(Dieser Artikel erschien in der Ausgabe 45 des Eulenspiegels[1], dem Magazin der Mathematik- und Informatikfachschaft des KIT.)

Würfeln mit Wubbel

Mit Methoden aus Mathematik und Informatik wird die stärkste Strategie für den Programmierwettbewerb des Linux-Magazins berechnet und bewiesen.

Der Wettbewerb

Im Linux-Magazin 09/10 rief Nils Magnus zu einem Programmierwettbewerb[2] auf. Man solle Programme einreichen, die in einem Würfelspiel gegeneinander antreten. Die Programmiersprache war dabei nicht vorgegeben, da die Programme über ein einfaches Netzwerkprotokoll gegeneinander antreten.

Die Regeln des Spiels sind einfach und ähneln denen des Spiels „Schweinerei“, in dem mit kleinen Gummischweinchen gewürfelt wird: Ein Spieler beginnt und würfelt. Würfelt er eine Sechs, so ist der andere Spieler dran, ansonsten zählt er die Augenzahl zu seinen Punkten. Er darf nun wieder würfeln (ROLL), wenn er will, um mehr Punkte zu erreichen, oder freiwillig abgeben (SAVE). Das ist irgendwann auch sinnvoll, denn sobald er eine Sechs würfelt, muss er abgeben und alle Punkte aus dieser Runde verfallen. Sicher sind die Punkte nur, nachdem man freiwillig abgegeben hat. So geht das immer hin und her, bis einer der Spieler 50 Punkte erreicht und damit gewonnen hat.

Stärke und Optimalität

Das Linux-Magazin begeleitete Rebecca Schwerdt und mich mit in den verregneten Urlaub. Da mein Laptop zwar dabei, mir aber verboten war, näherten wir uns dem Problem von der theoretischen Seite: Gibt es eine stärkste Strategie? Wenn ja, kann man sie auch angeben? Und wenn auch hier ja, kann man sie auch berechnen? Alle drei Fragen ließen sich mit „Ja“ beantworten. Die Werkzeuge hierzu sind ein bisschen Stochastik, Dynamische Programmierung und der Banachsche Fixpunktsatz.

Aber was heißt es denn nun, dass eine Strategie die (korrekter: eine) stärkste ist? Dass es keine stärkere gibt! Wir müssen also zwei Strategien vergleichen können. Naheliegend ist dabei, die Gewinnwahrscheinlichkeiten zu vergleichen. Ist die Wahrscheinlichkeit, dass Strategie 1 gegen Strategie 2 gewinnt, echt größer 50%, so kann man Strategie 1 als stärker bezeichnen. In unserem Fall muss man noch bedenken dass der, der das Spiel beginnt, einen Vorteil hat. Es ist also sinnvoll, vorher mit fairer Münze festzulegen, wer beginnt. Das ist gleichbedeutend damit, die Gewinnwahrscheinlichkeiten von einer Strategie als erster Spieler und als zweiter Spieler zu mitteln.

[1] `http://www.fsmi.uni-karlsruhe.de/Angebote/Publikationen/Eulenspiegel/`
[2] `http://wettbewerb.linux-magazin.de/`

Ich vermeide bewusst, von einer „optimalen Strategie“ zu reden. Darunter würden wir die Strategie verstehen, die im Programmierwettbewerb am ehesten gewinnt, und das ist nicht unbedingt die stärkste. Im Wettbewerb tritt jeder Teilnehmer gegen jeden an und am Ende zählen die meisten gewonnenen Spiele. Wenn also einige Teilnehmer nicht die stärkste Strategie fahren, kann man dies ausnutzen und gegen diese Spieler mit einer (wie auch immer) angepassten Strategie mehr Siege herausholen.

