Mathematisches Praktikum — Sommersemester 2018
Prof. Dr. Arnold Reusken — Sven Gross, Hauke Sass, Johannes Stemick

igpm Institut für Geometrie und Praktische Mathematik | RWTH AACHEN UNIVERSITY

Aufgabe 4

Bearbeitungszeit: zwei Wochen (bis Montag, 4. Juni 2018)

Mathematischer Hintergrund: Zwei-Personen-Nullsummenspiele und der MiniMax-Algorithmus

Elemente von C++: rekursive Funktionen, Klassendesign, Zufallszahlen, Netzwerk

Aufgabenstellung

Schreiben Sie ein Vier-Gewinnt-Programm, das in der Lage ist, die Vier-Gewinnt-Strategie der Praktikumsumgebung (mit Schwierigkeitsgrad 4) in einem Best-of-Five-Match zu schlagen. Implementieren Sie das beschriebene Netzwerkprotokoll, verbinden Sie sich mit einem Programm Ihrer Kommilitonen und spielen Sie ein Spiel gegeneinander.

Das Vier-Gewinnt-Spiel

„Vier Gewinnt“ wird von zwei Personen gespielt. Jeder Spieler besitzt eine genügend große Anzahl gleicher Spielsteine. Die Steine des einen Spielers sind gelb, die des anderen rot. Der Spieler mit den gelben Steinen beginnt. Beide Spieler werfen abwechselnd jeweils einen Stein in einen senkrecht vor ihnen stehenden Rahmen (*Spielbrett*) mit n Spalten. Jede Spalte kann insgesamt m Steine aufnehmen. Aufgrund der Schwerkraft fallen die Steine in der Spalte nach unten auf die unterste freie Position. Es gewinnt derjenige Spieler, dem es zuerst gelingt, vier seiner Spielsteine in eine Reihe zu bringen, entweder waagerecht, senkrecht oder diagonal. Sind alle Positionen besetzt, ohne dass dies einem Spieler gelungen ist, dann endet die Partie unentschieden. Bei einem Best-of-Five-Match gewinnt derjenige Spieler, der in fünf Spielen die meisten Siege erringt.

Zwei-Personen-Nullsummenspiele

„Vier Gewinnt“ gehört zu den sogenannten *Zwei-Personen-Nullsummenspielen*. Darunter versteht man solche Spiele, bei denen zwei Spieler P_1, P_2 abwechselnd ihre Züge machen, bis das Ende der Partie erreicht ist. Die Stellung am Ende der Partie hat für jeden Spieler P_i einen bestimmten Wert w_i, wobei die Zwei-Personen-Nullsummenspiele durch $w_1 + w_2 = 0$ charakterisiert sind. Der Wert w_i wird auch als *Gewinn* des Spielers P_i bezeichnet. Gewinner der Partie ist derjenige Spieler, für den $w_i > 0$ gilt. Er bekommt den Betrag w_i von dem anderen Spieler, für den $w_i < 0$ gilt, ausbezahlt.

Den Spielverlauf kann man noch etwas formalisieren: Sei dazu X die Menge aller möglichen *Stellungen*, die während des Spiels auftreten können. Wir nehmen an, dass in jeder Stellung $x \in X$ genau ein Spieler P_i mit dem nächsten Zug an der Reihe ist, es sei denn, es handelt sich um eine Endstellung. Die Menge X lässt sich dann aufspalten in $X = X_1 \cup X_2 \cup E$, wobei X_i die Menge der Stellungen ist, in denen der Spieler P_i an der Reihe ist, und E die Menge der Endstellungen.

Üblicherweise gehört zu jedem Spiel eine fest vorgegebene Anfangsstellung $x_0 \in X$. Für jede Stellung $x \in X$ sei N_x die Menge der möglichen Nachfolgestellungen, die durch einen gültigen Zug von x aus erreicht werden können. Ein *Zug* ist also nichts anderes als die Auswahl eines Elements x' aus der Menge N_x. Für $x \in E$ gilt selbstverständlich $N_x = \emptyset$. Unter der Voraussetzung, dass die Spieler immer abwechselnd ziehen, gilt $N_x \subset X_2 \cup E$ für alle $x \in X_1$ und $N_x \subset X_1 \cup E$ für alle $x \in X_2$.

