja, wird die Iteration in beiden Fällen beendet.

# 4 Beispiele

- Ausgabe besteht aus den einzelnen Würfeln mit ihren Seiten, ihrer durchschnittlichen Gewinnwahrscheinlichkeit und Position in der Bestenliste.
- Die hier dargestellten Ergebnisse können bei eigener Durchführung vor allem bei den Wahrscheinlichkeiten leicht variieren, da diese natürlich auch nicht definit bestimmt werden können, da es sich um Zufallsexperimente handelt.

## 4.1 Beispieldaten Nr. 1

*Eingabe:* wuerfel0.txt aus den Beispiel-Eingaben
*Ausgabe:*

```
Würfel-Bestenliste:
1. 6 Seiten -> : 1 1 1 6 6 6; 93.40%
2. 6 Seiten -> : 1 2 3 4 5 6; 62.80%
3. 12 Seiten -> : 1 2 3 4 5 6 7 8 9 10 11 12; 56.60%
4. 10 Seiten -> : 0 1 2 3 4 5 6 7 8 9; 56.20%
5. 20 Seiten -> : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20; 31.00%
6. 4 Seiten -> : 1 2 3 4; 0.00%
```

## 4.2 Beispieldaten Nr. 2

*Eingabe:* wuerfel1.txt aus den Beispiel-Eingaben
*Ausgabe:*

```
Würfel-Bestenliste:
1. 6 Seiten -> : 2 3 4 5 6 7; 67.20%
2. 6 Seiten -> : 1 2 3 4 5 6; 59.20%
3. 6 Seiten -> : 3 4 5 6 7 8; 54.40%
4. 6 Seiten -> : 4 5 6 7 8 9; 40.40%
5. 6 Seiten -> : 5 6 7 8 9 10; 20.60%
6. 6 Seiten -> : 6 7 8 9 10 11; 15.20%
```

### 4.3 Beispieldaten Nr. 3

*Eingabe:* wuerfel2.txt aus den Beispiel-Eingaben
*Ausgabe:*

```
Würfel-Bestenliste:
1. 6 Seiten -> : 1 6 6 6 6 6; 98.50%
2. 6 Seiten -> : 1 1 6 6 6 6; 75.00%
3. 6 Seiten -> : 1 1 1 6 6 6; 51.50%
4. 6 Seiten -> : 1 1 1 1 6 6; 23.50%
5. 6 Seiten -> : 1 1 1 1 1 6; 1.50%
```

### 4.4 Beispieldaten Nr. 4

*Eingabe:* wuerfel3.txt aus den Beispiel-Eingaben
*Ausgabe:*

```
Würfel-Bestenliste:
1. 4 Seiten -> : 1 2 5 6; 77.80%
2. 6 Seiten -> : 1 2 3 4 5 6; 68.80%
3. 8 Seiten -> : 1 2 3 4 5 6 7 8; 66.20%
4. 10 Seiten -> : 0 1 2 3 4 5 6 7 8 9; 41.60%
5. 12 Seiten -> : 1 2 3 4 5 6 7 8 9 10 11 12; 37.20%
6. 20 Seiten -> : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20; 8.60%
```