Strategien

Eine Strategie ist eine Vorschrift, wie man in den verschiedenen Spielsituationen reagiert. Um das mathematisch zu modellieren, sei $\mathcal{Z}$ die Menge der Spielzustände, wobei ein Spielzustand die Information zu den bisherigen Entscheidungen, den Würfen und damit auch den Punkteständen enthält. Eine Strategie ist dann eine Abbildung

$$S\colon \mathcal{Z} \to \{\mathrm{ROLL}, \mathrm{SAVE}\}.$$

Allerdings sind bei einem Würfel, der etwas auf sich hält, die Würfelergebnisse unabhängig. Das heißt, dass die konkreten Würfelergebnisse in einem Zustand $Z \in \mathcal{Z}$ für die Strategie egal sein sollten. Entscheidend ist nur der Punktestand: Wie viele Punkte e hat man selbst sicher, wie viele Punkte g hat der Gegner und wie viele unsichere Punkte u hat man in dieser Runde schon angesammelt. Damit schrumpft der Spielzustandsraum auf

$$\mathcal{Z}' := \{(e, g, u) \mid e, g, u \in \{0...50\}\}$$

(wobei hier noch ein paar ungültige Konfigurationen drin sind, was uns nicht weiter stören soll).

Angenommen, ein Spieler wüsste bereits alle Gewinnwahrscheinlichkeiten, also die Abbildung

$$P\colon \mathcal{Z}' \to [0,1],$$

die für jeden Zustand die Wahrscheinlichkeit angibt, dass der Spieler am Zug das Spiel gewinnen wird. Dann fiele die Entscheidung nicht mehr schwer: Man spielt so, dass die eigene Gewinnwahrscheinlichkeit maximiert wird:

$$S(e,g,u) = \begin{cases} \mathrm{SAVE}, & \text{falls } P_{\mathrm{SAVE}}(e,g,u) > P_{\mathrm{ROLL}}(e,g,u) \\ \mathrm{ROLL}, & \text{sonst,} \end{cases}$$

wobei sich die Gewinnwahrscheinlichkeiten nach einem SAVE bzw. ROLL wie folgt berechnen lassen: Wählt man SAVE, so ist die eigene Gewinnwahrscheinlichkeit gerade Eins minus der Gewinnwahrscheinlichkeit des Gegners. Wählt man ROLL, so ist in einem von sechs Fällen der Gegner dran, ansonsten erhöhen sich die eigenen unsicheren Punkte. In Formeln:

$$\begin{aligned} P_{\mathrm{SAVE}}(e,g,u) &= 1 - P(g, e+u, 0) \\ P_{\mathrm{ROLL}}(e,g,u) &= \frac{1}{6}\big(P(e,g,u+1) + P(e,g,u+2) + P(e,g,u+3) + \\ &\qquad P(e,g,u+4) + P(e,g,u+5) + (1 - P(g,e,0))\big) \end{aligned} \tag{1}$$

Terminationsvereinfachung

Wir brauchen also die Funktion P, um die Strategie angeben zu können. Man ist jetzt versucht, die vorangegangenen Gleichungen als Definition zu verwenden. Leider klappt das nicht ohne weiteres, daher wollen wir erst einmal die Spielregel etwas abändern: Man bekomme nun beim Wurf einer Sechs einen Gnadenpunkt. Damit ändern sich obige Formeln minimal ab:

$$\begin{aligned} P_{\text{SAVE}}(e,g,u) &= 1 - P(g, e+u, 0) \\ P_{\text{ROLL}}(e,g,u) &= \frac{1}{6}\big(P(e,g,u+1) + P(e,g,u+2) + P(e,g,u+3) + \\ &\qquad P(e,g,u+4) + P(e,g,u+5) + (1 - P(g, e+1, 0))\big) \end{aligned}$$

Was macht das für einen Unterschied? Jetzt kann man mit diesen Definitionen die Gewinnwahrscheinlichkeit für jeden Spielstand rekursiv berechnen: Entweder das Spiel ist schon vorbei und wir wissen, wer gewonnen hat, oder man ist am Zug und die Gewinnwahrscheinlichkeit ist die der besseren Spielentscheidung:

$$P(e,g,u) = \begin{cases} 0, & \text{falls } g \geq 50 \\ 1, & \text{falls } e+u \geq 50 \\ \max\{P_{\text{SAVE}}(e,g,u), P_{\text{ROLL}}(e,g,u)\}, & \text{sonst.} \end{cases} \tag{2}$$

Entscheidend ist hier, dass die Punktestände unaufhaltsam steigen und so diese rekursive Definition terminiert.