Eine endliche Folge von Stellungen $(x_0, x_1, \ldots, x_k)$ mit $x_k \in E$ und $x_j \in N_{x_{j-1}}$ für $1 \leq j \leq k$ heißt eine *Partie*. Jede Partie beginnt also mit der Anfangsstellung x_0 und endet mit einer Endstellung $x_k \in E$. Im Allgemeinen ist nicht ausgeschlossen, dass Stellungen sich in einer Partie wiederholen, bei „Vier Gewinnt“ ist das jedoch nicht möglich (da die Anzahl der Steine im Spielbrett streng monoton wächst). Jeder Endstellung $x \in E$ wird nun für jeden Spieler P_i ein Gewinn $w_i(x)$ zugeordnet. Die Funktion w_i heißt *Gewinnfunktion* des Spielers P_i. Wie oben schon erwähnt, muss bei einem Zwei-Personen-Nullsummenspiel immer $w_1(x) + w_2(x) = 0$ gelten. Bei „Vier gewinnt“ kann w_i nur drei Werte annehmen: 1, falls man gewonnen hat, -1, falls man verloren hat, und 0 bei einem Unentschieden.

MiniMax-Algorithmus

Wie kann man, wenn man am Zug ist, vorgehen, um seinen Gewinn zu maximieren? Man muss für jeden der möglichen Züge entscheiden, wie „gut“ oder „schlecht“ er bzw. die Stellung ist, die man mit ihm erreicht, d. h., man muss die Stellungen bewerten können. Man wird dann denjenigen Zug wählen, der auf die Stellung mit der besten Bewertung (höchsten Gewinnerwartung) führt.

Formal kann man eine solche Bewertung als eine Funktion $f\colon X \to \mathbb{R}$ auffassen, die jeder Stellung $x \in X$ einen Wert $f(x)$ zuordnet. Am einfachsten ist die Situation bei den Endstellungen: Hier ist die Bewertung schon durch den Gewinn w vorgegeben. Das Problem ist jetzt nur noch, die Bewertung auf die anderen Stellungen auszuweiten. Betrachten wir dazu die Situation aus der Sicht von P_1:

Angenommen, die Stellung x soll bewertet werden und die Bewertungen $f(x')$ für alle Nachfolger $x' \in N_x$ seien bereits berechnet. Wäre der Spieler P_1 am Zuge, dann würde er den Zug machen, der auf die höchste Bewertung führt. Die Bewertung (Gewinnerwartung) dieser Stellung ist daher

$$f(x) = \max_{x' \in N_x} f(x').$$

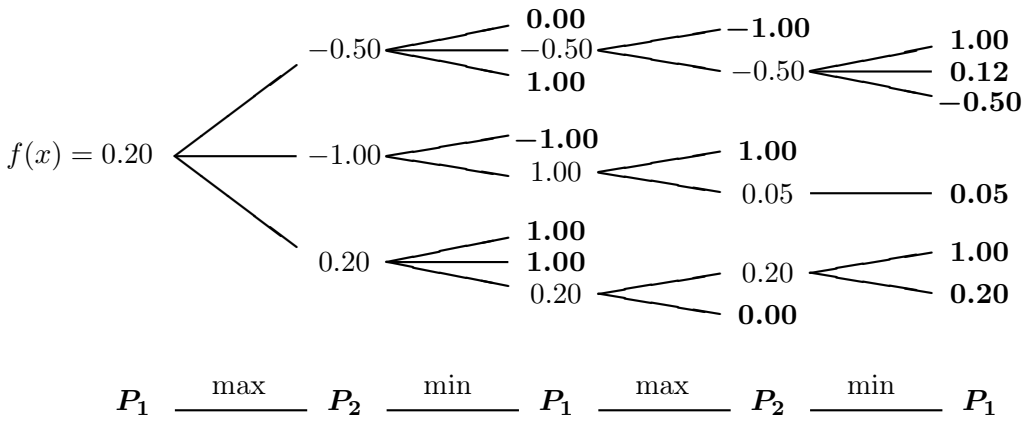
Wäre dagegen P_2 am Zuge, würde dieser Spieler den für sich günstigsten Zug machen, was bei Nullsummenspielen auf den schlechtesten Zug für P_1 hinausläuft. In diesem Fall liegt die Gewinnerwartung für P_1 daher bei