## 5 Quellcode

```
function main() {
  var diceCount = +readline();

  var dices = [];

  for (let i = 0; i < diceCount; i++) {
    let diceInput = readline().split(" ");

    // dices[x][1] = Summe aller Gewinnwahrwscheinlichkeiten
    dices.push([diceInput, 0]);
  }

  // For-Schleife zur Kombinationsbestimmung
  for (let i = 0; i < diceCount; i++) {
    let x = i;
    while (x < diceCount) {
      if (x != i) {
        let result = simulateGames(dices[i][0], dices[x][0]);

        // Addieren der jeweiligen Gewinnwahrscheinlichkeiten aus dem Spiel
        dices[i][1] += result[1]
        dices[x][1] += result[2]
      }

      x++;
    }
  }

  dices.sort((a, b) => b[1] - a[1])

  // Ausgeben der Ergebnisse
  console.log("Würfel-Bestenliste:")
  dices.forEach((x,i) => console.log(`${i + 1}. ${x[0][0]} Seiten -> : ${x[0].slice(1).
    join(" ")}; ${(x[1] / (dices.length - 1) * 100).toFixed(2) + '%'}`))
}

// Simuliere (mehrere) Spiele zwischen Spielern mit diesen Würfeln, bis die relativen
    Häufigkeiten der Spiele nicht mehr signifikant steigen
```

```
// gibt ein Array mit true bzw. false zurück, falls Spieler 1 mit Würfel 1 bzw. Spieler 2 mit Würfel 2 gewonnen hat
function simulateGames(dice1, dice2) {
  // wins[0] = Siege Würfel 1, wins[1] = Siege Würfel 2
  let wins = [0, 0];
  let playedGames = 0;

  // Letzte relative Häufigkeit von der Gewinnwahrscheinlichkeit von Würfel 1 & 2
  let lastWinPercentage = [0, 0];

  while (true) {
    let gameResult;
    if ((playedGames + 1) % 2 == 0) {
      gameResult = playGame(dice1, dice2, true);
    } else {
      gameResult = playGame(dice1, dice2, false);
    }

    // Egal wie das Spiel ausgegangen ist, wird die Spielanzahl um 1 erhöht
    playedGames++;

    // Sieganzahl um 1 erhöhen, falls wir kein Unentschieden haben
    if (gameResult != 0) {
      wins[gameResult - 1]++;
    }

    // Prüfung, ob Abweichung zwischen Wahrscheinlichkeiten ist kleiner als 0.5% & beide Spieler gleich oft anfangen durften
    // ==> Falls ja, Stoppen mit weiteren Spielen & Ausgabe des Ergebnis
    if (
      Math.abs(
        lastWinPercentage[gameResult - 1] - wins[gameResult - 1] / playedGames
      ) < 0.005 &&
      playedGames % 2 == 0 &&
      playedGames >= 15
    ) {
      break;
    } else {
      lastWinPercentage[gameResult - 1] = wins[gameResult - 1] / playedGames;
    }
  }

  // Zurückgeben der auf 2 Nachkommastellen gerundeten Wahrscheinlichkeiten sowie den Gewinner
  return [wins[1] > wins[0] ? false: true, Math.round((wins[0] / playedGames) * 100) / 100, Math.round((wins[1] / playedGames) * 100) / 100]
}

// Je nachdem, ob Würfel 1 starten soll, wird true bzw. false als Argument angegeben bei dice1Begins
// gibt true bzw. false zurück, wenn der Gewinner Würfel 1 bzw. 2 ist; 0 bei einem Unentschieden
function playGame(dice1, dice2, dice1Begins) {
  // Spielfeld
  let board = new Array(40).fill(0);

  // Besetzung der Felder
  board[0] = [1, 0];
  board[20] = [2, 0];

  // Verbleibende Steine der Spieler;
  // leftPieces[0] = Steine des Spieler 1, leftPieces[1] = Steine des Spieler 2
  let leftPieces = [3, 3];

  // jeweiligen Zielfelder der Spieler
  let goalFields = [
    [0, 0, 0, 0],
    [0, 0, 0, 0],
  ];

  let turnNumber = 1;

  // Array, um festzuhalten, wann welcher Spieler keine Zugmöglichkeit mehr hat.
```

```
  let isStuck = [false, false];

  // Falls Würfel 1 nicht beginnen soll, wird turnNumber auf 2 gesetzt, da bei geraden
    Zahlen Würfel 1 spielt & ungeraden Zahlen Würfel 2
  if (!dice1Begins) {
    turnNumber = 2;
  }

  // Gameloop; jede Iteration 1 Spielzug (inkl. Würfel)
  while (true) {
    let dice;

    // Kennzeichner auf dem jeweiligen Spielfeld
    let mark;

    // Index des Anfangsfeld des Spielers (ausgehend von board)
    let startField;

    // Falls Zugnummer ungerade ist, spielt Spieler 1, sonst Spieler2
    if (turnNumber % 2 != 0) {
      dice = dice1;
      mark = 1;
      startField = 0;
    } else {
      dice = dice2;
      mark = 2;
      startField = 20;
    }

    // isStuck wieder auf false setzen, falls nun doch ein Zug möglich sein sollte
    isStuck[mark - 1] = false;

    let diceThrow = simulateDiceThrow(dice);

    if (diceThrow == 6) {
      if (leftPieces[mark - 1] > 0) {
        if (board[startField] != 0) {
          if (board[startField][0] == mark) {
            // Falls das 6.te Feld nach dem Startfeld auch besetzt sein sollte, kann man
  keine weitere Figur hinausbringen