Allerdings ist das noch keine sehr effiziente Berechnungsmethode. Die vielfache Verzweigung sorgt bereits für 4 169 474 Funktionsaufrufe um $P(42,42,0)$ zu berechnen. Die Startgewinnwahrscheinlichkeit wird man so zu Lebzeiten nicht berechnen können.

Die Lösung ist eine Technik aus der Informatik, genannt Dynamische Programmierung. Hierbei legen wir eine Tabelle an, die für jeden Zustand $Z \in \mathcal{Z}'$ die Gewinnwahrscheinlichkeit enthält. In diese tragen wir erst einmal die trivialen Fälle ein: Für $g \geq 50$ ist sie null, für $e+u \geq 50$ ist sie Eins. Die restlichen Felder der Tabelle kann man nun „von rechts nach links“ auffüllen, da die Ergebnisse der rekursiven Aufrufe dann schon bekannt sind. Somit berechnen wir jeden Eintrag nur einmal und erhalten die Tabelle ungemein schneller.

Verschränkte Rekursion

Leider gibt es im Wettbewerb keinen Gnadenpunkt und die Gleichungen oben sind keine wohldefinierte Definition für P. Halten wir fest, dass wir, so es denn das Würfelglück erlaubt, von $(e,g,0)$ zu jedem (e,g,u) kommen können. Mit einer Sechs landen wir dann bei $(g,e,0)$ und somit jedem (g,e,u). Eine weitere Sechs führt uns wieder zurück. Für je zwei feste e und g hängen also die Gewinnwahrscheinlichkeiten in den Zuständen $\mathcal{Z}_{e,g} := \{(e,g,u), u \in \{0..50\}\} \cup \{(g,u,u), u \in \{0..50\}\}$ jeweils voneinander ab.

Der kanonische Weg, eine Lösung solch einer rekursiven Gleichung zu finden, ist ein Funktional $F\colon (\mathcal{Z}_{e,g} \to [0,1]) \to (\mathcal{Z}_{e,g} \to [0,1])$ zu definieren, dessen Fixpunkte $\{p \mid F(p) = p\}$ dann gerade die Gleichung erfüllen. In unserem Fall erhält man den Fixpunkt, indem man $F(p)(e,g,u)$ der rechten Seiten von Gleichung 2 und 1 gleichsetzt und für noch nicht berechnete Werte, also für $Z \in \mathcal{Z}_{e,g}$, nicht $P(Z)$ sondern $p(Z)$ verwendet:

$$F(p)(e,g,u) = \begin{cases} P(e,g,u), & \text{falls } (e,g,u) \notin \mathcal{Z}_{e,g} \\ \max\Big\{p(g,e+u,0), \\ \quad \frac{1}{6}\big(p(e,g,u+1) + \cdots + p(e,g,u+5) + (1 - p(g,e,0))\big)\Big\}, \\ & \text{falls } (e,g,u) \in \mathcal{Z}_{e,g} \end{cases}$$

Zu diesem Funktional suchen wir jetzt einen Fixpunkt. Ein schöner Satz dazu ist der Banachsche Fixpunktsatz:

> Sei X ein vollständiger metrischer nichtleerer Raum und die Abbildung $F\colon X \to X$ eine Kontraktion. Dann gibt es genau einen Fixpunkt $x_0 \in X$ und es gilt $\lim_{i\to\infty} F^i(x) = x_0$ für jeden Punkt $x \in X$.