$$f(x) = \min_{x' \in N_x} f(x').$$

Die Bewertungen der x' könnte man wiederum aus ihren Nachfolgern bestimmen usw. bis man an den Endstellungen angekommen ist, wo die Bewertung durch w gegeben ist. Theoretisch lässt sich auf diese Weise rekursiv die Bewertung für alle $x \in X$ berechnen. In der Praxis scheitert dies an der Komplexität, da die Anzahl möglicher Stellungen meist exponentiell mit der Anzahl der Züge (in diesem Fall mn) wächst. Daher muss man die Anzahl der zu untersuchenden Nachfolger begrenzen. Dies kann geschehen, indem man die Anzahl der Züge begrenzt: Statt das Spielgeschehen immer bis zur Endstellung zu verfolgen, rechnet man nur einige Züge in die Zukunft (Look-Ahead) und bewertet die Stellungen dort auf eine heuristische Art durch eine Funktion $h\colon X \to \mathbb{R}$. Je besser diese Heuristik ist und je weiter man in die Zukunft rechnet, desto bessere Gewinnaussichten wird die Strategie haben.

Die Skizze soll die Arbeitsweise des MiniMax-Algorithmus mit 4 Zügen Look-Ahead verdeutlichen. Ausgehend von der Stellung $x \in X_1$ ist für jede Stellung y, die von x aus in höchstens vier Zügen erreichbar ist, der entsprechende MiniMax-Wert $f(y)$ eingetragen. Die MiniMax-Werte der Endstellungen und die MiniMax-Werte von Stellungen, die genau vier Züge von x entfernt sind, sind fett eingetragen, die restlichen wurden durch Minimum- bzw. Maximumbildung bestimmt, je nachdem, ob dann gerade P_1 oder P_2 am Zug ist.

In diesem Beispiel gibt es nur eine Stellung $y \in N_x$ mit maximaler Bewertung 0.20. Der Spieler P_1 würde hier also den dritten Zug wählen. Die MiniMax-Strategie für den Spieler P_2 unterscheidet sich nicht wesentlich von der Strategie für Spieler P_1, außer dass er mit der Minimumsbildung beginnt.



Die Heuristik der Praktikumsumgebung

In der Praktikumsumgebung wird eine recht einfache Heuristik h verwendet: Zu einer Stellung x werden nacheinander alle möglichen waagerechten, senkrechten und diagonalen Brettausschnitte betrachtet, wobei jeder Ausschnitt aus vier aufeinanderfolgenden Positionen besteht. Jede Position ist entweder leer oder mit einem gelben oder roten Stein belegt. Enthält ein Ausschnitt sowohl gelbe als auch rote Steine, so gibt es dafür keine Punkte. Enthält er zwei oder drei gelbe Steine, so wird eine bestimmte Punktzahl z_2 bzw. z_3 zum Wert der aktuellen Stellung hinzuaddiert. Enthält der Auschnitt zwei oder drei rote Steine, so wird die entsprechende Anzahl von Punkten subtrahiert. Befinden sich in einem Ausschnitt vier gleichfarbige Steine, so wird die komplette Stellung mit 1 bzw. -1 bewertet, je nachdem ob die Steine gelb oder rot sind. In diesem Fall brauchen die restlichen Brettausschnitte nicht mehr betrachtet zu werden.

Die Schnittstelle

Die Schnittstelle der Praktikumsumgebung, die Header-Datei `unit.h`, stellt Ihnen zunächst die folgenden drei Konstanten zur Verfügung:

```
extern const unsigned int AnzahlZeilen;

extern const unsigned int AnzahlSpalten;

extern const unsigned int AnzahlSpiele;
```