            if (board[startField + 6][0] != mark) {
              // Falls es von keiner gegnerischer Figur besetzt ist ==> Nutzen des Zugs
  um neue Figur bzw. blockierende Figur weiterzuziehen
              if (board[startField + 6][0] != 3 - mark) {
                board[startField + 6] = [mark, 6];
                leftPieces[mark - 1]--;

                // Vorzeitiges Beenden dieser Iteration, da er nun erneut würfeln darf.
                continue;
              } else {
                // Falls es von einer gegnerischer Figur besetzt ist ==> schlagen
                board[startField + 6] = [mark, 6];
                leftPieces[mark - 1]--;

                // Steinanzahl des anderen Spielers um 1 erhöhen, da seine Figur
  geschlagen wurde
                leftPieces[2 - mark]++;

                // Vorzeitiges Beenden dieser Iteration, da er nun erneut würfeln darf.
                continue;
              }
            }
          } else {
            board[startField] = [mark, 0];
            leftPieces[mark - 1]--;
            leftPieces[2 - mark]++;
          }
        } else {
          board[startField] = [mark, 0];
          leftPieces[mark - 1]--;
        }
      }
    }
```

```
173
    // Bestimmen aller auf dem Feld verbleibenden Stücke
175 let pieces = [];

177 board.forEach((x, i) => {
      if (x != 0) {
179     if (x[0] == mark) {
          pieces.push(i);
181     }
      }
183 });

185 // Sortieren der Steine nach ihrem Fortschritt auf dem Spielfeld
    if (mark == 1) {
187   pieces = pieces.sort((a, b) => b - a);
    } else {
189   pieces = pieces.sort((a, b) => {
        a = 21 - a;
191     b = 21 - b;

193     if (a <= 0) {
          a = 40 - Math.abs(a);
195     }

197     if (b <= 0) {
          b = 40 - Math.abs(b);
199     }

201     return a < b ? -1 : 1;
      });
203 }

205 // Alle freien Felder in den Zielfeldern
    let freeFields = [];
207
    for (let l = 0; l < goalFields[mark - 1].length; l++) {
209   if (goalFields[mark - 1][l] == 0) {
        freeFields.push(l + 1);
211   }
    }
213
    // Durchgehen durch alle auf dem Spielfeld findbaren Steine, startend mit dem
    vordersten
215 check: for (var z = 0; z < pieces.length; z++) {
      let piece = pieces[z];
217
      // Falls ein Stein schon eine Runde vollendet hat, darf er nicht mehr gezogen
    werden
219   if (board[piece][1] == 40) {
        if (diceThrow <= 4 && diceThrow > 0) {
221       for (let field of freeFields) {
            if (field == diceThrow) {
223           goalFields[mark - 1][diceThrow - 1] = mark;
              board[piece] = 0;
225           break check;
            }
227       }
        }
229     continue;
      } else {
231     // Überprüfen, ob das Ende des "Spielfelds" erreicht wurde
        if (piece + diceThrow < 40) {
233       // Falls das Ende nicht erreicht wurde, wird überprüft ob man das Anfangsfeld
    überspringt
          // (Wird allerdings nur bei Spieler 2 der Fall sein, da bei Spieler 1 ein
    Überspringen des Feldes (ohne schon auf dem Anfangsfeld zu sein)
235       // nur mit dem Erreichen des Endes vom Spielfeld einhergeht)
          if (board[piece][1] + diceThrow > 40) {
237         let remainingFieldsInGoal = piece + diceThrow - startField;

239         // Überprüfen ob die verbleibende Zahl zu ziehen klein genug ist
            // Falls sie zu gro  ist, verfällt der Zug für diesen Stein
241         if (remainingFieldsInGoal <= 4 && remainingFieldsInGoal > 0) {
```

```
            for (let field of freeFields) {
              if (field == remainingFieldsInGoal) {
                goalFields[mark - 1][remainingFieldsInGoal - 1] = mark;
                board[piece] = 0;
                break check;
              }
            }
          }
          continue;
        }

        // Überprüfen,ob das Feld besetzt ist...
        if (board[piece + diceThrow] != 0) {
          // falls ja, ...
          if (board[piece + diceThrow][0] == mark) {
            // ... wird bei Besetzung durch seine eigene Figur die Schleife fortgesetzt
, da kein Zug mit dieser Figur möglich ist
            continue;
          } else {
            // ... wird bei Besetzung durch eine gegnerische Figur geschlagen
            board[piece + diceThrow] = [mark, board[piece][1] + diceThrow];
            board[piece] = 0;
            leftPieces[2 - mark]++;

            break check;
          }
        } else {
          board[piece + diceThrow] = [mark, board[piece][1] + diceThrow];