Dieser gibt uns nicht nur die Existenz eines Fixpunktes, sondern sogar seine Eindeutigkeit und eine Berechnungsvorschrift – mehr wollen wir gar nicht. Also müssen wir die Voraussetzungen nachrechnen.

Die Menge $\mathcal{Z}_{e,g} \to [0,1]$ lässt sich auch als Produktraum $[0,1]^{100}$ auffassen – 50 Wahrscheinlichkeiten für (e,g,u) und 50 für (g,e,u). Das ist mit der Standardmetrik offensichtlich ein vollständiger metrischer Raum. Leider ist bezüglich dieser Metrik unser Funktional F keine Kontraktion. Auch andere naheliegende Normen wie l^1 und l^∞ funktionieren nicht. Wir müssen uns also hier mehr anstrengen.

Unnormale Norm

Unsere Norm ist von der Form

$$d(p,p') := \sum_{u=0}^{50} d_u \cdot |p(e,g,u) - p'(e,g,u)| + \sum_{u=0}^{50} d_u \cdot |p(g,e,u) - p'(g,e,u)|$$

mit geschickten Gewichten $d_u > 0$. Damit F eine Kontraktion ist, muss gelten (wobei wir die Terme mit (g,e,u) aus Gründen der Übersichtlichkeit ignorieren, abkürzend $\Delta_u := |p(e,g,u) - p'(e,g,u)|$ schreiben und $d_i = 0$ für $i < 0$ annehmen.)

$$\begin{aligned} d(F(p),F(p')) = {} & d_0 \cdot |F(p)(e,g,0) - F(p')(e,g,0)| \\ & + \sum_{u=0}^{50} d_u \cdot |F(p)(e,g,u) - F(p')(e,g,u)| \end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{6}\Big(d_0 \cdot \sum_{i=0}^{5} \Delta_i + \sum_{u=0}^{50} d_u\big(\Delta_0 + \sum_{j=1}^{5} \Delta_{u+j}\big)\Big) \\
&= \frac{1}{6}\Big((\sum_{i=0}^{50} d_i) \cdot \Delta_0 + \sum_{j=1}^{50} (d_{j-5} + \cdots + d_{j-1}) \cdot \Delta_j\Big) \\
&\overset{!}{<} d_0 \cdot \Delta_0 + \sum_{j=1}^{50} d_j \cdot \Delta_j = d(p, p')
\end{aligned}
$$

Durch Koeffizientenvergleich sieht man, dass also die Gleichungen

$$
\begin{aligned}
\frac{1}{6}\sum_{u=0}^{50} d_u &< d_0 \\
\frac{1}{6}(d_{u-5} + \cdots + d_{u-1}) &< d_u
\end{aligned}
$$

erfüllt sein müssen. Dies ist, was nach einigem Probieren gefunden wird, mit den folgenden Definitionen gegeben

$$
\begin{aligned}
d_0 &:= 1 \\
d'_u &:= \frac{1}{6}\left(\frac{7}{6}\right)^{u-1} && (1 \leq u \leq 5) \\
d'_u &:= \frac{1}{6}(7 \cdot d_{i-1} - d_{i-6}) && (u > 5) \\
\varepsilon &:= \frac{\frac{1}{6}\sum_{i=51}^{\infty} d'_i}{100} \\
d_u &:= d'_u + \varepsilon && (u > 0)
\end{aligned}
$$

was zu überprüfen den fleißigen Lesern, sofern existent, überlassen wird.

Die Voraussetzungen des Banachschen Fixpunktsatzes sind damit erfüllt und wir wissen, dass es eine Lösung zu unserem Problem gibt. Wir wissen auch, dass wir sie näherungsweise berechnen können, in dem wir die Einträge zu $\mathcal{Z}_{e,g}$ in der Tabelle beliebig füllen und so oft mittels F neu berechnen, bis sie sich nicht mehr wesentlich ändern – eine Aufgabe, die der Computer einem dankbarer Weise abnimmt.