Die ersten beiden Konstanten geben die Dimension des Spielbretts an, die dritte die Anzahl der Spiele pro Match. Mit Hilfe der Funktion

```
void Start ( unsigned int Schwierigkeitsgrad );
```

wird die Praktikumsumgebung initialisiert und ein Grafikfenster mit der Darstellung des Spielbretts geöffnet. Mit dem Parameter `Schwierigkeitsgrad` stellen Sie die Spielstärke der Praktikumsumgebung ein. Indem Sie `Schwierigkeitsgrad=0` übergeben, können Sie die Praktikumsumgebung auch abschalten und über die Tastatur gegen Ihr eigenes Programm spielen. Mit der Funktion

```
int NaechsterZug ( int Spalte );
```

übergeben Sie der Praktikumsumgebung Ihren nächsten Zug, d. h. die Nummer der `Spalte`, die Ihr Programm für den nächsten Zug ausgewählt hat. Sie bekommen dann den nachfolgenden Zug der Praktikumsumgebung von der Funktion zurückgeliefert. Beachten Sie, dass nur Zahlen zwischen `0` und `AnzahlSpalten-1` gültige Züge darstellen können. Das Ende der aktuellen Partie erkennen Sie daran, dass ein Wert < 0 zurückgegeben wurde. Dann hat entweder Ihr Zug oder der Gegenzug der Praktikumsumgebung die Partie beendet.

Spielablauf

Ihr Programm soll gegen die Praktikumsumgebung insgesamt `AnzahlSpiele` Partien bestreiten. In den Partien mit ungerader Nummer hat Ihr Programm die gelben Steine und fängt an. Sie übergeben dann einfach Ihren ersten Zug mit der Funktion `NaechsterZug`. In den geradzahligen Partien spielt Ihr Programm mit den roten Steinen, und das Programm der Praktikumsumgebung beginnt. In diesem Fall müssen Sie die Funktion `NaechsterZug` mit `Spalte`$= -1$ aufrufen, und Sie bekommen den ersten Zug der Praktikumsumgebung zurück. Nach jeder Partie wird der aktuelle Spielstand angezeigt.

Notwendig zum Bestehen des Testats, ist das Erreichen eines Unentschiedens oder Sieg gegen die Praktikumsumgebung mit Schwierigkeitsgrad 4 nach fünf Partient. Sie können dazu versuchen, die oben beschriebene Strategie zu implementieren. Je nach Wahl der Konstanten z_2 und z_3 stehen Ihre Chancen dann ungefähr 50:50, dass Ihr Programm bei der Vorführung gewinnt. Sie sollten daher versuchen, die Strategie so zu verbessern, dass Ihr Programm sein Ziel sicher erreicht. Denkbar wäre eine Verbesserung der Heuristik oder ein größerer Look-Ahead. Allerdings ist die Zeit, die Sie für einen Zug zur Verfügung haben, begrenzt. Sollte Ihr Programm mehr als 2 Sekunden für einen Zug brauchen, wird es abgebrochen, und das Match gilt als verloren. Wenn es Ihnen dann gelingt, die Praktikumsumgebung auf Stufe 4 sicher zu schlagen, versuchen Sie es doch einmal gegen Stufe 5.

Zufallszahlen

Insbesondere am Anfang einer Partie kommt es häufig vor, dass in N_x mehrere Züge mit gleicher Bewertung existieren. Damit Ihr Programm in der gleichen Stellung nicht immer den gleichen Zug macht, sollten Sie Zufallszahlen verwenden, um einen der Kandidaten auszuwählen. Benutzen Sie dazu eine passende Zufallsverteilung aus dem `<random>`-Header der Standardbibliothek[1].

Denken Sie daran, den Zufallszahlengenerator mit einem zufälligen Wert zu initialisieren, um zu verhindern, dass er bei jedem Programmdurchlauf die gleichen Zufallszahlen erzeugt.

Implementierungsvorschlag

Hier ein paar Vorschläge zur Programmgestaltung, falls Ihnen keine bessere Idee kommen sollte:

- Entwerfen Sie eine Klasse `Spielbrett`, die ein Vier-Gewinnt-Spielbrett repräsentiert. Überlegen Sie sich neben einer geeigneten Datenstruktur einige nützliche Elementfunktionen, die einen sinnvollen Zugriff auf ein solches Objekt erlauben.
- Konzipieren Sie den MiniMax-Algorithmus als rekursive Funktion, d. h. eine Funktion, die sich selber aufruft. Diese sollte u. a. einen Parameter enthalten, der angibt, ob die Bewertung aus Sicht von P_1 oder P_2 erfolgen soll.