          board[piece] = 0;
          break check;
        }
      } else {
        // Felder, die man ziehen muss nach Erreichen des "Ende" des Spielfelds
        let remainingFields = piece + diceThrow - 40;

        // Falls die übrigen Felder nicht das Zielfeld überschreiten, als normalen Zug
ausführen
        if (remainingFields <= startField) {
          // Überprüfen,ob das Feld besetzt ist...
          if (board[remainingFields] != 0) {
            // falls ja, ...
            if (board[remainingFields][0] == mark) {
              // ... wird bei Besetzung durch seine eigene Figur die Schleife
fortgesetzt, da kein Zug mit dieser Figur möglich ist
              continue;
            } else {
              // ... wird bei Besetzung durch eine gegnerische Figur geschlagen
              board[remainingFields] = [mark, board[piece][1] + diceThrow];
              board[piece] = 0;
              leftPieces[2 - mark]++;

              break check;
            }
          } else {
            board[remainingFields] = [mark, board[piece][1] + diceThrow];
            board[piece] = 0;

            break check;
          }
        } else {
          let remainingFieldsInGoal = remainingFields - startField;

          if ((remainingFieldsInGoal <= 4) & (remainingFieldsInGoal > 0)) {
            for (let field of freeFields) {
              if (field == remainingFieldsInGoal) {
                goalFields[mark - 1][remainingFieldsInGoal - 1] = mark;
                board[piece] = 0;
                break check;
              }
            }
          }
          continue;
        }
```

```
        }
      }
    }

    // Falls die Zählvariable z == pieces.length, bedeutet das, dass man keine der
    Figuren auf
    // dem Spielfeld (ausgeschlossen der Zielfelder) ziehen kann
    // ==> Falls man mit keinem der Figuren auf dem Zielfeld ziehen kann, wird gar nichts
     gezogen
    control: if (z == pieces.length) {
      var i;
      var x = 0;
      for (i = 0; i < 4; i++) {
        // Falls das derzeitige Feld nicht belegt ist, skippen, da nichts gezogen werden
    kann
        if (goalFields[mark - 1][i] != mark) {
          continue;
        }

        // Berechnen freier Felder nach einem belegten Zielfeld
        x = 0;
        while (true) {
          if (goalFields[mark - 1][x + i + 1] == mark || x + i + 1 > 3) {
            break;
          } else {
            x++;
          }
        }

        if (x > 0) {
          if (diceThrow <= x) {
            goalFields[mark - 1][x + i] = mark;
            goalFields[mark - 1][i] = 0;

            break control;
          }

          // Die Schleife kann abgebrochen werden, da die Anzahl der freien Felder nicht
    überboten werden
          //==> (X0X0) (Anzahl 1), (X00X, 0X00) (Anzahl 2), (X000) (Anzahl 3)
          break;
        }
      }

      // Falls die Felderanzahl nicht mehr gewürfelt werden kann, kann man überprüfen
      // ob der Spieler "feststeckt", also ausser durch eine Interaktion des Gegners
      // sich nicht mehr bewegen kann
      if (
        !dice
          .slice(1, dice.length)
          .some(
            (y) =>
              (+y == x && x != 0) ||
              freeFields.includes(+y - (40 - board[pieces[0]]?.[1])) ||
              (40 - board[pieces[0]]?.[1] >= +y &&
                0 < 40 - board[pieces[0]]?.[1])
          ) &&
        leftPieces[mark - 1] == 0
      ) {
        isStuck[mark - 1] = true;
      }
    }

    // Falls der Spieler eine 6 gewürfelt hatte, ist er immernoch dran und darf daher
    erneut würfeln
    if (diceThrow != 6) {
      turnNumber++;
    }

    // Überprüfen, ob Spieler gewonnen hat
    let goalFieldsFull = !goalFields[mark - 1].some((x) => x == 0);

    if (goalFieldsFull) {
```

```
      return mark;
    }

    // Falls beide Spieler nicht mehr ziehen können sollten ==> Unentschieden
    if (!isStuck.some((X) => X == false)) {
      return 0;
    }
  }
}

// Gibt eine zufällig ausgewählte Zahl, ausgehend der Seiten eines Würfels, zurück
// Da es sich um ein Laplace-Experiment handelt und es gleichwahrscheinlich ist jede Seite zu würfeln, muss man hier nicht die Wahrscheinlichkeiten
// der einzelnen würfelbaren Zahlen ausrechnen, sondern zufällig eine Zahl zwischen 1 und der Seitenanzahl bestimmen und die Zahl dieser "Seite" zurückgeben
function simulateDiceThrow(dice) {
  let sides = dice[0];

  let rndmNum = Math.floor(Math.random() * sides) + 1;

  return +dice[rndmNum];
}
```