Der Code

Damit genug theoretische Vorbereitung: Wir wollen endlich programmieren! Für einen Algorithmus wie diesen, der keine aufwendige Datenstrukturen benötigt und einfach nur viel rechnen muss, bietet sich C als Programmiersprache an.

```
#include <math.h>
#include <stdio.h>
```

```
#define GOAL 50
#define MIN(X,Y) ((X) < (Y) ? (X) : (Y))
#define EPS 1e−8

typedef double table [GOAL+1][GOAL+1][GOAL+1];

// Zwei Tabellen und Pointers darauf, um sie schnell auszutauschen
table table1, table2;
table *current, *next;

// Entspricht P_SAVE
double probSave(int ep, int gp, int u) {
  return 1−(*current)[gp][MIN(ep+u,GOAL)][0];
}

// Entspricht P_ROLL
double probThrow(int ep, int gp, int u) {
  return 1.0/6*(
    (*current)[ep][gp][MIN(u+1,GOAL)] +
    (*current)[ep][gp][MIN(u+2,GOAL)] +
    (*current)[ep][gp][MIN(u+3,GOAL)] +
    (*current)[ep][gp][MIN(u+4,GOAL)] +
    (*current)[ep][gp][MIN(u+5,GOAL)] +
    1−(*current)[gp][ep][0]);
}

// Entspricht P
double prob(int ep, int gp, int u) {
  if (ep==GOAL) return 1;
  if (gp==GOAL) return 0;
  return fmax(probSave(ep,gp,u),probThrow(ep,gp,u));
}

// Berechnet P
void wubbelStats() {
  current = &table1;
  next = &table2;

  // Initialisiere Tabellen
  for (int i=0; i<=GOAL; i++)
    for (int j=0; j<=GOAL; j++)
      for (int k=0; k<=GOAL; k++)
        (*current)[i][j][k] = (*next)[i][j][k] = 1;
```

```
    // Dynamische Programmierung
    for(int ps=GOAL*2; ps>=0; ps--){
      for(int ep=0; ep<=GOAL; ep++){
        int gp = ps-ep;
        // Basisfälle
        if (gp<0 || gp>GOAL || gp>ep) continue;
        // Fixpunktiteration, delta ist die Änderung im letzten Schritt
        double delta = 1;
        while (delta>EPS){
          delta=0;
          for (int u=0; u<=GOAL; u++){
            // Berechne next auf der Basis von current
            (*next)[ep][gp][u] = prob(ep,gp,u);
            (*next)[gp][ep][u] = prob(gp,ep,u);
            delta = fmax( fmax(
                  delta,
                  fabs((*current)[ep][gp][u]-(*next)[ep][gp][u])),
                  fabs((*current)[gp][ep][u]-(*next)[gp][ep][u]));
          }
          // Vertausche next und current
          table *help = current; current = next; next = help;
        }
      }
    }
}
```

Die Berechnung der Tabelle geht nun erstaunlich flott und ist in weniger als einer Sekunde abgeschlossen. Diese Tabelle kann dann in eine Textdatei geschrieben und von einem anderen Programm (bei uns in Python geschrieben) benutzt werden, um den Wettbewerb zu bestreiten.

Anschauungsmaterial

Wir könnten hier die Tabelle mit den Wahrscheinlichkeiten abdrucken, aber das würde wohl wenig erhellend sein. Statt dessen haben wir die Daten mit dem freien Visualisierungs-Programm Paraview analysiert. In Abbildung 1 ist die Fläche zwischen den ROLL- und SAVE-Zuständen dargestellt: In den Situationen hinter der Fläche wird gewürfelt, davor abgegeben. Die Farbe gibt die Gewinnwahrscheinlichkeit des Spielers am Zug an.

Es ist erstaunlich, dass die Entscheidung nicht monoton ist: Steht es etwa 0:24 würfelt man so lange bis man 16 oder 17 unsichere Punkte hat und gibt dann ab. Würfelt man aber darüber und erreicht 18, 19 oder 20 unsichere Punkte, sollte man noch einmal würfeln. Erst ab 21 unsicheren Punkte gibt man dann immer ab. Aufgrund dieser unebenen geometrischen Form heißt unser Beitrag zum Programmierwettbewerb auch „wubbel“.