Wie Sie das Vier-Gewinnt-Programm letztendlich realisieren, bleibt natürlich Ihnen selbst überlassen.

Hinweise zur Kompilierung

Die Praktikumsumgebung benutzt zur Darstellung des Spielbrettes und für die Netzwerkfunktionalität (siehe unten) die Bibliotheken `X11` und `pthread`. Dies müssen Sie dem Linker mitteilen, sonst erhalten Sie eine Reihe von `undefined reference`-Fehlermeldungen. Dazu übergeben Sie die Linkeroptionen `-lX11 -pthread` an den Linker. Der besseren Übersicht wegen bietet es sich an, eine Variable in Ihrer Makefile zu erstellen

```
LIBS = -lX11 -pthread
```

und diese mittels `$(LIBS)` an den Linker weiterzureichen. Weiterhin müssen Sie den Code mindestens mit dem `C++11`-Standard kompilieren. Übergeben Sie dazu an den Compiler den Flag `-std=c++11`.

[1] `http://en.cppreference.com/w/cpp/numeric/random`

Multiplayer

Multiplayerspiele über das Internet sind heutzutage alltäglich. Die Praktikumsumgebung gibt Ihnen eine einfache Möglichkeit, Ihre Programme untereinander antreten zu lassen und so einen kleinen Einblick darin zu gewinnen, wie so etwas programmiert wird.

Damit sich ein Computer C (Client) mit einem Computer S (Server) verbinden kann, muss er dessen IP, also dessen Adresse im Netzwerk (beziehungsweise Internet) kennen. Sie können sich das wie einen Straßennamen vorstellen. Da ein Computer viele Netzwerkverbindungen gleichzeitig öffnen kann, gibt es zusätzlich zur IP noch einen so genannten Port. Die Analogie ist hier eine Hausnummer. Erst die Kombination IP und Port identifiziert einen Anschluss eindeutig. Allerdings kann sich C nicht mit einem beliebigen Port von S verbinden; S muss explizit an diesem Port auf eine Verbindung warten (klingeln reicht nicht, es muss auch jemand dort sein und öffnen).

Um erfolgreich eine Verbindung herzustellen, muss folgendes geschehen:

- S 'öffnet' einen Port P, um dort auf neue Verbindungen zu warten.
- C fragt bei S am Port P an, ob eine Verbindung hergestellt werden kann.
- S bestätigt nun die Verbindung und teilt C einen Port P' mit, an dem beide miteinander kommunizieren werden.

Bei typischen Serverandwendungen (z.B. der Server, der einer Website an Ihren Webbrowser schickt), kann S nun weitere Verbindungen annehmen und an andere, noch nicht belegte Ports P'' weiterleiten. Der Einfachheit halber unterstützt die Praktikumsumgebung nur eine 1-zu-1 Verbindung.

Nachdem eine Verbindung hergestellt wurde, können S und C beliebige Daten austauschen. Damit Ihre Programme allerdings untereinander kompatibel sind, müssen Sie ein einheitliches Übertragungsprotokoll zum Datenaustausch implementieren.

Das Multiplayerprotokoll

Sobald beide Computer miteinander verbunden sind, senden S und C immer abwechselnd ihren Zug. Damit dies eindeutig ist, muss klar sein, welcher Spieler anfängt. Dazu wird Ihnen bei der Verbindung von der Praktikumsumgebung eine Farbe nach der folgenden Regel zugewiesen: Der wartende Computer S hat die Farbe `gelb`, der sich verbindende Computer C hat die Farbe `rot`. Wie im Spiel gegen die Praktikums-KI beginnt der gelbe Spieler.

Der Spieler, der gerade am Zug ist, prüft zuerst, ob das Spiel bereits zu Ende ist. Sollte dies der Fall sein, sendet der aktive Spieler statt einer Spaltennummer die Konstante `SPIELENDE`, die Ihnen in der `unit.h` bereitgestellt wird.

Sollte ein Zug möglich sein, sendet der Spieler seine gewählte Spalte zwischen `0` und `AnzahlSpalten-1` an seinen Gegenspieler. Danach wartet er auf dessen Antwort.

