Blatt 10

Abgabe bis: 19. Januar 2012

13. Vier gewinnt (10P)

In dieser Woche duerft ihr wieder ein Spiel programmieren: Vier gewinnt. Die Regeln findet ihr in der Wikipedia. Eure Implementierung soll modular sein. Sie soll aus der Spielelogik und aus separaten Spielern bestehen. Ein Makefile soll die Module zum fertigen Spiel zusammen bauen.

Beim Programmaufruf sollen zwei Spielernamen uebergeben werden. Diese waehlen die jeweiligen Spieler aus. Der Aufruf

```
four human karl
```

startet ein Spiel bei dem du gegen den Computergegner 'karl' antrittst. Bei anderen Aufrufen kann auch karl gegen karl antreten, oder zwei Menschen gegeneinander. Ueberlegt euch sinnvolles Verhalten fuer Aufrufe mit weniger als zwei Argumenten, und wie man erfaehrt, welche Bots es gibt.

Die Spiellogik soll von generischen Spielern ausgehen. Den naechsten Zug eines Spielers erfaehrt sie durch Aufruf einer bestimmten Funktion des Spielers (Funktionszeiger!). Der Zustand des Spielbrettes sollte dabei als Parameter uebergeben werden (notfalls kann er auch global gehalten werden).

Um einen weiteren (Computer-)Spieler 'Hans' einzubauen, sollen nur die zwei Dateien 'bot_hans.c', 'bot_hans.h' angelegt und wenige Zeilen in 'four.c' geaendert werden muessen. (Um die Abhaengigkeiten kuemmert sich dann `gcc-makedepend`.)

Orientiert euch an dem Funktionen-Beispiel aus der Vorlesung, das ihr auch auf der Vorlesungswebsite findet. (Die statische Variante reicht aus.)

Ihr muesst zumindest einen menschlichen Spieler und einen (nicht-trivialen) Computerspieler implementieren. Da weitere Bots einfach hinzuzufuegen sind, koennt ihr euch natuerlich auch gleich ein paar mehr schreiben und diese gegeneinander antreten lassen. Wer sich unterfordert fuehlt, darf auch gerne dynamisches Nachladen von Bots einbauen. Stichwort: *dlopen()*.

Zum Einreichen eurer Loesung:

```
submit ss1 13 team [notes] makefile four.c four.h
human.c human.h bot_karl.c bot_karl.h [utils.c]
[utils.h]
```

Wichtig: Euer Makefile muss 'makefile' -- mit kleinem 'm' heissen. Die Dateien 'utils.c' und 'utils.h' sind optional. Ihr koennt sie nutzen, falls ihr Bedarf dafuer habt.

Beispiel:

```
$ ./four human karl

|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
  0   1   2   3   4   5   6
Player ‘X’: 3

[...]

|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   | X |   | O |   |   |   |
|   | O | X | X | O |   |   |
| X | O | O | X | X | O |   |
+---+---+---+---+---+---+---+
  0   1   2   3   4   5   6
Player ‘X’:
```

Fragen zur Selbstkontrolle

- Welche Aufgaben uebernimmt der Praeprozessor bei C?
- Nenne ein paar Anwesungen an ihn.
- Wozu bedingte Uebersetzung? Diskussion!
- Wozu modularisiert man?
- Beschreibe Modularisierung bei C?
- Erklaere die Unterschied von Deklaration und Definition.
- Was bedeutet `extern`? Und was `static`?
- Was sind Header-Dateien?
- Wie laeuft der Zusammenbau eines C-Programmes im Detail ab?
- Erklaere Funktionszeiger an einem Beispiel.