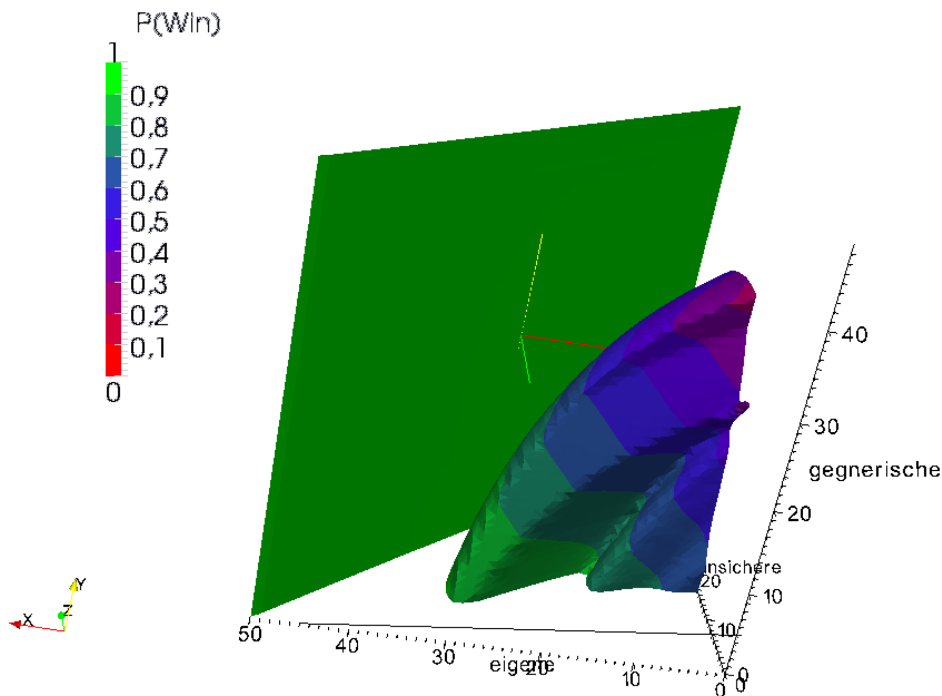

Abbildung 1: Die Entscheidungsgrenze

Hält sie was sie verspricht?

Somit haben wir eine Strategie berechnet, die recht plausibel konstruiert wurde. Einem Mathematiker genügt Plausibilität nicht: Es muss ein Beweis her, dass dies die stärkste Strategie ist! Dazu gehen wir zurück zu $\mathcal{Z}$, dem Raum der Spielzustände, und stellen ihn uns als Baum vor. Die Wurzel ist der Startzustand und die Kindknoten eines Knotens Z sind die Zustände nach einer Spielentscheidung, entweder Z_{SAVE} nach einem SAVE oder $Z_{\text{ROLL},i}$ nach einem ROLL mit Augenzahl i. Blätter in diesem Baum sind Endzustände, wenn einer gewonnen hat. Die Knoten des Baumes können wir mit der Gewinnwahrscheinlichkeit des ersten Spielers beschriften. An den Blättern steht dann entweder 1 oder 0.

Ist so ein Baum endlich, kann man eine Aussage über die Knoten induktiv von den Blättern bis zur Wurzel beweisen. Unser Baum ist es nicht, aber das lassen wir für einen Moment beiseite. Nehmen wir an, der Gegner weicht von unserer Strategie S ab und verwendet statt dessen S'. Die Aussage, die wir zeigen wollen, ist: „Im jedem Zustand $Z \in \mathcal{Z}$ ist unsere Gewinnwahrscheinlichkeit P gegen S größer als die neue Gewinnwahrscheinlichkeit P' gegen S'.“ Für die Blätter des Baumes gilt das offensichtlich. Sei also $Z \in \mathcal{Z}$ kein Blatt, und die Aussage gelte für alle Kinder. Dann folgt aus der Monotonie in der rekursiven Gleichung, dass die Aussage auch