Es ist wichtig, dass Daten, die aus einer unsicheren Quelle kommen (sprich aus dem Netzwerk oder Internet), immer überprüft werden! Erwarten Sie nie, dass die erhaltenen Werte immer zwischen `0` *und* `AnzahlSpalten-1` *sind, sondern überprüfen Sie dies! Fehlerhafte Überprüfung von Daten aus unsicherer Quelle ist eine der Hauptgründe für Sicherheitslücken und 'Hacks', bei denen massenhaft Nutzerdaten gestohlen werden. Diese Gefahr besteht hier zwar nicht unbedingt, aber sollte Ihnen bei Netzwerkprogrammierung immer im Hinterkopf bleiben.*

Die Multiplayerschnittstelle

In der Datei `vierGewinnt.cpp` wird Ihnen eine Methode

```
void NetzwerkMain()
```

bereitgestellt, die Ihnen über die Konsole die Verbindungsaufnahme zwischen zwei Programmen unterschiedlicher Teams erleichtert. Sollte eine Verbindung erfolgreich hergestellt worden sein, ruft `NetzwerkMain` die von Ihnen zu implementierende Methode

```
SpielStatus Netzwerkspiel( Feld MeineFarbe )
```

auf, die Ihnen als Variable die zugewiesene Spielerfarbe mitteilt. Nachdem das Spiel abgeschlossen wurde, soll diese Methode das entsprechende Mitglied der `SpielStatus`-Enumeration zurückgeben.

Die `C++11`-Enumeration (`enum class`) ist eine typsichere Enumeration, auf deren Werte sie zugreifen können, indem sie `SpielStatus::` vor den Namen des jeweiligen Wertes setzen. (*Warum sind typsichere enumerationen meistens eine bessere Wahl als gewöhnliche C++-enums?*)

Nachdem eine Verbindung erfolgreich hergestellt worden ist, können Sie folgende Funktionen aus der Datei `unit.h` benutzen:

Mittels

```
bool SendeZug( int meinZug )
```

können Sie Ihren gewählte Spalte oder die Konstante `SPIELENDE` (sollte das Spiel zu Ende sein) senden. Diese Funktion gibt `true` zurück, falls die Übertragung geklappt hat und `false`, falls es zu einem Fehler gekommen ist. In diesem Fall können Sie die Netzwerkverbindung als unterbrochen ansehen.

Die Funktion

```
int EmpfangeZug()
```

wartet solange, bis der Zug Ihres Kontrahenten über das Netzwerk eingetroffen ist. Sollte ein Verbindungsfehler aufgetreten sein, wird der Wert `VERBINDUNGSFEHLER` zurückgegeben. Sie müssen alle Rückgabewerte dieser Funktion überprüfen und können alle Werte ausserhalb von `0` und `AnzahlSpalten-1` sowie `SPIELENDE` als Verbindungsfehler interpretieren.

Eigene IP herausfinden

Unter Linux können Sie mit dem Befehl `ip -4 addr show` Ihre IP herausfinden. Der erste angegebene Wert ist immer die so genannte Loopback-Adresse `127.0.0.1`, mit der Sie eine Verbindung mit Ihrem eigenen Computer herstellen können. Damit können Sie zum Beispiel gegen eine zweite Instanz Ihres eigenen Programms spielen.

Die richtige IP für die Verbindung liegt im Bereich `134.130.160.13` bis `134.130.160.16`.

Die Ports, die Sie zur Verbindungsaufnahme nutzen können, liegen im Bereich `31000 - 31009` (inklusive). Sollte ein Port bereits belegt sein, versuchen Sie einen anderen im obigen Bereich zu wählen.

Notwendig zum erfolgreichen Testieren ist das Absolvieren eines Spiels ihres Programms gegen das Programm einer anderen Gruppe.

Literatur

[1] Manteuffel, K. und D. Stumpe: *Spieltheorie.* Teubner, 1979.

[2] Neumann, J. v. und O. Morgenstern: *Spieltheorie und wirtschaftliches Verhalten.* Physica-Verlag, 1973.

[3] Rauhut, B., N. Schmitz und E.-W. Zachow: *Spieltheorie.* Teubner, 1979.