für Z gilt:

$$\begin{aligned} P'(Z) &= \begin{cases} P'(Z_{\text{SAVE}}), & \text{falls } S'(Z) = \text{SAVE} \\ \frac{1}{6}\sum_{i=1}^{6} P'(Z_{\text{ROLL},i}), & \text{falls } S'(Z) = \text{ROLL} \end{cases} \\ &\le \begin{cases} P(Z_{\text{SAVE}}), & \text{falls } S(Z) = \text{SAVE} \\ \frac{1}{6}\sum_{i=1}^{6} P(Z_{\text{ROLL},i}), & \text{falls } S(Z) = \text{ROLL} \end{cases} \\ &\le \max\left\{P(Z_{\text{SAVE}}), \frac{1}{6}\sum_{i=1}^{6} P(Z_{\text{ROLL},i})\right\} \\ &= P(Z) \end{aligned}$$

Nun ist unser Baum nicht endlich, es könnte ja eine Sechs nach der anderen fallen und das Spiel endet nie. Somit ist unser Beweis noch nicht korrekt. Wir können aber statt P' eine „falsche“ Gewinnwahrscheinlichkeit P^* betrachten, die nach n Würfeln einfach auf eins gesetzt und das Spiel abgebrochen wird. Wir schneiden also die unendlich langen Zweige ab einer bestimmten Stelle ab und haben wieder einen endlichen Baum. Als neuen Basisfall haben wir jetzt „abgeschnittene Zweige“, aber da dort P' gleich eins ist, ist auch dort unsere Aussage wahr.

Wir wissen also

$$P(0,0,0) \le P^*(0,0,0) \le P'(0,0,0) + |P^*(0,0,0) - P'(0,0,0,0)| \le P'(0,0,0) + L_n,$$

wobei L_n die Wahrscheinlichkeit angibt, dass ein Spiel nach n Würfen noch nicht vorbei ist und somit den Fehler abschätzt. Nun geht $L_n \to 0$ für $n \to \infty$, da selbst ein Durchmarsch von 0 auf 50 Punkten in einem Zug mit Wahrscheinlichkeit 1 eintritt, wenn es man nur oft versucht. Damit erhält man wie gewünscht

$$P(0,0,0) \le P'(0,0,0),$$

das heißt wenn der Gegner eine andere Strategie verwendet als wir, steigt unsere Gewinnwahrscheinlichkeit. Was zu beweisen war.

Fazit

Falls also jemand fragen sollte, wofür Mathematik und Informatik denn nützlich sind, dann... naja, lassen wir das. Aber es hat Spaß gemacht und das Ergebnis ist auch befriedigend: Mit systematischen Vorgehen und mathematischer Herangehensweise konnte eine Lösung für das Problem gefunden werden, auf die man nur über Raten, Probieren oder Intuition nicht gekommen wäre.

Den Wettbewerb haben wir, wie abzusehen war, nicht gewonnen. Die Einzelergebnisse sind in mehreren Gigabyte an Protokollen und Ergebnistabellen veröffentlicht worden. Auf den ersten schnellen Blick ist die beobachtete Gewinnhäufigkeit unseres Bots plausibel, dies harrt aber noch der Überprüfung mit statischen Methoden. Auch wurde die These nicht überprüft,

dass der Bot gewonnen hat, der gegen die Masse der wenig starken Gegner die meisten Siege herausgeholt hat.

Mehr Ergebnisse und Visualisierungen für dieses und ähnliche Spiele finden sich auf der Website „Solving the Dice Game Pig: an introduction to dynamic programming and value iteration“[3], auf die später im Wiki des Programmierwettbewerbs verwiesen wurde. [Joachim Breitner]

[3] http://cs.gettysburg.edu/~tneller/nsf/pig/